MATrA: Meta-modelling Approach to Traceability for Avionics

Paul Andrew James Mason

Submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

Department of Computing Science
University of Newcastle

March 2002
Abstract

Traceability is the common term for mechanisms to record and navigate relationships between artifacts produced by development and assessment processes. Effective management of these relationships is critical to the success of projects involving the development of complex aerospace products.

Practitioners use a range of notations to model aerospace products (often as part of a defined technique or methodology). Those appropriate to electrical and electronic systems (avionics) include Use Cases for requirements, Ada for development and Fault Trees for assessment (others such as PERT networks support product management). Most notations used within the industry have tool support, although a lack of well-defined approaches to integration leads to inconsistencies and limits traceability between their respective data sets (internal models).

Conceptually, the artifacts produced using such notations populate four traceability dimensions. Of these, three record links between project artifacts (describing the same product), while the fourth relates artifacts across different projects (and hence products), and across product families within the same project.

The scope of this thesis is to define a meta-framework that characterises traceability dimensions for aerospace projects, and then to propose a concrete framework capturing the syntax and semantics of notations used in developing avionics for such projects which enables traceability across the four dimensions. The concrete framework is achieved by exporting information from the internal models of tools supporting these notations to an integrated environment consisting of: i) a Workspace comprising a set of structures or meta-models (models describing models) expressed in a common modelling language representing selected notations (including appropriate extensions reflecting the application domain); ii) well-formedness constraints over these structures capturing properties of the notations (and again, reflecting the domain); and iii) associations between the structures. To maintain consistency and identify conflicts, elements of the structures are verified against a system model that defines common building blocks underlying the various notations.

The approach is evaluated by (partial) tool implementation of the structures which are populated using case study material derived from actual commercial specifications and industry standards.
Acknowledgements

I owe a debt of gratitude to my supervisor Dr. Amer Saeed for shaping my ideas and for his continued help, guidance and encouragement throughout. Thanks are also due to Amer’s wife and family for understanding my demands on his time, especially during the period of distance supervision.

With Amer based in the South East for the last two years, it befell Professor Tom Anderson to provide me with managerial supervision in Newcastle. Thank you Tom for ensuring I saw things through.

Further thanks go to my examiners, Dr. Julian Johnson and Dr. Nick Rossiter for their valuable comments and observations.

I would also like to thank all at the Centre for Software Reliability for providing a convivial and stimulating research environment.

Thanks are due to BAE SYSTEMS for their sponsorship and intellectual support and for affording me the opportunity to evaluate my research in an industrial context. I am also grateful to the EPSRC for their financial support.

A special mention goes to my dad James for his love and generosity and to my Uncle Bill who has encouraged me throughout. Thank you both.

And to my mum Joyce whose strong will and resilience I inherited; you are never far from my thoughts.

Finally to Chompoo, for the times we had and for being there almost every day during the past four and a half years. You gave me perspective, encouragement and so much more. You tolerated my grumpiness, not to mention relentless and often clu-el (sorry cruel) sarcasm. And what’s more, you did it with a smile. Khob Khun Krub, babe!!!
# Table of Contents

List of Figures i  
List of Tables v  
Author's Declaration vi  

## Chapter One: An Introduction to Traceability

- **1.1 Introduction**  
- **1.2 Aerospace Industry Characteristics: The Need for Traceability**  
- **1.3 Thesis Argument**  
- **1.4 Survey of Traceability Literature**  
  - **1.4.1 A Brief History of Traceability**  
  - **1.4.2 Literature Definitions**  
  - **1.4.3 Author's Definitions**  
  - **1.4.4 On Formal Definitions of Traceability**  
  - **1.4.5 Dimensions For Traceability**  
  - **1.4.6 Drivers For Traceability**  
    - **1.4.6.1 The Emergence of Requirements Engineering (RE)**  
    - **1.4.6.2 Increased use of Quality Management & Compliance Frameworks**  
      - **1.4.6.2.1 Aerospace Recommended Practice (ARP) 4754 & 4761**  
      - **1.4.6.2.2 On Traceability Requirements for ARP 4754 & 4761**  
- **1.5 Chapter Summary**  
- **1.6 Thesis Structure**

## Chapter Two: Techniques and Tools for Traceability

- **2.1 Introduction**  
- **2.2 Traceability Techniques**  
  - **2.2.1 Cross-Referencing**  
    - **2.2.1.1 Foundations**  
    - **2.2.1.1.1 Lower-Order Cross-Referencing**  
    - **2.2.1.1.2 Higher-Order: Multiple Relations**  
    - **2.2.1.1.3 Higher-Order: Multiple Artifacts**  
  - **2.2.1.2 Traceability Enhancements**  
  - **2.2.1.3 Traceability Applications**  
  - **2.2.1.4 Evaluation**  
- **2.2.2 Conceptual Data Modelling**  
  - **2.2.2.1 Foundations**  
    - **2.2.2.1.1 Structural Constructs**  
    - **2.2.2.1.2 Constraints**  
    - **2.2.2.1.3 Operators**  
  - **2.2.2.2 Representative Conceptual Data Models**  
    - **2.2.2.2.1 Relational Data Model**  
    - **2.2.2.2.2 Entity-Relationship Model**  
    - **2.2.2.2.3 Object-Role Modelling (ORM)**
2.2.2.2.4 GEM 42
2.2.2.2.5 IDEFIX 44
2.2.2.2.6 DAPLEX 44
2.2.2.2.7 EXPRESS/EXPRESS-G 45
2.2.2.2.8 O-Telos 45
2.2.2.2.9 Unified Modelling Language (UML) & Object Constraint Language (OCL) 47
2.2.2.2.10 Other Conceptual Data Models 52

2.2.3 Traceability Enhancements 52
2.2.4 Traceability Applications 53
2.2.5 Evaluation 54

2.3 Tool Support For Traceability 55

2.3.1 General Purpose Tools 56
2.3.1.1 Hypertext 56
2.3.1.2 General Purpose Database Management Systems 57
2.3.1.2.1 Relational Database Management Systems (RDMS) 57
2.3.1.2.2 Object-Oriented Database Systems (OODS) and Deductive Database Systems (DDS) 58
2.3.1.2.3 Deductive Object-Oriented Database Systems (DOODS) 59

2.3.2 Commercial Traceability Tools 60
2.3.2.1 DOORS (Dynamic Object Oriented Requirements System) 60
2.3.2.2 RTM (Requirements & Traceability Management) 62
2.3.2.3 RDD-100 63
2.3.2.4 Additional Traceability Tools 64

2.3.3 Comment on Tools 65

2.4 Techniques and Tools for MATrA 66

2.5 Chapter Summary 66

Chapter Three: MATrA: Foundations and Fundamentals

3.1 Introduction 67

3.2 Foundations of MATrA 67

3.2.1 Novel Approaches to Theories Underlying Requirements Engineering (NATURE) 67
3.2.1.1 NATURE Influence 68

3.2.2 System Engineering Data Representation and Exchange Standardisation (SEDRES) 69
3.2.2.1 SEDRES Influence 70

3.2.3 Design Rationale Capture System (DRCS) 70
3.2.3.1 DRCS Influence 72

3.3 Fundamentals of MATrA 73

3.3.1 MATrA Fundamentals Overview 73
3.3.2 Notation Dependent Traceability Structures (Meta-models) 74
3.3.3 MATrA Systems Engineering Notation Meta-class Model 75
3.3.4 ‘tool2matra’ Mapping Function 78
### Table of Contents

#### 3.3.5 Product Data Synthesis (PDS)
- 3.3.5.1 Build Elements 80
- 3.3.5.2 Build Associations 81

#### 3.3.6 MATrA Framework Model
- 3.3.6.1 Aerospace Traceability Entity 84
- 3.3.6.2 Aerospace Build Entity 85
- 3.3.6.3 Aerospace Engineering Entity 85
  - 3.3.6.3.1 Aerospace Engineering Object 85
  - 3.3.6.3.2 Aerospace Engineering Association 85
- 3.3.6.4 Aerospace Management Entity 86
  - 3.3.6.4.1 Aerospace Engineering Project 86
  - 3.3.6.4.2 Product Data Synthesis 86
  - 3.3.6.4.3 Traceability Workspace 87
  - 3.3.6.4.4 Aerospace Link Entity 87

#### 3.3.7 Circuit Diagrams: A Meta-model Worked Example
- 3.3.7.1 Motivation 88
- 3.3.7.2 Concepts 88
- 3.3.7.3 Circuit Diagram Structure (Meta-model) 89
  - 3.3.7.3.1 Specification of Circuit Diagram Meta-model in UML 89
  - 3.3.7.3.2 OCL Constraints over Circuit Diagram Meta-model 91
  - 3.3.7.3.3 O-Telos Implementation of Circuit Diagram Base Classes 92
- 3.3.7.4 Circuit Diagram Example 92
- 3.3.7.5 Circuit Diagram Example Summary 94

#### 3.4 Chapter Summary 94

#### Chapter Four: Structuring Development Artifacts

##### 4.1 Introduction 95

##### 4.2 MATrA Natural Language Structure 95
- 4.2.1 Introduction 95
- 4.2.2 Motivation 95
- 4.2.3 Tracing Textual Artifacts: A Natural Language Structure 95
  - 4.2.3.1 Concepts 95
  - 4.2.3.2 Meta-Model Definitions 97
    - 4.2.3.2.1 MATrA Natural Language Structure Meta-model 97
    - 4.2.3.2.2 OCL Constraints 99
    - 4.2.3.2.3 O-Telos Implementation of MNLS Base Classes 100
  - 4.2.3.3 Worked Example 101
- 4.2.4 Summary 103

##### 4.3 User Centred Requirements Structure 104
- 4.3.1 Introduction 104
- 4.3.2 Motivation 104
  - 4.3.2.1 Use Case Overview 104
  - 4.3.2.2 Scenarios Overview 106
  - 4.3.2.3 Message Sequence Chart Overview 107
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3.3 Tracing User Centred Requirements in MATrA: UCRS - An Integrated Use Case &amp; Interaction Structure</td>
<td></td>
</tr>
<tr>
<td>4.3.3.1 Concepts</td>
<td></td>
</tr>
<tr>
<td>4.3.3.1.1 UCRS Notation</td>
<td>108</td>
</tr>
<tr>
<td>4.3.3.2 User Centred Requirements Structure (Definition)</td>
<td></td>
</tr>
<tr>
<td>4.3.3.2.1 User Centred Requirements Structure Meta-model</td>
<td>111</td>
</tr>
<tr>
<td>4.3.3.2.2 OCL Constraints over the User Centred Requirements Structure Meta-model</td>
<td>111</td>
</tr>
<tr>
<td>4.3.3.2.3 O-Telos Base Classes for User Centred Requirements Meta-model</td>
<td>112</td>
</tr>
<tr>
<td>4.3.3.3 Use Case View</td>
<td></td>
</tr>
<tr>
<td>4.3.3.3.1 Use Case View Meta-model</td>
<td>112</td>
</tr>
<tr>
<td>4.3.3.3.2 OCL Constraints over Use Case View Meta-model</td>
<td>114</td>
</tr>
<tr>
<td>4.3.3.3.3 O-Telos Base Classes for Use Case View Meta-model</td>
<td>117</td>
</tr>
<tr>
<td>4.3.3.4 Interaction View</td>
<td></td>
</tr>
<tr>
<td>4.3.3.4.1 Interaction View Meta-model</td>
<td>117</td>
</tr>
<tr>
<td>4.3.3.4.2 OCL Constraints over Interaction View Meta-model</td>
<td>118</td>
</tr>
<tr>
<td>4.3.3.4.3 O-Telos Base Classes for Interaction View Meta-model</td>
<td>125</td>
</tr>
<tr>
<td>4.3.4 Relationship to the Traceability Dimensions</td>
<td></td>
</tr>
<tr>
<td>4.3.5 Summary</td>
<td>132</td>
</tr>
<tr>
<td>4.4 Real-Time Network Specification Language Structure (Graphical)</td>
<td></td>
</tr>
<tr>
<td>4.4.1 Introduction</td>
<td>134</td>
</tr>
<tr>
<td>4.4.2 Motivation</td>
<td>134</td>
</tr>
<tr>
<td>4.4.2.1 RTN-SL Overview</td>
<td>135</td>
</tr>
<tr>
<td>4.4.2.2 RTN-SL Graphical Syntax</td>
<td>136</td>
</tr>
<tr>
<td>4.4.3 Tracing Real-Time Network Specifications in MATrA: An RTN-SLg Model</td>
<td></td>
</tr>
<tr>
<td>4.4.3.1 Concepts</td>
<td>138</td>
</tr>
<tr>
<td>4.4.3.2 RTN-SLg Meta-model Definitions</td>
<td>139</td>
</tr>
<tr>
<td>4.4.3.2.1 RTN-SLg Meta-model</td>
<td>139</td>
</tr>
<tr>
<td>4.4.3.2.2 OCL Constraints</td>
<td>142</td>
</tr>
<tr>
<td>4.4.3.2.3 O-Telos Implementation of RTN-SLg Base Classes</td>
<td>150</td>
</tr>
<tr>
<td>4.4.3.3 RTN-SLg Worked Examples</td>
<td>152</td>
</tr>
<tr>
<td>4.4.4 Relationship to the Traceability Dimensions</td>
<td>159</td>
</tr>
<tr>
<td>4.4.5 Summary</td>
<td>159</td>
</tr>
<tr>
<td>4.5 Towards a SPARK Ada Programming Language Structure</td>
<td></td>
</tr>
<tr>
<td>4.5.1 Introduction</td>
<td>160</td>
</tr>
<tr>
<td>4.5.2 Motivation</td>
<td>160</td>
</tr>
<tr>
<td>4.5.2.1 SPARK Ada Overview</td>
<td>161</td>
</tr>
<tr>
<td>4.5.3 Tracing Software Implementations in MATrA: A SPARK Ada Model</td>
<td></td>
</tr>
<tr>
<td>4.5.3.1 Concepts</td>
<td>163</td>
</tr>
<tr>
<td>4.5.3.1.1 A Modelling Philosophy towards the Object-Based Representation of String Grammars for Specification &amp; Code Level Languages</td>
<td>163</td>
</tr>
<tr>
<td>4.5.3.2 SPARK Ada Meta-model Definitions</td>
<td>166</td>
</tr>
<tr>
<td>4.5.3.3 SPARK Ada Meta-model: Worked Examples</td>
<td>190</td>
</tr>
<tr>
<td>4.5.4 Application to RTN-SL Textual Specifications</td>
<td>198</td>
</tr>
</tbody>
</table>
Table of Contents

4.5.4.1 Towards An RTN-SL (Textual) Structure 198
4.5.4.2 Meta-model Definitions 198
4.5.4.3 Specifying Activities and Ports using the RTN-SL Structure: A Worked Example 205
4.5.4.4 Relationship Between RTN Meta-Models 206
4.5.5 Relationship to the Traceability Dimensions 208
4.5.6 Summary 208

4.6 Chapter Summary 209

Chapter Five: Structuring Safety Assessment and Product Management Artifacts

5.1 Introduction 210
5.2 Fault Tree Analysis Structure 210
5.2.1 Introduction 210
5.2.2 Motivation 210
5.2.2.1 Fault Tree Analysis Overview 211
5.2.3 Tracing Safety Properties in MATrA: An FTA Model 214
5.2.3.1 Concepts 214
5.2.3.2 FTA Meta-model Definitions 215
5.2.3.2.1 FTA Meta-model 215
5.2.3.2.2 OCL Constraints 219
5.2.3.2.3 O-Telos Implementation of FTA Base Classes 223
5.2.4 Relationship to the Traceability Dimensions 225
5.2.5 Summary 226

5.3 Failure Modes and Effects Analysis Structure 227
5.3.1 Introduction 227
5.3.2 Motivation 227
5.3.2.1 Failure Modes and Effects Analysis Overview 227
5.3.3 Tracing Safety Properties in MATrA: An FMEA Model 229
5.3.3.1 Concepts 229
5.3.3.2 FMEA Meta-model Definitions 229
5.3.3.2.1 FMEA Meta-model 229
5.3.3.2.2 OCL Constraints 232
5.3.3.2.3 O-Telos Implementation of FMEA Base Classes 233
5.3.4 Relationship to the Traceability Dimensions 234
5.3.5 Summary 235

5.4 Programme Evaluation & Review Technique Structure 236
5.4.1 Introduction 236
5.4.2 Motivation 236
5.4.2.1 Programme Evaluation & Review Technique Overview 236
5.4.3 Tracing Programme Evaluation & Review Technique Networks in MATrA: A PERT Model 238
5.4.3.1 Concepts 238
5.4.3.2 PERT Meta-model Definitions 239
5.4.3.2.1 PERT Meta-model 239
5.4.3.2.2 OCL Constraints 240
# Table of Contents

5.4.3.2.3 O-Telos Implementation of PERT Base Classes .................................................. 241
5.4.3.3 PERT Worked Example ....................................................................................... 242
5.4.4 Summary ............................................................................................................... 245
5.5 The MATrA Configuration Model ............................................................................. 246
5.5.1 Introduction ........................................................................................................... 246
5.5.2 Motivation ............................................................................................................. 246
5.5.2.1 Configuration Management Overview ............................................................ 247
5.5.3 Across Revisions and Variants in MATrA: A Configuration Model for Tracing ... 249
      Evolutionary Development
      5.5.3.1 Concepts ......................................................................................................... 249
      5.5.3.2 MATrA Configuration Model Definitions ....................................................... 249
      5.5.3.2.1 MATrA Configuration Model .................................................................... 249
      5.5.3.2.2 OCL Constraints ......................................................................................... 253
      5.5.3.2.3 O-Telos Implementation of Configuration Model Base Classes ................. 256
      5.5.3.3 MCM Worked Example : Tracing Revisions for the Airbus A320-100/A320-200 Flight Control System
      5.5.3.3.1 Background ................................................................................................. 258
      5.5.3.3.2 Scenario ...................................................................................................... 258
5.5.4 Summary ............................................................................................................... 275
5.6 Chapter Summary .................................................................................................... 277

Chapter Six : Tracing Development and Assessment Artifacts

6.1 Introduction ............................................................................................................. 278
6.2 Case Study I: A Hypothetical Mission Planning System for the Hawk 100 and 200 Series Aircraft
  6.2.1 Mission Planning System Overview ................................................................. 279
  6.2.2 BASE Control Software Requirements and MATrA Representation .............. 280
     6.2.2.1 Use Case View .............................................................................................. 280
     6.2.2.1.1 Instantiation of Use Case View ................................................................. 284
     6.2.2.2 Interaction View .......................................................................................... 289
     6.2.2.2.1 Erase Cartridge - Normal Path ................................................................. 290
     6.2.2.2.2 Erase Cartridge - No Cartridge Present .................................................... 298
     6.2.2.2.3 Erase Cartridge - No Data on Cartridge .................................................. 301
     6.2.2.2.4 Erase Cartridge - Pilot Chooses Not to Erase Data .................................. 304
     6.2.2.2.5 Retrieve from Cartridge - Normal Path (Timing Fragment) .................... 307
     6.2.2.2.6 Choose Mission and Aircraft (Event Group Fragment) .............................. 310
  6.2.3 Trace Relations .................................................................................................. 313
     6.2.3.1 Instantiation of Trace Relations ................................................................... 315
  6.2.4 Summary ........................................................................................................... 316
6.3 Case Study II: A Brake System Control Unit for a Wheel Braking System of a Hypothetical Aircraft
  6.3.1 Scope of Case Study ........................................................................................... 317
  6.3.2 Overview of S18 Wheel Braking System and Brake System Control Unit ........ 317
  6.3.3 Preliminary System Safety Assessment - Brake System Control Unit ............... 318
6.3.3.1 Background on BSCU Design 318
6.3.3.2 Fault Tree Analysis - Preliminary 319
  6.3.3.2.1 Instantiation of Fault Tree Analysis Meta-model - Preliminary Fault Tree 321
6.3.4 System Safety Assessment - Brake System Control Unit 334
  6.3.4.1 Background on BSCU Power Supply Design 334
  6.3.4.2 Failure Modes and Effects Analysis 336
    6.3.4.2.1 Functional Failure Modes and Effects Analysis 336
    6.3.4.2.2 Instantiation of Functional Failure Modes and Effects Analysis Meta-model 337
    6.3.4.2.3 Piece-Part Failure Modes and Effects Analysis 339
    6.3.4.2.4 Instantiation of Piece-Part Failure Modes and Effects Analysis Meta-model 341
  6.3.4.3 Fault Tree Analysis - Updated 348
    6.3.4.3.1 Instantiation of Fault Tree Analysis Meta-model - Updated Fault Tree 349
6.3.5 Trace Relations 350
  6.3.5.1 Instantiation of Trace Relations 352
6.3.6 Summary 353
6.4 Chapter Summary 354

Chapter Seven: Conclusions

7.1 Introduction 355
7.2 Concluding Remarks 355
  7.2.1 Traceability for Avionics: Definitions & Drivers 356
    7.2.1.1 Contribution 356
  7.2.2 Modelling Concepts, Techniques & Tools 357
    7.2.2.1 Contribution 357
  7.2.3 MATrA: Towards A Concrete Framework for Traceability 357
    7.2.3.1 Contribution 358
  7.2.4 Meta-models for System Development 359
    7.2.4.1 Contribution 359
  7.2.5 Meta-models for Safety Assessment & Product Management 360
    7.2.5.1 Contribution 361
  7.2.6 Application of the MATrA Framework 362
  7.2.7 Overall Contribution 363
7.3 Limitations 363
  7.3.1 Absence of tool2matra Function 363
  7.3.2 Duplication of CASE Tool Data in Workspace 363
  7.3.3 Object Proliferation 364
7.4 Further Work 364
  7.4.1 Extension of Workspace Notations 364
  7.4.2 Application of MATrA to Other Safety-Critical Domains 365
  7.4.3 Survey of Domain Requirements for Trace Associations 365
  7.4.4 Enriching the Product Data Synthesis 365
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4.5</td>
<td>Specifying Requirements Using the MATrA Natural Language Structure</td>
<td>366</td>
</tr>
<tr>
<td>7.4.6</td>
<td>Systems Engineering Process Issues for MATrA</td>
<td>366</td>
</tr>
<tr>
<td>7.4.7</td>
<td>Investigation of an Inverse Mapping Function (<em>matra2tool</em>)</td>
<td>367</td>
</tr>
<tr>
<td>7.4.8</td>
<td>Contiguous Case Study Across All Dimensions</td>
<td>367</td>
</tr>
<tr>
<td>7.4.9</td>
<td>Investigation of Analysis Objectives</td>
<td>367</td>
</tr>
<tr>
<td>7.4.10</td>
<td>Use of <em>tool2matra</em> to Optimise Workspace Revisions</td>
<td>367</td>
</tr>
<tr>
<td>7.4.11</td>
<td>Confinement of Persistent Workspace to Trace Associations</td>
<td>368</td>
</tr>
<tr>
<td>7.4.12</td>
<td>Incorporation of Standards Knowledge into MATrA</td>
<td>368</td>
</tr>
<tr>
<td>7.4.13</td>
<td>Use of a Commercial tool as a Basis for Implementing MATrA</td>
<td>368</td>
</tr>
<tr>
<td>7.4.14</td>
<td>Re-expression of Structures in EXPRESS</td>
<td>368</td>
</tr>
<tr>
<td>7.5</td>
<td>Epilogue</td>
<td>369</td>
</tr>
</tbody>
</table>

References

Glossary

Appendix A

Appendix B

Appendix C

Appendix D

Appendix E
List of Figures

Figure 1.1 Inter/Intra, Macro/Micro Horizontal/Vertical Traceability Types 10
Figure 1.2 Horizontal, Vertical and Revision Traceability Dimensions 12
Figure 1.3 Variant Traceability 12
Figure 1.4 Overview of ARP 4754/4761 Safety Assessment Process 19
Figure 1.5 Example Relationship Between Aircraft and System FHA and Aircraft FTA 21
Figure 1.6 Steps in ARP 4754/4761 Safety Assessment Process 21
Figure 1.7 Information Flow and Traceability (context ARP 4761) 22
Figure 1.8 Relationship Between Meta (Dimensions) and Concrete (Workspace) Frameworks 23

Figure 2.1 Directed Graph, with Adjacency and Reachability Matrix Representations 26
Figure 2.2 Example Matrix Representation of Multiple Weighted Directed Graphs 28
Figure 2.3 Example Semantic Network 29
Figure 2.4 Example of a Defined Text Based Cross-Reference Format 30
Figure 2.5 An Example Quality Function Deployment (QFD) Matrix 32
Figure 2.6 An Example Project/Features Index (PFI) 33
Figure 2.7 Cross-Referencing Techniques of Modelling Notations 35
Figure 2.8 Examples of Conceptual Data Modelling Techniques 43
Figure 2.9 Example UML Class Diagram 49
Figure 3.1 Project NATURE Entity-Relationship Meta-Model 67
Figure 3.2 SEDRES Data Exchange Concept 69
Figure 3.3 DRCS Artifact Synthesis Structure 71
Figure 3.4 Rationale Components of DRCS 72
Figure 3.5 MATrA Systems Engineering Notation Meta-class Model 75
Figure 3.6 Inter-Tool Traceability Problem 78
Figure 3.7 Realising Inter-Tool Traceability using tool2matra 78
Figure 3.8 Product Data Synthesis Elements 80
Figure 3.9 MATrA Framework Model Elements 83
Figure 3.10 Aerospace Engineering Association (and example subtypes) 85
Figure 3.11 Aerospace Engineering Project 86
Figure 3.12 Product Data Synthesis 86
Figure 3.13 Traceability Workspace 87
Figure 3.14 Aerospace Link Entity Concept 87
Figure 3.15 Aerospace Link Entity 88
Figure 3.16 Generic Two Pin Electrical Component 89
Figure 3.17 Circuit Diagram Structure : Elements 90
List of Figures

Figure 3.18 Circuit Diagram Structure : Associations 90
Figure 3.19 Simple Electrical Circuit Diagram 95
Figure 4.1 MATrA Natural Language Structure : Elements 97
Figure 4.2 MATrA Natural Language Structure : Associations 98
Figure 4.3 MNLS Sample Population : Node Splitting Stage 1 102
Figure 4.4 MNLS Sample Population : Node Splitting Stage 2 102
Figure 4.5 Use Case Diagram of the AGA Sub-system 105
Figure 4.6 Message Sequence Chart for the Provide Guidance Data to Fin Controller Scenario (Normal Path) 107
Figure 4.7 MSC Timer Event Types 110
Figure 4.8 User Centred Requirements Structure (UCRS) 111
Figure 4.9 UCRS - Use Case View : Elements 113
Figure 4.10 UCRS - Use Case Model : Elements 113
Figure 4.11 UCRS - Use Case Model : Associations 114
Figure 4.12 UCRS - Interaction View : Elements 119
Figure 4.13 UCRS - Interaction Model : Associations 120
Figure 4.14 Scenario Event Natural Language Structure (SENLS) : Elements 122
Figure 4.15 Interaction Model - Communication Event : Elements and Associations 123
Figure 4.16 Interaction Model - Internal Action Event : Elements and Associations 124
Figure 4.17 Interaction Model - Timing Event : Elements and Associations 125
Figure 4.18 ASM Dynamic State Graphical Syntax 136
Figure 4.19 ASM Graphical Syntax for Composite Dynamic States 137
Figure 4.20 Example RTN-SLg Specification 137
Figure 4.21 RTN-SLg Structure : Elements 140
Figure 4.22 RTN-SLg Structure : Associations 141
Figure 4.23 Hypothetical Composite Dynamic State 155
Figure 4.24 RTN-SLg Specification for A Missile Target Tracking System 156
Figure 4.25 library_item schema 167
Figure 4.26 package_declaration schema 168
Figure 4.27 packageSpecification schema 169
Figure 4.28 inherit_clause schema 170
Figure 4.29 defining_program_unit_name schema 171
Figure 4.30 package_annotation schema 172
Figure 4.31 own_variable_clause schema 172
Figure 4.32 own_variable_list schema 173
Figure 4.33 initialization_specification schema 174
Figure 4.34 package_declarative_item schema 174
Figure 4.35 basic_declarative_item schema 175
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.36</td>
<td>basic_declaration schema</td>
<td>176</td>
</tr>
<tr>
<td>4.37</td>
<td>type_declaration schema</td>
<td>176</td>
</tr>
<tr>
<td>4.38</td>
<td>full_type_declaration schema</td>
<td>177</td>
</tr>
<tr>
<td>4.39</td>
<td>type_definition schema</td>
<td>178</td>
</tr>
<tr>
<td>4.40</td>
<td>enumeration_type_definition schema</td>
<td>179</td>
</tr>
<tr>
<td>4.41</td>
<td>subprogram_declaration schema</td>
<td>180</td>
</tr>
<tr>
<td>4.42</td>
<td>procedure_specification schema</td>
<td>180</td>
</tr>
<tr>
<td>4.43</td>
<td>parameter_profile schema</td>
<td>181</td>
</tr>
<tr>
<td>4.44</td>
<td>formal_part schema</td>
<td>181</td>
</tr>
<tr>
<td>4.45</td>
<td>parameter_specification schema</td>
<td>182</td>
</tr>
<tr>
<td>4.46</td>
<td>defining_identifier_list schema</td>
<td>183</td>
</tr>
<tr>
<td>4.47</td>
<td>procedure_annotation schema</td>
<td>183</td>
</tr>
<tr>
<td>4.48</td>
<td>moded_global_definition schema</td>
<td>185</td>
</tr>
<tr>
<td>4.49</td>
<td>entire_variable_list schema</td>
<td>186</td>
</tr>
<tr>
<td>4.50</td>
<td>entire_variable schema</td>
<td>186</td>
</tr>
<tr>
<td>4.51</td>
<td>dependency_relation schema</td>
<td>187</td>
</tr>
<tr>
<td>4.52</td>
<td>dependency_clause schema</td>
<td>188</td>
</tr>
<tr>
<td>4.53</td>
<td>imported_variable_list schema</td>
<td>189</td>
</tr>
<tr>
<td>4.54</td>
<td>activity schema</td>
<td>200</td>
</tr>
<tr>
<td>4.55</td>
<td>ports schema</td>
<td>201</td>
</tr>
<tr>
<td>4.56</td>
<td>port_defs schema</td>
<td>201</td>
</tr>
<tr>
<td>4.57</td>
<td>port_def schema</td>
<td>202</td>
</tr>
<tr>
<td>4.58</td>
<td>id_list schema</td>
<td>202</td>
</tr>
<tr>
<td>4.59</td>
<td>port_type schema</td>
<td>203</td>
</tr>
<tr>
<td>4.60</td>
<td>a_type_ref schema</td>
<td>204</td>
</tr>
<tr>
<td>4.61</td>
<td>RTN-SL Textual and RTN-SLg Meta-model Fragments</td>
<td>207</td>
</tr>
<tr>
<td>5.1</td>
<td>Example Fault Tree for Aircraft Wheel Braking System - investigating causes of Loss of All Wheel Braking (source APR 4761)</td>
<td>213</td>
</tr>
<tr>
<td>5.2</td>
<td>Fault Tree Analysis Structure : Elements</td>
<td>216</td>
</tr>
<tr>
<td>5.3</td>
<td>Fault Tree Analysis Structure : Associations</td>
<td>217</td>
</tr>
<tr>
<td>5.4</td>
<td>FMEA Structure : Elements</td>
<td>230</td>
</tr>
<tr>
<td>5.5a</td>
<td>Functional FMEA Structure : Associations</td>
<td>231</td>
</tr>
<tr>
<td>5.5b</td>
<td>Piece-Part FMEA Structure : Associations</td>
<td>231</td>
</tr>
<tr>
<td>5.6</td>
<td>Example of A Basic PERT Network</td>
<td>237</td>
</tr>
<tr>
<td>5.7</td>
<td>Example of A PERT Network - Time Analysis</td>
<td>238</td>
</tr>
<tr>
<td>5.8</td>
<td>Programme Evaluation &amp; Review Technique Structure : Elements &amp; Associations</td>
<td>239</td>
</tr>
<tr>
<td>5.9</td>
<td>PERT Network - Worked Example</td>
<td>242</td>
</tr>
</tbody>
</table>
List of Figures

Figure 5.10  Product First, Version First and Intertwined Models  248
Figure 5.11  MATrA Configuration Model (MCM)  250
Figure 5.12  A320-100 Product Data Synthesis Fragment - Properties  260
Figure 5.13  A320-100 Product Data Synthesis Fragment - Architecture  260
Figure 5.14  Decomposition of Inhibit Slat Retraction at High Angles of Attack (A320-100)  261
Figure 5.15  Decomposition of Inhibit Slat Retraction at Low Speed (A320-100)  262
Figure 5.16  Statechart Representation of Inhibit Slat Retraction  266
Figure 5.17  Linking the Traceability Workspace and PDS over Aerospace Link Entities  267
Figure 5.18  Example Configuration for A320-100  268
Figure 5.19  Impacts Associations Showing Dependency Propagation  271
Figure 5.20  Revising the A320-100 PDS (Including Partial Configuration Selection)  272
Figure 5.21  Tracing Change Over SucceedsAEO, AddedTo and Removes Associations: A Statechart Example (Logical Level)  273
Figure 6.1  Mission Planning System - Software Interaction  279
Figure 6.2  Erase Cartridge Use Case Diagram  281
Figure 6.3  Retrieve From Cartridge Use Case Diagram  282
Figure 6.4  Choose Mission and Aircraft Use Case Diagram  283
Figure 6.5  Hawk MPS : Services  284
Figure 6.6  MSC: Erase Cartridge - Normal Path  290
Figure 6.7  MSC: Erase Cartridge - No Cartridge  298
Figure 6.8  MSC: Erase Cartridge - No Data on Cartridge  301
Figure 6.9  MSC: Erase Cartridge - Pilot Chooses Not To Erase Data  305
Figure 6.10  Retrieve Data from Cartridge - Normal Path (Timing Fragment)  307
Figure 6.11  Choose Mission and Aircraft - New Mission From Open Missions (Event Group Fragment)  310
Figure 6.12  Exemplar Intra-Micro Horizontal Traceability Relations  314
Figure 6.13  ARP Assessment Process (Partial)  317
Figure 6.14i  BSCU Commands Braking in Absence of Brake Input Causing Inadvertent Braking - Preliminary Fault Tree (page 1)  320
Figure 6.14ii  BSCU Commands Braking in Absence of Brake Input Causing Inadvertent Braking - Preliminary Fault Tree (page 2)  320
Figure 6.14iii  BSCU Commands Braking in Absence of Brake Input Causing Inadvertent Braking - Preliminary Fault Tree (page 3)  321
Figure 6.15  BSCU Power Supply Block Diagram  335
Figure 6.16  BSCU + 5 Volt Power Supply Monitor Circuit Schematic  335
Figure 6.17  BSCU Commands Braking in Absence of Brake Input Causing Inadvertent Braking Updated Fault Tree (partial)  349
Figure 6.18  Exemplar Intra-Micro Horizontal & Vertical Traceability Relations  351
List of Tables

| Table 1.1 | Summary of Inter/Intra, Macro/Micro Vertical/Horizontal Traceability Types | 10 |
| Table 2.1 | PSSA Safety Requirements & Design Decisions Table | 34 |
| Table 3.1 | Comparison of MATrA Modelling Conventions : UML and O-Telos | 77 |
| Table 3.2 | Product Data Synthesis Build Associations | 81 |
| Table 4.1 | Example Scenario for the Autopilot Use Case | 107 |
| Table 4.2 | IDA Communication Protocols | 135 |
| Table 4.3 | Mapping Table : RTN-SL (Textual) to RTN-SLg Graphical Meta-model Elements | 207 |
| Table 5.1 | Fault Tree Event Types | 211 |
| Table 5.2 | Fault Tree Gate Types | 212 |
| Table 5.3 | ARP 4761 Functional and Piece-Part FMEA Contents | 228 |
| Table 5.4 | Functional FMEA Fragment for Break System Control Unit Power Supply | 228 |
| Table 5.5 | Activities & Relationships for Basic PERT Network | 237 |
| Table 6.1 | Chapter 6: Suggested Reading Paths | 278 |
| Table 6.2 | Summary of Traceability Relations (from figure 6.12) | 314 |
| Table 6.3 | Functional FMEA (Partial) of BSCU Power Supply | 337 |
| Table 6.4 | Piece Part FMEA (Partial) of BSCU Power Supply Monitor | 340 |
| Table 6.5 | Summary of Traceability Relations (from figure 6.18) | 351 |
Author's Declaration

All work contained within this thesis represents the original contribution of the author. However, some of the material presented has previously appeared in the following:-

- **Mason & Saeed, 1998**

- **Mason, 1998**
  Mason, P. - An Introduction to Traceability, University of Newcastle upon Tyne/BAe Dependable Computing Systems Centre Technical Note, TN DCSC/TN/98/04
Chapter 1    An Introduction to Traceability

1.1    Introduction

As systems complexity has increased, so too has the complexity of their development processes. In effect, this usually implies a greater number of sub-processes and hence more intermediate artifacts. Consequently, interest has grown in the study of relationships between these artifacts\(^1\). Traceability is the common term for mechanisms used to record and navigate such relationships.

In the remainder of Chapter One, we provide a context for work in this thesis. Specifically, the next section describes our domain of interest, namely the aerospace industry. We then introduce the thesis argument to be maintained throughout. Next, we consider traceability as a topic of interest among the wider software and systems engineering communities. This includes a brief history of the subject, defining terminology and forces ‘driving’ recent growth. Finally, we provide a synopsis of the overall thesis structure. The chapter is based on an extensive review of existing traceability literature which we underpin with views of practitioners from several business units within BAE SYSTEMS.

1.2    Aerospace Industry Characteristics: The Need for Traceability

This thesis considers traceability for aerospace systems engineering, in particular, traceability of artifacts describing systems in which (often safety-critical) functions are allocated to avionics (i.e. electrical and electronic equipment, such as communications and datalink systems, flight control systems, instruments, navigation systems, radar, mission planning systems and weapons). Certain characteristics make traceability an issue for most aerospace projects; this section considers five key characteristics influencing the need for traceability within that domain.

C1.    Product Life-cycle

A major difference between projects in the aerospace sector and those in other industries is ‘time-to-market’. Civil aircraft may be a decade or more in development, whilst military systems can be planned around twenty years hence. During this time, strategic and mission needs (and hence requirements) may be revised, sometimes instigated by sudden and unforeseen events (like the end of the cold war or the terrorist attacks of September 11, 2001). Moreover, total product life-cycles, particularly those for civil aircraft, can exceed thirty years. Inevitably over such a long period, manufacturers revise their designs to incorporate the likes of new technologies and changes in safety regulations. They may also develop variants of aircraft to accommodate specific market needs. Managing the ‘evolving product’ is therefore a significant problem on projects with protracted life-cycles.

Because few engineers are involved throughout the duration of such projects, a further and often overlooked problem is retention of knowledge capturing what is best described as ‘engineering

---

\(^1\) In keeping with the literature, we use the terms (trace) relation(ship), association and linkage interchangeably and synonymously throughout to describe any mechanism for linking artifacts produced by an engineering process.
judgement', i.e., the rationale for key development and assessment decisions. This is especially important when engineers must revisit the design of an in-service aircraft (normally following an accident) that has remained static for many years - an obvious example being Concorde which first flew in 1969, but which required a number of modifications following the Paris crash in July 2000.\(^2\)

Traceability may be viewed as the common denominator in managing these and other time related problems, both by structuring the relationships between multiple versions of artifacts, and by providing means to capture rationale relating to options, argumentation and intent.

C2. Process Characteristics
The development context for aerospace systems containing safety-critical functions or components can be described in terms of two concurrent processes (yielding two sets of artifacts) - development and assessment - each comprising a number of tightly coupled sub-processes. Development proceeds top-down through several iterations of requirements and design, gradually refining abstract aircraft level requirements, into systems or sub-systems and eventually detailed hardware and software component designs.\(^3\)

For each iteration, safety analysis determines whether the mission requirements contain any inherent hazards and what safeguards (safety requirements) are required to exclude them. Design proposals are also analysed to ensure they themselves do not introduce any hazards, that they satisfy the safety requirements and that they preserve the original intent (i.e., mission requirements). Following implementation and further safety analysis to ensure no new hazards have been introduced, assessment proceeds bottom up with the integration of components and verification that safety requirements have been satisfied at each level of abstraction.

A major problem with complex processes such as this is lack of visibility, especially where enactment extends over a protracted period (as described in C1). However, process visibility can be enhanced by effective traceability. In simplistic terms, if we consider all artifacts as inputs \((I)\) to, and outputs \((O)\) from the various sub-processes, then a traceability association between two artifacts, \(\text{from } i \in I \text{ to } o \in O\), may be regarded as an abstraction of the activity involved in producing \(o\) from \(i\) (where \(i\) and \(o\) denote requirements and designs, for example).

C3. Complex Integrated Systems
Aircraft functions such as automatic landing, auto-stabilisation and stall-protection are implemented by systems that are not only individually complex, but which involve a considerable number of interfaces between systems; e.g., a relatively simple automatic rudder control function with a yaw-damper to correct "Dutch-roll" characteristics will interconnect sensors, actuators, hydraulics, mechanical systems

---

\(^2\) In particular, engineers needed to consider rationale for location of the wheel assembly, relative to the engines and engine intakes. Interested readers are referred to Weir (2001) for more information.

\(^3\) Note the process we describe is simplified and takes no account, for example, of the complex interaction between technical definition and procurement activities.
An Introduction to Traceability

power plant and cockpit display units. Artifacts describing such systems are often similarly large and complex, as are the relationships between them. Therefore practitioners require means to support traceability not only within descriptions of individual systems, but also between these descriptions, in particular for impact analysis and change control purposes. Traceability can also help maintain consistency across such complex, high-volume data sets.

C4. Fault Tolerant Architectures

Aircraft systems having a direct bearing on safety employ fault tolerant architectures to guard against the risk of faults with the potential to cause a system failure. As the name suggests, these architectures allow continued and correct functioning in the presence of faults, where a fault is defined as some defect within a system (Storey, 1996).

Fault tolerance is achieved through redundancy, i.e., multiple modules that replicate means of achieving the same function. Fault tolerant architectures include those employing majority voting mechanisms which compare outputs from $N$ modules based on identical inputs. Where a single fault means the output from one module differs, voter output corresponds to the majority view. Simple configurations use triple modular redundancy (capable of tolerating a single fault), whilst greater use of redundant modules affords increased protection - termed $N$-modular redundancy (hardware) and $N$-version programming (software). Dynamic hardware redundancy schemes and software recovery blocks are further, comparable techniques where fault detection mechanisms switch operation to stand-by modules when a unit fails.

A corollary of using fault tolerant architectures is that multiple redundant modules yield an increase in development and assessment artifacts describing them, and hence even greater need for effective traceability so that these descriptions may be related back to their original requirement(s). In addition, traceability can provide means of managing assumptions made to justify claims associated with fault tolerant architectures (Popov et al., 2001).

C5. Certification

Finally, certification is normally a legal requirement before new aircraft or aircraft systems can enter operational use. Certification is undertaken by an appropriate regulatory body, with different authorities governing projects within particular aerospace sectors; e.g., in the UK, all civil aircraft require approval by the Civil Aviation Authority. One of the most important documents submitted in support of an application for certification is the safety case, a rigorous argument and supporting evidence stating why the system is safe for its intended use. A safety case may run to several volumes and so maintaining this body of evidence is often extremely difficult. Again, traceability can help practitioners alleviate such problems.

---

4 Dutch roll occurs when the aircraft has relatively strong lateral stability and weak directional stability. Readers are referred to Mair & Birdsall (1992) for more information.
5 Readers are referred to works by Laprie (1989), Leveson (1995) and Storey (1996) for definitions of the safety-critical vocabulary used in this thesis.
6 For more information on these and other approaches, the reader is referred to Anderson & Lee (1991).
An Introduction to Traceability

However, since certification often requires conformance to a particular standard, and since many standards demand traceability, then traceability can itself be a requirement for certification; e.g., DO-178b (EUROCAE, 1992), an international standard covering certification of software in airborne systems, requires that safety requirements should be traceable through various stages of development to specific elements of the low-level implementation.

This combination of factors demonstrate why traceability is of major significance to the aerospace industry and also why the traceability concerns of practitioners differ markedly from those in other sectors. We are unaware of any further work considering traceability from such a specialised and domain specific perspective.

1.3. Thesis Argument

Issue - We assert that characteristic properties of aerospace systems cause several distinct traceability problems, whilst magnifying those of a more ‘traditional’ nature. The aim is to propose a theoretical basis for a practical solution to these problems.

Position - Practitioners use a range of notations to model aerospace systems (often in conjunction with a process as part of a technique or methodology). For avionics, notations fall into two broad categories: those with a well-defined syntax and semantics (for example Circuit Diagrams and Ada) and those that are less rigorously defined but which offer flexibility as a result (such as Use Cases and Scenarios). Practitioners also require the ability to conduct safety assessment over these models using established techniques for hazard analysis (e.g., Fault Tree and Failure Modes and Effects Analyses) and to manage operations using further techniques for planning and control (notably Critical Path Analysis and PERT). Most of the above have tool support (normally bespoke or a commercial CASE tool), however a lack of well-defined approaches to integration limits traceability between their respective data sets.

We argue that traceability across tools can be achieved by exporting these data sets to an integrated environment consisting of: i) a Workspace comprising a set of structures or meta-models (literally, models describing models) capturing data elements for a representative set of development, assessment and product management notations; ii) a further structure capturing fundamental elements of the emerging product that maintains consistency within the Workspace (where ‘fundamental elements’ refers to system components, their functions and behaviour, etc.); iii) well-formedness constraints over these structures; and iv) associations and consistency constraints between the structures.

We also argue that development and assessment information populates four traceability dimensions, of which three record links between project artifacts (conceptualised as a cube), while the fourth relates artifacts across different projects (i.e., cubes) and across product families within the same project. Hence, the structures must provide coverage across four dimensions.

7 ‘Traditional’ traceability problems concern the following: what kind of information and relationships to record?; how to organise the information into coherent structures reflecting stakeholder viewpoints?; how to populate the structures with information from development and assessment activities?; and how to analyse the populated structures? (Pohl, 1996)
Conceptual data modelling techniques have been used previously to define traceability structures (including meta-models) to represent and trace between semiformal notations for IS development; first their syntactic structure is expressed in an appropriate notation or modelling language and then well-formedness constraints are added to give these constructs a semantics. Some requirements management tools (e.g., DOORS) already adopt this approach to an extent, although failure to capture both syntax and semantics undermines their potential use in developing safety or mission critical systems.

Our position is that capturing the syntax and semantics of appropriate notations as traceability structures embedded in a conceptual modelling language (along with means to verify and create linkages between these structures) provides the theoretical basis for a practical traceability environment for avionics engineering.

**Evidence** - To substantiate our argument, we do the following:-

- Develop an approach towards tackling practitioner concerns stated by our position using traceability structures. The approach known as MATrA (Meta-modelling Approach to Traceability for Avionics) provides:
  1. custom representations that define the syntax and well-formedness constraints for flexible notations (applicable to the aerospace domain);
  2. meta-models capturing the syntactic elements and well-formedness constraints of well defined notations and custom representations;
  3. a seamless environment (afforded by 1 and 2) that enables traceability between these meta-models;
  4. a mechanism for ensuring consistency across the meta-models;
  5. coverage of the traceability dimensions.

- Demonstrate the approach using material supplied by aerospace practitioners, including actual commercial specifications and industry standards.

### 1.4 Survey of Traceability Literature

This subsection provides a survey of traceability literature. Among the areas considered are key traceability concepts and factors motivating interest in the subject, both generally and throughout the aerospace industry.

#### 1.4.1 A Brief History of Traceability

Clearly, any engineering endeavour requires an orderly means of tracking between its initial set of objectives and the overall finished product. Hence traceability is nothing new! Even in software engineering, primitive traceability mechanisms were evident as far back as the early seventies (Teichroew & Sayani, 1971).
Alford (1994) observes that traceability's emergence as a topic of concern coincided with the 'birth' of systems engineering in late fifties America where it was used as a means of demonstrating compliance with requirements under Contract Law. The term itself is thought to originate from the US military who subsequently introduced it as a requirement for all defence contracts. However, it was the mid-seventies before explicit references began to appear in published literature (Alford & Burns, 1976; Belford & Taylor, 1976; Boehm, 1976; Dreyfus & Karacsony, 1976; Stallman & Sussman, 1977), so clearly the concepts significantly pre-date the actual term. Calls for improvement (Alford, 1977) were initially realised as languages with inherent support for traceability (Bell et al., 1977; Davis & Vick, 1977), followed by the first purpose built tools (Pierce, 1978; Johnson & Merrithew, 1978).

The eighties were marked by further calls for improved traceability (Distaso et al., 1980; Hoffnagle & Beregi, 1985; Roman, 1985; Tamanaha et al., 1989), as well as a succession of proprietary and research tools (cf. Bigelow, 1988; Dorfman & Flynn, 1984; Garg & Scacchi, 1989; Horowitz & Williamson, 1986; Lueders, 1984; Nejmeh et al., 1989; Pirnia & Hayek, 1981; and Sciortino & Dunning, 1984).

However, for reasons to be discussed in subsection 1.4.6, the 1990s saw an (ongoing) surge in the popularity of traceability, with further calls for improvement (Fickas & Finkelstein, 1993; IEEE, 1993b; Morris et al., 1995; Plant & Tsoumpas, 1995) and growing academic interest, e.g. the AMES, CREWS, NATURE, 2RARE and LESD projects. It is worth noting that whereas with some techniques, say formal methods, where the 'push' has generally been from academia into industry, traceability has moved in the other direction - i.e., from industry to academia.

1.4.2 Literature Definitions

Greenspan & McGowan (1978) were among the first to propose a definition of traceability. It offers a useful early perspective before the scope of definitions widened and different types of traceability became a feature (as will become evident).

"Traceability is a property of a system description technique that allows changes in one of the three system descriptions - requirements, specifications, implementation - to be traced to the corresponding portions of the other descriptions" Greenspan & McGowan (1978).

Until recently, the most commonly cited 'definition' of traceability featured in the ANSI/IEEE Guide to Software Specifications (ANSI/IEEE Std. 830, 1984):-

"A SRS (software requirements specification) is traceable if the origin of each of its requirements is clear and it facilitates the referencing of each requirement in future development and enhancement documentation" ANSI/IEEE Std. 830 (1984).

This is actually not so much a definition as a statement of the conditions necessary to establish traceability. However, it is notable for prescribing traceability of artifacts leading to production of the SRS. This contrasts with early definitions (including Greenspan & McGowan) which view the specification as a 'black-box'. Equally significant is the fact that it explicitly recommends backward traceability to previous development stages and forward traceability to all documents spawned by the SRS, thereby introducing the notion of direction which features prominently in later work.
An Introduction to Traceability

However, Davis arguably shaped current attitudes towards traceability with the following definition:-

"traceability can be defined as the ability to describe and follow the life of an artifact, in both a forward and backward direction, i.e. from its origin to development and vice versa" Davis (1990).

The influence of this definition can be clearly seen in later interpretations (notably that of Gotel, 1995). More significant is that Davis distinguishes between different types of traceability in two further subdefinitions, pre-traceability and post-traceability. These recognise the fact that producing an SRS involves tasks, problems and information which differ in content and structure from those that follow (a point affirmed by Feather, 1991 and Goguen, 1996a). This distinction is supported in studies by Ramesh & Edwards (1993) and Gotel (1995) who proposed the following definitions:-

"pre-requirements traceability refers to the ability to describe and follow those aspects of a requirement's life prior to its inclusion in the requirements specification in both a forwards and backwards direction" Gotel (1995).

"post-requirements traceability refers to the ability to describe and follow those aspects of a requirement's life that result from its inclusion in the requirements specification in both a forwards and backwards direction" Gotel (1995).

Davis (1990) also reinforced the notion of direction by specifying a need for i) backward traceability (from requirements); ii) forward traceability (from requirements); iii) backward traceability (to requirements); and iv) forward traceability (to requirements). This has greater scope than ANSI/IEEE Std. 830-1984 (which merely recommends tracing backwards and forwards from the SRS), Ince et al., 1993 (whose definitions of forward and 'reverse' traceability disregard pre-traceability) and Ramesh & Edwards, 1993 (who confine their interpretation to traceability between requirements and design).

Pearson (1996) extended the existing divisions with definitions of Pre-Formal-RS and Post-Formal-RS traceability which take into account the production of a formal requirements specification document. Pohl (1996) meanwhile further emphasises the need to maintain traceability between requirements expressed using natural language and their corresponding formal representation.

Some authors also use the terms horizontal and vertical traceability (e.g., Bersoff & Davis, 1991; Boldyreff et al., 1996; Gotel, 1995; Nejmeh, 1989; Ramesh & Edwards, 1993) in referring to associations between life-cycle objects of the same type, i.e., same development (or assessment) phase (e.g., a parent and its child requirement(s)) and different types (e.g. a requirement and its corresponding design element(s)). Although usage of these terms is fairly standard, it is worth noting that Bohner (1995) applies them visa versa, as does Lindvall (1997), whereas Choi & Scacchi (1989), Fiksel & Hayes-Roth (1993) and Lanubile & Visaggio (1995) use the designations internal and external traceability in preference to horizontal and vertical. Corriveau's (1996) interpretation of horizontal traceability is different again. Based on an incremental, object-oriented approach, he uses it to describe the tracking of artifact versions through various macro-iterations, where each iteration represents a 'complete' development life-cycle.
Consultation with practitioners revealed a further distinction in which horizontal traceability refers to artifacts related within a tool and vertical traceability refers to artifacts related between tools (though the original intent is often preserved since a separate tool is typically associated with each development stage). Similarly, Kalinsky et al. (1989) and Lindvall (1997) view traceability as relating dependent items within a model and corresponding items between models. Finally, in a document-centric approach based on syntax-trees, Han (1995, 1996 and 1997b) differentiates between inter-document relationships and intra-document relationships. Again, as he considers a separate document to be produced by each stage in development, Han also preserves the usual intent implied by horizontal and vertical traceability.


Most definitions of traceability fit into this category. However, Gotel (1995) proposes some additional groupings as part of a simple taxonomy which also describes definitions as:-

**Purpose-driven** - defined in terms of what traceability should do, for example:-

> "Traceability provides software developers with facilities to track the history of every feature of a system and the impact of changes to these features on the system design, cost and schedule" Tran et al. (1997).

Other purpose-driven definitions include those proposed by Kelley (1990), Wright (1991), Jackson (1991) and Hughes & Martin (1998). The main weakness of such definitions is their tendency to define usage rather than meaning.

**Solution-driven** - defined in terms of how traceability should be done, for example:-

> "Traceability refers to the ability of tracing from one entity to another based on given semantic relations" Ramamoorthy et al. (1986).

Other solution-driven definitions include those proposed by Ecklund jr. et al. (1996), Gardner (1994) and Roman (1985). The main weakness of these definitions is their tendency to be overly prescriptive.

**Information-driven** - emphasising the information to be traced between, for example:-

> "The characteristic of a software system that allows identification and control of relationships between requirements, software components, data and documentation at different levels in the system hierarchy" NASA-Std-2202 (1993).

Other information-driven definitions include those by Bergstein (1993), Castell et al. (1993), DO-178b (EUROCAE, 1992) and IEEE (1993a). The main weakness of such definitions is their tendency to over state (and hence restrict) what artifacts should be traced.
1.4.3 Author's Definitions

Most definitions of traceability (including those featured above) originate from software engineering literature and are couched in terms that prohibit their wider application to systems engineering. Moreover, they normally imply a single level of decomposition and single system perspective, while showing bias towards software life-cycle models. Also, the notion of traceability between artifact revisions (i.e., the maturation of an artifact) and between 'base' artifacts and their variants (modifications developed for use on different products) - we collectively term these different forms of evolution versions - is either not considered or else is captured by overloading definitions for horizontal traceability (i.e. between artifacts of the same type).

Clearly therefore significant aspects are lost when the intent of software based definitions is 'stretched to fit' a systems engineering context. We therefore provide our own taxonomy which, to keep as general as possible: i) states meaning not usage; ii) is free of bias towards particular methods of realisation; and iii) avoids excessively delimiting the artifacts to trace. The taxonomy itself comprises definitions of horizontal and vertical, revision and variant, pre and post requirements and forward and backward traceability. These are then used to provide a definition of traceability itself.

Note: vertical/horizontal traceability is adequate for 'normal' systems engineering, but for avionics (and aerospace systems generally), characteristics C2 and C3 (subsection 1.2) imply more complex products and hence more complex processes. Therefore richer definitions of traceability are required. The terms micro and macro are introduced to differentiate traceability within and across decomposition levels (e.g., system, sub-system, component). Similarly, the terms intra and inter distinguish traceability within and across system descriptions (i.e., systems that interact with one another).

- Inter-Macro-Vertical Traceability - the ability to describe and navigate relationships across system descriptions, across levels of decomposition, between development or assessment artifacts of different types.

- Inter-Macro-Horizontal Traceability - the ability to describe and navigate relationships across system descriptions, across levels of decomposition, between development or assessment artifacts of the same type.

- Inter-Micro-Vertical Traceability - the ability to describe and navigate relationships across system descriptions, within a decomposition level, between development or assessment artifacts of different types.

- Inter-Micro-Horizontal Traceability - the ability to describe and navigate relationships across system descriptions, within a decomposition level, between development or assessment artifacts of the same type.

- Intra-Macro-Vertical Traceability - the ability to describe and navigate relationships within a system description, across levels of decomposition, between development or assessment artifacts of

---

* Following Gotel's classification, these properties imply a direction driven approach.
different types.

- **Intra-Macro-Horizontal Traceability** - the ability to describe and navigate relationships within a system description, across levels of decomposition, between development or assessment artifacts of the same type.

- **Intra-Micro-Vertical Traceability** - the ability to describe and navigate relationships within a system description, within a decomposition level, between development or assessment artifacts of different types.

- **Intra-Micro-Horizontal Traceability** - the ability to describe and navigate relationships within a system description, within a decomposition level, between development or assessment artifacts of the same type.

These definitions are shown graphically in figure 1.1 and summarised in table 1.1.

![Inter/Intra, Macro/Micro Horizontal/Vertical Traceability Types](image)

**Table 1.1 - 'Summary of Inter/Intra, Macro/Micro Vertical/Horizontal Traceability Types'**

<table>
<thead>
<tr>
<th>Traceability Type</th>
<th>Within System</th>
<th>Within Decomposition</th>
<th>Same Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inter Macro Vertical</td>
<td>✗</td>
<td>✗</td>
<td>❔</td>
</tr>
<tr>
<td>Inter Macro Horizontal</td>
<td>✗</td>
<td>✗</td>
<td>❔</td>
</tr>
<tr>
<td>Inter Micro Vertical</td>
<td>✗</td>
<td>✓</td>
<td>❔</td>
</tr>
<tr>
<td>Inter Micro Horizontal</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intra Macro Vertical</td>
<td>✓</td>
<td>✗</td>
<td>❔</td>
</tr>
<tr>
<td>Intra Macro Horizontal</td>
<td>✓</td>
<td>✗</td>
<td>❔</td>
</tr>
<tr>
<td>Intra Micro Vertical</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intra Micro Horizontal</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
An Introduction to Traceability

- **Revision Traceability** - the ability to describe and navigate relationships between instances of the same artifact at different stages of maturity.9

- **Variant Traceability** - the ability to describe and navigate relationships across different projects (and across product families within the same project), between base artifacts and their derivatives.

- **Pre-Requirements Traceability** - the ability to describe and navigate relationships between a requirement and its originating artifacts.

- **Post-Requirements Traceability** - the ability to describe and navigate relationships between a requirement and any artifacts relating to its development, assessment or evolution.

- **Forward Traceability** - the ability to describe and navigate relationships between i) artifacts originating a requirement and (verification of) its realisation; ii) revision\textsubscript{1} and revision\textsubscript{a} of an artifact; and iii) base artifacts and their variants.

- **Backward Traceability** - the ability to describe and navigate relationships between i) (verification of) some hardware or software feature and the artifacts originating its requirement(s); ii) revision\textsubscript{a} and revision\textsubscript{1} of an artifact and iii) a variant artifact and its base.

- **Traceability** - the ability to describe and navigate relationships, forwards and backwards, within and across system descriptions, within and across decomposition levels, between artifacts of the same or different types, their revisions and variants, pre and post requirements specification.

1.4.4 On Formal Definitions of Traceability

At present, no formal definition of traceability exists. However, we can formally conceptualise what is meant by claims that a project is traceable (where a project refers to a completed product and all its documentation) based on the above definitions.

For the following types of traceability, inter (ir), intra (ia), macro (ma), micro (mi), horizontal (ht), vertical (vt), revision (re), variant (va), forward (fw), backward (bw), pre-RS (pr) and post-RS (ps), we introduce the predicates, ir\textsubscript{ma_vt} ir\textsubscript{ma_ht}, ir\textsubscript{mi_vt}, ir\textsubscript{mi_ht}, ia\textsubscript{ma_vt}, ia\textsubscript{ma_ht}, ia\textsubscript{mi_vt}, ia\textsubscript{mi_ht}, re, va, pr, ps, fw and bw. These are of the form type: \( P \rightarrow \mathbb{I}B \), where \( P \) represents the hypothetical set of all projects. Each predicate holds for a project \( P \), iff the project complies with the predicated type of traceability. We also introduce a predicate \( T \), which holds when the project complies with our definition of traceability. We now assert the following relations:

\[
T(P) = \text{ir}_{\text{ma_vt}}(P) \land \text{ir}_{\text{ma_ht}}(P) \land \text{ir}_{\text{mi_vt}}(P) \land \text{ir}_{\text{mi_ht}}(P) \land \\
\text{ia}_{\text{ma_vt}}(P) \land \text{ia}_{\text{ma_ht}}(P) \land \text{ia}_{\text{mi_vt}}(P) \land \text{ia}_{\text{mi_ht}}(P) \land \\
\text{re}(P) \land \text{va}(P)
\]

\[
T(P) = \text{pr}(P) \land \text{ps}(P)
\]

\[
T(P) = \text{fw}(P) \land \text{bw}(P)
\]

9 Note the intersection of ideas with Configuration Management literature. Ramesh & Jarke (1999) suggest the main difference is one of granularity such that CM mainly concerns coarse-grained relationships, whereas traceability can be finer-grained.
1.4.5 Dimensions For Traceability

Figure 1.2 illustrates the concepts of pre-requirements and post-requirements traceability and horizontal, vertical and revision traceability\(^\text{10}\). Henceforth, the latter three are referred to as traceability dimensions; we therefore speak of the horizontal, vertical and revision dimensions. To aid readability, traceability is considered between just four basic artifact types (Requirement) Source, Requirement, Design and Implementation.

![Figure 1.2 - 'Horizontal, Vertical and Revision Traceability Dimensions'](#)

Figure 1.3 illustrates (one aspect of) variant traceability (i.e. the variant dimension), namely traceability across projects, using a subset of the Airbus 'family' of aircraft. It depicts hypothetical relationships between requirements and design artifacts for the A300 and A320, which together derive inputs for the A340. Note, where a project is composed of a 'family' of (e.g., A319, A320 and A321) rather than individual aircraft, then variant associations can exist within as well as between cubes.

![Figure 1.3 - 'Variant Traceability'](#)

---

\(^{10}\) In order to simplify figures 1.2 and 1.3, we only illustrate the intra-micro forms of horizontal and vertical traceability.
An Introduction to Traceability

1.4.6 Drivers For Traceability
The introduction referred to a growing interest in traceability, with several authors (including Gotel, 1995; Kenny, 1996; Palmer, 1997; Ramesh et al., 1995; Watkins & Neal, 1994; White, 1994a; and Wieringa, 1995) highlighting some general motivations for its increased use.

That said, traceability remains something of a paradox. Many still regard it as time-consuming, tedious and labour-intensive (as reported in Kotonya & Sommerville, 1998; Ramesh, 1993; Cockram et al., 1998; and White, 1994a), with perceived 'benefits' such as reduction in rework (Palmer, 1997) and the ability to manage costs (Dönges & Pohl, 1998; Watkins & Neal, 1994), planning and scheduling (Ramesh & Edwards, 1993; Booth, 1993; Lindvall, 1997), performance (Hodge, 1994) and risk (Cross, 1996; Wilson et al., 1997b) often seen as speculative. This is because while direct traceability costs are relatively easy to calculate, the return on investment is far more difficult to measure. And yet, we estimate around ninety percent of published work on traceability has emerged during the last ten years.

The literature suggests two 'key drivers' have contributed to this growing interest. These are as follows:-

- The Emergence of Requirements Engineering
- Increased use of Quality Management & Compliance Frameworks

1.4.6.1 The Emergence of Requirements Engineering (RE)
The contribution of sub-standard requirements to systems that are delivered late, over budget and which don't fulfil the needs of users is well-known (Boehm, 1981). Traditionally, the requirements phase of a project was seen as little more than a front-end to development and as a result was not accorded the same degree of precision as down-stream activities.

Requirements Engineering (RE) is a relatively recent term encapsulating all of the activities involved in eliciting, understanding, analysing, documenting and managing requirements. The term engineering is intended to convey the impression that this is accomplished through a practical, systematic and repeatable process, even in areas with more philosophical and social underpinnings (e.g., ethnography; Sommerville et al., 1993).

Though the idea of applying an engineering orientation to systems analysis dates back to the mid-seventies (IEEE, 1977), the last few years have seen an escalation of interest. The literature provides a useful barometer of this growth. Before 1990 material was both sparse and disjoint, whilst nowadays, in addition to being the subject of conferences, symposia and books (Kotonya & Sommerville, 1998; Pohl, 1996; Sommerville & Sawyer, 1997; Wieringa, 1996), RE now has its own journal, newsletter and BCS specialist group.

Moves to improve Requirements Engineering within the aerospace sector are evident from its

11 Return on traceability investment has been estimated at four per cent of total expenditure on US DoD projects Ramesh (1994).
collaboration with academia. In Europe alone, recent or ongoing involvement includes such projects as 2RARE (Alcatel Espace), CREWS (GEC-Marconi), DCSC (BAE SYSTEMS), ISRE (GEC-Marconi), KARE (BAE SYSTEMS), REALMS (Aerospatiale) and STEFFIE (Lucas Aerospace and GEC-Marconi). This is additional to a number of in-house initiatives also taking place (e.g., the Airbus Industrie CARE project; Airbus, 2001).

The emergence of RE and efforts to improve its attendant areas for eliciting, understanding, analysing, and documenting software requirements has had an indirect bearing on traceability. For example, the transition from analysis to design in object-oriented approaches is claimed to naturally enhance traceability by removing the 'air gaps' in disparate notations across the two sets of artifacts (Barbier, 1994; Jacobson et al., 1993). Though beyond the scope of this thesis, a growing corpus of work reflects the issues (and problems) relevant to traceability for object-oriented development (cf. Börstler, 1996; Bosch, 1998; Buhr, 1995; Corriere & Hayashi, 1994; Diagne & Kordon, 1996; Ecklund et al., 1996; Galle, 1996; Gossain, 1995; Ihme et al., 1995; Lindvall, 1997; Premerlani, 1994; Scalzo & Hugue, 1996; Wieringa, 1998; Wood, 1995).

However, the aspect of RE that has influenced growth in traceability more than any other is requirements management. It is widely acknowledged that requirements change is a major source of risk in terms of cost, schedule and quality (Strens, 1995), while with safety-critical systems, failure to properly manage its effects can pose a threat to human life (de Lemos et al., 1995). To minimise these difficulties, such changes must be managed effectively and for that, traceability is required.

The first thing to say about requirements change is that it is unavoidable and while especially prevalent in the early phases of a project, normally occurs throughout. Indeed as Lehman & Belady (1985) maintain, systems must continually respond to their environment or become progressively less effective. Practitioners consulted during this study estimate that following the initial volatility, requirements typically change on average around three per-cent per month during the lifetime of a project; this is corroborated by published figures in Gries (1997). It is worth noting, as Chudge & Fulton (1994) and Harker & Eason (1993) point out, that early life-cycle based approaches had naively assumed a complete and static set of requirements were attainable prior to design. However, several alternative models have since emerged which accommodate change (Gladden, 1982; Lee & Yen; 1993; Rolland, 1994a), some with explicit traceability mechanisms (Brouse [1992], Martin et al. [1993], Tiel [1993] and Hugge & Lang [1995]).

The other key point to make is that requirements change need not necessarily reflect poor practice, such as basing requirements on erroneous assumptions (Ramesh & Jarke, 1999), failure to resolve conflicting viewpoints (Easterbrook et al., 1994; Hughes et al., 1995) or failure to identify missing information (Takahashi & Yamamoto, 1995). It can actually result from a combination of other factors, many of which are beyond a requirements engineer's control. For instance, customer's evolving knowledge of the target system is often a major source of instability whereby expectations grow (and their
An Introduction to Traceability

requirements change) as the product emerges and they see new possibilities (Pohl & Jacobs, 1994); this is especially true for interface requirements (Lubars et al., 1993). External or environmental factors can force further unforeseen change on both customer and developer, especially for systems with protracted life-cycles. Finally, technical problems encountered when implementing a requirement may also lead to change. This may be due for example, to timing issues discovered during systems integration, or the emergence of additional safety evidence (Kelly & McDermid, 1999). Interested readers are referred to Kotonya & Sommerville (1998) for other factors causing requirements change.

The main problem in change management lies in tracking the so-called ripple-effect (Collofello & Vennergrund, 1987; Yau & Tsai, 1987) where changes to one artifact can have an unforeseen impact elsewhere in the system. As Bohner (1995) observes, ripple-effects can be either direct (where connectivity between affected artifacts is immediate) or residual (where connectivity is transitive and the impact more difficult to detect). To a certain extent developers can minimise the problem by using modular development techniques. With software this often implies an object-oriented approach based on encapsulation, high cohesion and low coupling, etc. (Corriveau, 1996; Hoffman, 1990; Sugden & Strens, 1996). However, whilst such methods can localise the impact of change, developers still require means of making the ripple-effect more visible in order to address potential 'side-effects' of proposed changes (Freedman & Weinberg, 1981).

Horizontal and vertical traceability provide such means and are therefore pre-requisites for effective impact assessment (Escudie et al., 1994; Sugden & Strens 1996; Tryggeseth & Nytro, 1997). From a (intra-micro) horizontal standpoint, suppose ultimate fuel pressure (UP) is defined as a function of a separate requirement for normal fuel pressure (NP); if NP were to change, then so too must the requirement for UP. Similarly, from a (inter-macro) horizontal perspective, changes to the specification of sensors calculating measured aircraft state may propagate to navigation system requirements and thence requirements for the flight control system; clearly intra/inter and micro/macro vertical traceability will also be needed to track any design changes arising. In addition, revision traceability is also useful for change management as it allows practitioners to re-construct the evolutionary history of an artifact which in turn, permits the point where errors were introduced to be identified, as well as providing useful volatility data for future projects. Finally, variant traceability is necessary for tracking the effects of change propagation between products. It is especially useful for changes instigated by requirements errors where one replicated error can potentially 'contaminate' an entire product family.

Besides dependencies among artifacts, the rationale underlying development decisions (i.e., decision rationale) provides important supplementary trace information to aid change management (Sugden & Strens, 1996)\(^{12}\). Most approaches to representing decision rationale are based on the argumentation structuring principles pioneered by Toulmin (1958); their underlying models are typically constructed from nodes (such as issue, alternative and claim) which are linked to form networked structures by relationships (such as achieves, denies and pre-supposes). Prominent examples of the form include

---

IBIS (Kunz & Rittel, 1970) and gIBIS (Conklin & Yakemovic, 1991), DRL (Lee, 1991) and QOC (MacLean et al. (1991). However, these early models have been widely criticised for representing decisions out of context. Therefore, more recent examples attempt to link the rationale to a model of development artifacts; cf. Monk et al. (1995), Potts (1994), Han (1997a) and Ramesh & Dhar (1992), as well as the DRCS proposed by Klein (1993a) which we consider further in relation to this work in Chapter Three.

It is important to remember that traceability is not about preventing change (Card, 1988). Rather it is a communication and control mechanism to be used, typically as part of a change control process, in managing its realisation (cf. Chudge & Fulton, 1994; Coyne, 1993; Kotton & Sommerville, 1998; and Gries, 1997). With the trend towards larger and more complex systems, change control and impact analysis have become even more significant. Consequently, they rate among the more mature traceability sub-topics (cf. Bohner, 1995; Canfora et al., 1995; Cimitile et al., 1992; Fyson & Boldyreff, 1998; Han, 1996; Kelly & McDermid, 1999; Lindvall, 1997; Liu et al., 1993; Madhavji, 1992; Ramamoorthy et al., 1990; Westfechtel, 1989; Whitgift, 1991 and Yau et al., 1988).

Historically, the traceability information most commonly maintained for requirements management purposes is requirements-requirements and requirements-design traceability. However, work within the RE community has succeeded in evolving the practice of tracing requirements back to their source (i.e., pre-requirements traceability) and hence potential change provocateurs. Examples include Brouse (1992), Curran et al. (1994), Gotel (1995), Johnson et al. (1991, 1992), Laubengayer & Spearman (1994), Leite et al. (1997), Moores & Champion, 1994, Morris et al. (1994), Pohl (1996), Ramesh (1994) and Sawyer et al. (1996).

It can be seen therefore that the emergence of Requirements Engineering has provided a focus for work on traceability and a forum for the exchange of views. Improvements in traceability-practice have come both indirectly as a by-product of methods for the elicitation, understanding, analysis and documentation of requirements, and directly through better techniques for their management.

1.4.6.2 Increased use of Quality Management & Compliance Frameworks

Evaluative frameworks are increasingly used as arbiters of acceptability with respect to quality, safety and other hallmark attributes. Where quality and safety often differ is that with the former, framework compliance may be desirable for commercial reasons (i.e., a recognised stamp of approval can help gild the corporate image), whereas with the latter, it is usually necessary to achieve certification.

The growing popularity of evaluative frameworks is largely attributable to two factors. First, the prevailing 'quality culture' that places great emphasis on achieving quality within all fields of endeavour; as Storey (1996) observes, it often seems our goals in life may be encompassed by driving a quality car, attaining a quality home and spending quality time with our family! And second, increased
use of safety-critical systems; growth in using computers instead of electromechanical or other components to control safety-critical applications is largely due to their processing power\(^{13}\), physical size and weight, and flexibility\(^{14}\) - all of which can lead to cost savings across a project (Storey, 1996; Leveson, 1995).

According to Sheard (1997), the two main forms of evaluative framework are process improvement models and standards and guidelines. We now briefly consider examples of each from a traceability standpoint, before considering one particular set of guidelines for the aerospace domain (subsection 1.4.6.2.1) which we refer to regularly throughout this thesis.

**Process Improvement Models (PIM)** are founded on established links between the quality of a product and the quality of process used to create it. Though PIMs have no formal ratification (unlike standards which are subject to industry approval), they provide a way in which to assess the capabilities of an organisation based on its key processes. Perhaps the most widely known PIM is the Software Engineering Institute’s (SEI) Capability Maturity Model (CMM). Originally proposed by Humphrey (1988), but later revised by Paulk *et al.* (1993), it was devised to help the US Department of Defense assess the capabilities of software contractors. CMM provides a five-layer stratum which classifies software processes as: i) *initial*, ii) *repeatable*, iii) *defined*, iv) *managed* and v) *optimising*; traceability is necessary to achieve level two status (and above). The model itself provides only general guidance, although the need to support a minimum of post-requirements traceability may be assumed.

The systems engineering Capability Maturity Model (SE-CMM) and the systems engineering Capability Assessment Model (SECAM) are further examples of PIMs. SE-CMM (SEI, 1995) was again developed by the Software Engineering Institute and so has the advantage of association with the organisation that devised the original (software) CMM. As a minimum, traceability is necessary to support ‘Process Area’ Two of the model. Similarly, SECAM (INCOSE, 1996) - the product of an INCOSE working group - divides process capability into nineteen ‘Key Focus Areas’, with traceability influential in those on ‘Tracking’ (1.2) and ‘Configuration Management’ (1.5). The main difference between SE-CMM and SECAM is that the former includes non process characteristics.

**Standards and guidelines** meanwhile establish contractual and regulatory requirements for development and assessment. Notable examples include the ISO 9000 series which can be used to develop quality management systems across a range of organisations, from manufacturing to service based industries (Rothery, 1993). The most general of these standards is ISO 9001 which applies to systems and software inasmuch as it concerns the quality process of any organisations that design, develop and maintain products. A supporting document, ISO 9003 (implemented in the UK through TickIT), provides a further interpretation of ISO 9000 appropriate to the software industry. Traceability is necessary for accreditation to the ISO 9000 series; in particular, it is an explicit requirement for

---

\(^{13}\) This enables complex control functions, as well as sophisticated safety mechanisms such as self diagnostics and interlocks.  
\(^{14}\) By flexibility, we mean system characteristics can be changed through software upgrades, without need for hardware alterations.
realising clause 4.8 of ISO 9001, whilst Ince (1994) further notes the implicit relevance of clauses 4.3, 4.4 and 4.16, as well as 5.3 and 6.1 from ISO 9003. Moreover, the generic language used to express these standards makes them applicable across the range of traceability dimensions.

Traceability is also evident in the requirements management sections of UK Def-Std. 00-55 (MoD, 1997), for example 31.2.3 to 31.2.6 (inclusive), as well as US Mil-Std-498 (DoD, 1994), notably 4.2.6, 5.4.2 and 5.9.3. ARP 4754 (EUROCAE, 1996a) meanwhile provides guidance on traceability for the development of highly integrated or complex aircraft systems, while DO 178b does likewise for the production of software for airborne systems and equipment; respectively, parts 7.3 and 5.12b (pre-RS), 8.43 and 6.2c (post-RS), 7.6.1 and 5.5a (horizontal), 5.2.1 and 5.5c (vertical), 9 and 7 (revision), and 11.3.3 and 12.1.5 (variant) demonstrate that traceability across all four dimensions is recommended by both documents. ARP 4754 is discussed further in the following subsection.

Clearly, the above-mentioned frameworks are proving to be a key driver for growth in traceability. From a quality perspective, organisations are increasingly obliged to tackle the issue in establishing processes that conform to the requirements of their industry. For instance, ISO 9000 accreditation is already mandatory for contractors to many European governments, while the US government normally favours a minimum of CMM level 2 or level 3 status when awarding defence contracts. From a safety perspective, standards such as DO 178b have been even more influential as conformance is normally a requirement for legal reasons. Note that ARP 4754 is defined in the context of Joint Aviation Regulations and Federal Aviation Regulations (JARs & FARs) which effectively makes it a de facto standard.

Accordingly, tools and approaches are now emerging which help manage and/or demonstrate framework and more especially, standards compliance (cf. Dawkins, 1998; Emmerich et al., 1999; and Wilson et al., 1997a). Other works with potential application to framework compliance include those addressing traceability of safety properties. In particular, the Safety Argument Manager (SAM) which supports construction and analysis of safety arguments, as well as managing the inter-relationships between assessment techniques. SAM provides traceability at two levels: i) an underlying data model underpinning the analysis techniques (Wilson & McDermid, 1995); and ii) a goal structuring notation (GSN) for tracing safety arguments (Wilson et al., 1995; Wilson et al., 1996; Kelly & McDermid, 1997). Further examples include Jenkins et al. (1997), Mason & Saeed (1998), Pearson et al. (1998), Leveson & Reese (1998) and with respect to tracing safety properties of COTS (commercial-of-the-shelf) components, Dawkins & Riddle (2000).

In the following subsections we introduce ARP 4754, together with accompanying guidelines for its realisation, ARP 4761, and provide a brief overview of the traceability requirements they ‘impose’. This will serve as an aid to reader orientation for work in Chapters Five and Six.

---

15 It should be noted that despite their common purpose (see Coallier, 1994 or Paulk, 1995 for a comparison), ISO 9000 and CMM are very different in approach as efforts to integrate them demonstrate (Rozman et al., 1997).
16 GSN is conceptually similar to a number of other goal oriented methods (cf. Landes & Studer, 1995; Alvarez & Castell, 1996; Dardenne et al., 1993; van Lamsweerde et al., 1995; Yu, 1993 and Mylopoulos et al., 1992)
1.4.6.2.1 Aerospace Recommended Practice (ARP) 4754 & 4761

ARP 4754 and ARP 4761 (EUROCAE, 1996b) describe safety assessment guidelines and methods for the certification of civil aircraft\(^\text{17}\). Assessment runs parallel to development and comprises four primary sub-processes (see figure 1.4): Functional Hazard Assessment (FHA), Preliminary System Safety Assessment (PSSA), Common Cause Analysis (CCA) and System Safety Assessment (SSA); we briefly discuss these below. A number of interdependent analysis techniques are used to support assessment, each providing a different insight into failure behaviour of the target system. Those used with ARP 4754/4761 are Fault-Tree Analysis - FTA (Vesely \textit{et al.}, 1981) and Failure Mode and Effects Analysis - FMEA (IEC, 1985).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Figure1.4.png}
\caption{Overview of ARP 4754/4761 Safety Assessment Process}
\end{figure}

From ARP 4761, the primary assessment sub-process shown in figure 1.4 are summarised as follows:-

\textit{Functional Hazard Assessment} is conducted at the beginning of the aircraft/system development lifecycle. It aims to identify and classify the failure conditions\(^\text{18}\) associated with aircraft functions and combinations of aircraft functions and to establish the rationale for such classifications. Once aircraft functions have been allocated to systems, each system which integrates multiple aircraft functions is re-examined. The FHA is then updated to consider failure of single or combinations of aircraft functions allocated to a system. The output of FHA provides a starting point for conducting the PSSA.

\textit{Preliminary System Safety Assessment} is a systematic examination of proposed system architectures

\begin{itemize}
\item \textit{Functional Hazard Assessment} deals with the assessment process at a 'conceptual' level, whereas the latter considers application of specific assessment techniques in the context of that process.
\item Classifications used with ARP 4754 and 4761 are catastrophic, hazardous, major, minor and no safety effect.
\end{itemize}
An Introduction to Traceability

with the aim to identify failures that can lead to functional hazards identified by the FHA. The objective of PSSA is to establish safety requirements and to determine whether the proposed architecture can reasonably be expected to satisfy safety objectives identified by the FHA. PSSA is also an interactive process associated with design definition and is conducted at the system and item stages of development. At the lowest level, PSSA determines the safety related design requirements of hardware and software. The PSSA will usually take the form of a Fault Tree Analysis, and should also include Common Cause Analyses.

Common Cause Analysis supports development of system architectures by evaluating their sensitivity to common cause events. It is therefore largely conducted as part of PSSA, but may also form part of the SSA and to a lesser extent, FHA. Common Cause Analyses can comprise three sub-processes:

1. Particular Risk Analysis (PRA) considers threats from outside the system(s) and item(s) concerned; e.g. bird-strike, fire, leaking fluids, etc.;
2. Zonal Safety Analysis (ZSA) considers each aircraft zone to establish whether applicable safety requirements have been met; and
3. Common Mode Analysis (CMA) provides evidence that failures assumed to be independent, are independent.

System Safety Assessment involves detailed examination of the implemented system to show compliance with all relevant safety requirements. The process is similar to PSSA, except that instead of evaluating proposed architectures and deriving safety requirements, SSA aims to demonstrate that all requirements established by the FHA and PSSA have been satisfied. For each PSSA carried out at a different level, there should be a corresponding SSA (the highest level being the system SSA). The SSA is usually based on the PSSA FTA and uses quantitative values obtained through Failure Modes and Effects Analysis. Note, the case study in subsection 6.3 of this thesis is based on fragments from Appendix L of ARP 4761 that focus on the System Safety Assessment.

1.4.6.2.2 On Traceability Requirements for ARP 4754 & 4761

Traceability among artifacts produced by the above sub-processes is necessary to comply with various validation checks described in ARP 4754 (including those in the sections previously mentioned in 1.4.6.2). The checks themselves are phrased in broad terms and should be tailored to specific projects. For instance subsection 7.3 states the following check list item among an example set of questions for assessing requirements completeness at each hierarchical level:

- Do requirements trace to identified sources?
  - functions, hazards and failure condition classifications identified in the FHA.

Figure 1.5 illustrates one interpretation of this check. It depicts traceability to a fragment of the aircraft FHA - featuring a single function, hazard and failure classification - from the safety requirement excluding this particular condition. Fault Tree Analysis is then used to derive lower level requirements from those identified by the aircraft FHAs; figure 1.5 summarises the overall relationship between FHA

---

19 An 'item' is defined as one or more hardware and/or software components treated as a unit (ARP 4761).
20 ARP 4754 and 4761 exclude detailed coverage of software and hardware issues which are considered respectively by DO-178B and the working document "Design Assurance Guidance for Airborne Electronic Hardware".
An Introduction to Traceability

Aircraft FHA

<table>
<thead>
<tr>
<th>Function</th>
<th>Phase</th>
<th>Failure Condition</th>
<th>Effect</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deceleration on ground</td>
<td>Landing</td>
<td>Loss of deceleration capability on ground</td>
<td>Crew unable to stop aircraft on runway</td>
<td>Catastrophic</td>
</tr>
</tbody>
</table>

Aircraft Requirements

- Loss of deceleration capability on ground shall be less than $5.00E-09$ per flight.

Aircraft FTA

- Loss of Thrust Reversers
  - Loss of Effective Wheel Braking

System FHA

<table>
<thead>
<tr>
<th>Function</th>
<th>Phase</th>
<th>Failure Condition</th>
<th>Effect</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wheel braking</td>
<td>Landing</td>
<td>Loss of all wheel braking</td>
<td>Crew ability to stop aircraft on runway significantly reduced</td>
<td>Hazardous</td>
</tr>
</tbody>
</table>

Figure 1.5 - 'Example Relationship Between Aircraft and System FHA and Aircraft FTA'

Figure 1.6 - 'Steps in ARP 4754/4761 Safety Assessment Process'
and FTA whereby aircraft FHAs generate top level events in the aircraft FTAs; likewise, catastrophic or hazardous basic events from the latter provide inputs to system FHAs. As section 3.2 of ARP 4761 indicates, these paths should also be traceable.

Figure 1.6 shows the global flow of information among the FHA, PSSA CCA, and SSA sub-processes. However, it also provides a general indication of traceability requirements. Recall from subsection 1.2 (C2) that we may regard the activity that transforms an input to an output as an abstraction of a traceability association between the two. For instance, architectural requirements form inputs to the preliminary FTA/CCA during item requirements identification which in turn, derives hardware level requirements (figure 1.7A). A derives traceability association between these artifact types may therefore be considered an abstraction of this process (figure 1.7B).

![Figure 1.7 - 'Information Flow and Traceability (context ARP 4761)'](image)

In the absence of clear guidance from the standards themselves, such an approach could be used (in conjunction with figure 1.6) as a basis to determine traceability needs for the ARP 4754/4761 process.

### 1.5 Chapter Summary

This chapter has provided a context for the thesis by considering from a user viewpoint our domain and focus of interest, namely traceability for the aerospace industry. A number of characteristics were introduced to highlight traceability issues relevant to this domain; perhaps only the nuclear and rail industries offer similar challenges in the provision of effective support for traceability.

The literature distinguishes different types of traceability according to factors such as scope (e.g., pre-requirements, post-requirements) and direction (e.g., horizontal and vertical, forward and backward). However most existing work (including attempts to define the term) concentrates on traceability for software (rather than system) development and as such, fails to address issues pertinent to the industry characteristics referred to above. We therefore proposed our own definitions, in particular, refining the notion of horizontal and vertical traceability to accommodate aspects of these characteristics.

---

21 Not every step will be needed for assessment, but each must be considered for applicability.
A further visual conceptualisation was proposed by representing horizontal, vertical and revision traceability as dimensions of a cube, with variant traceability (i.e., the variant dimension) similarly depicted as traceability between cubes.

The dimensions concept can be viewed as a ‘meta framework’ relating abstractions from the different stages of development or assessment (e.g., requirements and design), whereas the Workspace concept (alluded to in subsection 1.3) can be viewed as (part of) a ‘concrete framework’ relating selected notations used in the different stages by aerospace practitioners. The Workspace is one actualisation of the dimensions with the artifact nodes being realised as meta-models representing selected notations, and the traceability dimensions realised through associations between elements of the meta-models. The relationship between these two frameworks is illustrated in figure 1.8; for the concrete framework, the meta-models are depicted by actual notations rather than their corresponding meta-models.

Finally, this chapter asserted that recent growth in both interest and application of traceability has been driven by two factors. Firstly, the emergence of requirements engineering which has provided a direct focus for traceability research, while ensuring it is not overlooked in the development of new notations and techniques. And secondly, growth in use of compliance frameworks as arbiters of quality (e.g., ISO 9001), safety (e.g., ARP 4754/4761) and other non-functional attributes.

1.6 Thesis Structure
The remainder of this thesis is structured as follows:
Chapter Two provides an overview of current traceability techniques, together with a basis for their classification. For reader orientation, we demonstrate means used in this thesis to represent and implement the traceability structures introduced in our thesis argument. Chapter Two also considers means of realising traceability using a variety of proprietary and commercial tools.

Chapter Three provides an introduction to the Meta-modelling Approach to Traceability for Avionics (MATrA) proposed by this thesis. The main works that have influenced its development are considered (foundations), together with the nature of their influence and/or perceived weaknesses. The chapter goes on to introduce the key principles of MATrA (fundamentals), their purpose and composition.

Chapter Four presents a number of novel meta-models (traceability structures) capturing development notations used by avionics engineers in a format amenable to traceability. Specifically: i) a Natural Language structure; ii) a User Centred Requirements Structure (featuring Use Case Models, Scenarios and Message Sequence Charts); iii) a structure for the representation of Real-Time Networks; and iv) a structure for the representation of SPARK Ada program code.

Chapter Five presents novel meta-models capturing safety assessment and product management notations used by avionics engineers, again in a format amenable to traceability. Specifically: i) a Fault Tree Analysis structure; ii) a Failure Modes & Effects Analysis structure; and iii) a Programme Evaluation & Review Technique structure. In addition, we extend MATrA to include support for traceability of artifacts across the revision and variant dimensions.

Chapter Six features two case studies that illustrate a subset of the structures from Chapters Four and Five. The first demonstrates aspects of the User Centred Requirements Structure (from Chapter Four) using an actual commercial specification supplied by BAE SYSTEMS. The second demonstrates the Fault Tree and Failure Modes and Effects Analysis structures (from Chapter Five) using extracts from a contiguous example featured in aerospace industry guidelines (ARP 4761).

Chapter Seven presents conclusions drawn from the thesis and the extent to which work in the previous chapters supports our thesis argument. A number of areas for possible future work are also highlighted.

Appendix A provides some additional constraints and rules over the User Centred Requirements Structure from Chapter Four.

Appendix B presents material supplementary to the work on revisions and variants in Chapter Five.

Appendix C and Appendix D include further data for the two case studies featured in Chapter Six.

Appendix E contains material relevant to proposals for further work discussed in Chapter Seven.
Chapter 2  Techniques and Tools for Traceability

2.1 Introduction
At a reductionist level, traceability is simply a means of managing relations (or functions) on sets of artifacts; in other words, set theory provides the mathematical foundation for all traceability techniques. Therefore the level of sophistication afforded by a particular technique depends on what additional concepts it adds to basic set theoretic constructs. Working from such a premise, this chapter provides an overview of current traceability techniques, before considering the support provided for them by various proprietary and commercial tools. The aim of the survey is to establish a firm technical basis for MATrA.

2.2 Traceability Techniques
Traceability techniques broadly divide into two categories, namely cross-referencing and conceptual data modelling (or simply data modelling). Cross-referencing can be further partitioned into higher and lower-order techniques, both of which are founded on the set-oriented principles of graphs and matrices; the main difference being that higher-order approaches build on these underlying mathematical principles by introducing features which make them more amenable to practical application. In contrast, conceptual data modelling has its origins in software engineering which has spawned a number of rich semantic notations suited to the development of more sophisticated traceability techniques. We note that these notations can employ either a graphical or lexical representation, although as we shall demonstrate, some include both.

The basis of our classification considers three aspects:- i) structure (in particular, support for the typing of data elements and relationships); ii) constraints (the ability to specify restrictions on the way in which data elements are related); and iii) operators (means of manipulating data elements).

2.2.1 Cross-Referencing
This subsection introduces the cross-referencing technique and the support it provides for traceability.

2.2.1.1 Foundations
In describing the foundations of cross-referencing techniques, we distinguish higher-order cross-referencing from basic lower-order approaches.

2.2.1.1 Lower-Order Cross-Referencing
The simplest of all traceability techniques comprise a single set of ‘traces-to’ associations between a single set of ‘artifact’ types. This assertion may seem overly reductionist given the range of entities used in some data-modelling approaches (cf. Herzog & Törne, 1999; Oliver, 1994; Pyle et al., 1993). However, if we regard the pairing as traceability super-classes of which all other types are simply specialisations (a view supported by Ramesh & Jarke, 1999 and Riddle & Saeed, 2000), then the
assertion becomes more credible. It is simply that the restriction of types within lower-order graph and matrix based schemes forces the extremes of abstraction.

Graph theory and directed graphs (digraphs) in particular are an intuitive and popular means of representing traceability relationships (cf. Attipoe, 1996; Bohner, 1995; Cimitile et al., 1992; Fyson & Boldyreff, 1998; Lanubile & Visaggio, 1995; Luqi, 1990; Yau et al., 1988). Formally, a digraph is an ordered triple \((N, A, g)\) where \(N\) is a set of vertices or nodes, \(A\) is a set of edges or arcs and \(g\) is a bijective function associating with each arc \(a\), an ordered pair \((x, y)\) of nodes; i.e., the function is both surjective as all arcs in \(A\) are assigned, and injective because arcs have only one \((x, y)\) pair.

As an example, consider the following function defined as: \(\text{traces-to} : (N \times N) \rightarrow A\) where the set of 2-tuples in the domain \(\{<x, y>\}\) correspond to section numbers in a document text. As such, the nodes are carriers of information and therefore any graph of the function is said to be labelled. One possible graph is shown in figure 2.1 (centre). It should be stressed that such functions are likely to be partial since we would not expect an association between every section pair. A case in point is that of instances of 2-tuples where \(x = y\), a feature which manifests as loops in the resultant graph and which makes little sense in a cross-reference context.

Further utility can be gained by making arcs (as well as nodes) carriers of information, specifically some numerical value or weight describing the 'strength' of association. We can therefore extend the digraph definition to accommodate these weightings which now reads \((N, A, g, W, w)\), where \(W\) is the set of possible degrees of strength (e.g., \{3, 6, 9\} indicating low, medium and high) and \(w\) is a function from \(A\) into \(W\) (defined as \(w : A \rightarrow W\)) that associates a weighting with each arc \(a\); \(w\) is neither surjective nor injective since a set of arcs may be assigned to a subset of \(W\), whilst each weighting \(w\) may be assigned to \(n > 1\) arcs in \(A\).

We note that pure directed graphs of the form shown in figure 2.1 are generally limited to support for horizontal traceability (i.e., among artifacts of the same type) by virtue of their restriction to single node...
Techniques and Tools for Traceability

types (i.e., from a single set)\(^1\). The exceptions are bipartite graphs which partition nodes into two (disjoint non-empty) subsets \(N_1\) and \(N_2\), such that each arc connects a node of \(N_1\) to a node of \(N_2\) (i.e., \(x \in N_1 \land y \in N_2\)). Hence bipartite graphs allow the possibility of associating, for example, artifact nodes of type requirements and design, thereby supporting a form of limited vertical traceability (a separate graph would be required for horizontal traceability).

As an alternative to the digraph representation shown in figure 2.1, the set of \(n\) nodes may be ordered and formed into an \(n^2\) matrix (or \(n_1 \times n_2\) for a bipartite graph), termed the adjacency matrix \(A\) (see figure 2.1 - left). Cell entries in \(A\) either indicate the presence or absence of an arc between nodes (using a 1 or a 0), or show the appropriate arc weighting. Thus, an adjacency matrix \(A\) describes reachability via paths of length 1. The property of reachability for a digraph is defined in terms of reachable nodes, such that a node \(n_j\) is reachable from node \(n_i\) if there is a path from \(n_i\) to \(n_j\). Paths greater than length 1 are computed using boolean matrix multiplication of the adjacency matrix \(A\); e.g., \(A^2\) gives reachability via length 2 paths and \(A^n\) reachable via length \(n\) paths. Hence the reachability matrix \(R\) (figure 2.1 - right), which is computed as the boolean sum of \(A\), \(A^2\), \(A^n\), indicates the presence of any path of length 1 to \(n\) between nodes. Furthermore, entries in \(R\) belong to the transitive closure of the relation \(p\), the adjacency relation of a directed graph; i.e., for the set of nodes \(N\), if \((n_i, n_j)\) is an ordered pair of nodes, then the binary relation on the set \(N\) is \(n_i \leftrightarrow n_j\) there is an arc from \(n_i\) to \(n_j\).

The graphs and matrices considered thus far adhere to basic mathematical principles and are what we term lower-order cross-referencing techniques. As such, any traceability approach utilising them represents a single function and therefore permits associations of one or at best, two artifact (node) types over one relationship (arc) type. In contrast, higher-order cross-referencing techniques relax some of the mathematical restrictions, whilst building on these techniques towards a more practical application. For traceability, this means being able to represent multiple node types corresponding to different development and assessment artifacts (e.g., requirements, designs, test-cases, etc.) and also multiple arc types between these nodes capturing the various semantic relationships that exist between artifact types (allocated-to, derives, supersedes, etc.). Therefore in the following paragraphs, we consider the additional facilities provided by higher-order techniques.

2.2.1.1.2 Higher-Order: Multiple Relations

The first extension permits representation of multiple functions over a single set of nodes. Formally we define this structure as an ordered pair \((N, G)\) where \(N\) is a set of nodes (or optionally, the union of two disjoint subsets \(N_1\) and \(N_2\)) and \(G\) is a set of digraphs defined as 4-tuples of the form \(\langle A, g, W, w \rangle\). As previously described, \(A\) is a set of arcs and \(g\) is a bijective function associating with each arc \(a\), an ordered pair \((x, y)\) of nodes. Also as described, \(W\) is a set of possible weightings, whilst \(w\) is a function associating a weighting with each arc \(a\). We note that \(|G| = 1\) corresponds to a single conventional graph structure, whereas \(|G| > 1\) can be thought of as a set of graphs overlaid on top of one another; a matrix

\(^1\)A single digraph node type can be overloaded to impart greater utility to lower-order cross-referencing; i.e., it may be regarded as an abstraction of \(n > 1\) subtypes. For instance, the section number nodes in figure 2.1 may actually represent paragraphs, figures, tables and appendices, etc.
Techniques and Tools for Traceability

structure (modified to accommodate multiple cell entries) can again be used as an alternative representation.

Consider an example \((N, G=\{<A_\alpha, a, W_w>, <A_\alpha, c, W_w>, <A_\alpha, s, W_w>\})\) where \(N\) is a partitioned set of artifact node subsets \(N_1 \{r_1, r_2, r_j\} \) and \(N_2 \{r_4, r_5, r_6\}\) and \(G\) is a set of graphs based on three functions \(a, c\) and \(s\) (denoting alternative, conflicts and similar respectively), each with its own set of arcs and strength weightings (again \(3, 6, 9\)). Here, the weightings are the same for each graph, though that need not necessarily be the case (indeed null weightings may be used where appropriate). It is therefore necessary to distinguish which weighting belongs to which particular function; hence we define the sets \(W_a = \{\mu, \alpha, \pi\}, W_c = \{\Omega, \Phi, \Theta\}\) and \(W_s = \{\sigma, \Theta, \Theta\}\). The matrix in figure 2.2A integrates possible graphs of the functions \(a, c\) and \(s\) shown in figures 2.2B, 2.2C and 2.2D respectively.

![Figure 2.2 - 'Example Matrix Representation of Multiple Weighted Directed Graphs'](image)

2.2.1.3 Higher-Order: Multiple Artifacts

Whilst the higher-order: multiple relation configuration improves on lower-order representations by allowing specification of \(n > 1\) relationships, the restriction of (at most) two artifact (node) types still remains. In contrast, the final cross-reference based approach (higher-order: multiple artifact) supports both multiple association and multiple node types by effectively collocating the previous higher-order graphs (or their equivalent matrices) ‘end-to-end’. Formally, this can be stated as \(\{(N, G)\}\) where for each pair in the set, \(N\) is a set of nodes (or optionally, the union of two disjoint subsets \(N_1\) and \(N_2\)) as before and \(G\) is again a set of digraphs defined as 4-tuples of the form \(<A, g, W, w>\); end-to-end connections exist where \(\exists g_1, g_2 \in \{(N, G)\} : g_1 \neq g_2 \wedge N(g_2) \subseteq N(g_1)\).

Two further augmentations worthy of note build on this last approach. The first uses symbols to annotate sets of node pairs that share a common domain; e.g., a requirement may relate to two design strategies as represented by the pairs \((r, d_1)\) and \((r, d_2)\). Such associations could be described as being either \(\Theta\) (exclusive) or \(\Theta\) (complementary). The second extension in which arcs are annotated with function names takes us into the area of semantic networks (Quillian, 1968), a classical knowledge representational technique used to state propositional information. An example of a semantic network.
Techniques and Tools for Traceability

associating requirements (r), design (d), implementation (i), test (t) and test-result (tr) elements is shown in figure 2.3; note weightings could have been used to further enhance expressiveness of the conflicts-with function.

Figure 2.3 - ‘Example Semantic Network’

2.2.1.2 Traceability Enhancements

From a traceability perspective, the means by which both lower and higher-order cross-referencing builds on the basic concepts of sets and relations can be seen to be largely structural; i.e., through the visual representation of underlying mathematical concepts. We note that predicate logic and standard set operations may be used to define implementation independent integrity constraints (e.g., preventing loops or cycles), and that the literature features a number of established graph algorithms, including those for traversal and computation of reachability matrices. However, since graph theory per se includes neither a constraint sub-language nor pre-defined operators for standard analyses (i.e., a graph calculus), it (and by implication, lower-order traceability approaches) can only be judged on the structural properties of links and nodes. Furthermore, (and as previously indicated) these properties are relatively weak when compared to such failings as the inability to represent elements from multiple sets, to describe these elements in terms of attributes and to capture the semantics (in a data modelling sense) of their associations.

The full extent of support for traceability afforded by lower-order approaches can therefore be said to come from combining labelled (bipartite) nodes with arcs enriched by weighting attributes. For this reason we assert that ‘pure’ graph-based approaches are the least sophisticated of all traceability techniques. In contrast, higher-order techniques that build on such basic approaches, and in particular semantic networks, enable representation of unlimited node and relationship types and hence support more detailed modelling of real world concepts. However, like lower-order approaches, the nodes themselves are atomic and so have no descriptive capabilities. In other words, both forms of technique tell us what elements are related, and how, but having navigated to a relevant node, we have to look elsewhere for the actual detail.

2.2.1.3 Traceability Applications

The principles of cross-reference based traceability are a feature of most development and assessment documents. Outwardly, footnotes and phrases like ‘see section x’ are merely navigational aids directing readers to a particular section, glossary or appendix. However, their underlying structures form graphs similar to those described above. Indeed, the use of such ‘stock-phrases’ implies a lower-order
approach based on a single function with either common or bipartite node types connected over a single set of (implicit) arcs.

In principle, the approach can be extended to support higher-order cross-referencing procedures by using multiple node and statement types; e.g., in figure 2.4, Requirement A, Requirement B, Design Decision X and Text Case Y (together with their respective section and paragraph numbers) are all (3-tuple) nodes, while the associations ‘determines (strong)’ and ‘determined-by (strong)’, etc. are labels with strength weightings on implicit (forward and backward) arcs.

Enumeration of different node and link types such as these, together with specific formats and guidelines on usage can be stipulated as part of an organisational traceability policy (Sommerville & Sawyer, 1997). However, without enforced typing it may be difficult to ensure all cross-references are rigorously maintained by project personnel.

| [Requirement A | section i, paragraph j] |
|-----------------|
The Fuel System pipes and equipment shall be designed to a maximum normal working pressure of <V> psi (including surge). [determines (strong) Requirement B, section x, para. y]; [fulfilled-by Design Decision X, section n, para. m]; [verified-by Test Case Y, section f, para. g]. |

| [Requirement B | section x, paragraph y] |
|-----------------|
The Fuel System pipes and equipment shall be designed such that the ultimate pressure equals 2.5 times the normal working pressure. [determined-by(strong) Requirement A, section i, para. j]; [fulfilled-by Design Decision X, section n, para. m]; [verified-by Test Case Y, section f, para. g]. |

| [Design Decision X | section n, paragraph m] |
|-----------------|
Use <X> to maintain Fuel System pipes and equipment within pressure tolerances ... fulfills Requirement A, section i, para. j]; [fulfills Requirement B, section x, para. y]. |

| [Test Case Y | section f, paragraph g] |
|-----------------|
Test rig simulation of refuel, defuel (suction and pressure), and transfer operations ... [verifies Requirement A, section i, para. j]; verifies Requirement B, section x, para. y]. |

Figure 2.4 - ‘Example of a Defined Text Based Cross-Reference Format’

In representing different perspectives on the target system, practitioners typically employ a range of modelling techniques, each of which introduces sets of named entities, processes, behaviours and so forth. To ensure the view the models provide is a coherent one, adequate mechanisms must exist to manage such data. For development artifacts this is normally accomplished through the data dictionary, a centralised (higher or lower-order) cross-reference base listing the name, format and usage of all

2 The juxtaposition of the artifacts (nodes) shown in figure 2.4 and the inclusion of both forward and backward traceability links, suggests some redundancy. However, in reality these elements may be diffused across several hundred pages of a document, whereupon the links would become essential for navigational purposes.
elements employed across the set of system descriptions. The data dictionary also maintains constraints governing consistency of the data set and flows between the various models; e.g., all DFD (Data Flow Diagram) data stores and ELH (Entity Life History) entity types are traceable to the Entity-Relationship model, and all ELH events are referenced in DFDs. Likewise, for assessment artifacts, most regulated industries use a Hazard Log as the centralised safety document to cross-reference and track the results of analysis (Hansford et al., 2000). Its main purpose is to enumerate the hazards identified at each stage, together with their severities, probabilities, risk, causes, consequences and intended measures for exclusion or mitigation. Also referenced are the actual models and analyses that derived this information, such as fault trees and FMEA. Again, the level of detail (i.e. whether the cross-references are maintained in higher or lower-order form) should be stated in either project specific procedures, or as part of general organisational traceability policy.

Explicit graphical traceability structures (where the visual representation is preserved, instead of being implicit to the underlying structure of some document or text) are also common. For instance, lower-order approaches based on single node and arc types lend themselves naturally to the representation of artifacts linked by some form of forward or backward chaining. Examples include the Dependency Structure (Riddle & Saeed, 1998) featuring an influenced-by function over a set of Module nodes, and the Impact Structure (Saeed et al., 1995) which is effectively a graph-based equivalent of traditional N×N charts (Lano, 1979). Meanwhile structurally at least, fault trees can be thought of as a basic form of higher-order approach which, despite representing a single set of arc types (causality) over a single set of node types (faux), include higher-order augmentations supporting disjunction and conjunction.

Goal-graphs (Mylopoulos et al., 1992 and Chung et al., 1995) are a further example of a higher-order approach; these structures relate a set of system development goals over satisficing and correlation link types with augmentations that include use of link correlation weightings and argument annotations to further enrich the arcs. Finally, examples of higher-order approaches based on n > 1 node and arc types include Safety-Specification Graphs (de Lemos et al., 1995) and the Design Rationale Capture System referred to in Chapter One (and described further in Chapter Three).

Traceability applications of matrices are just as prominent in the literature. In particular, they form the basis of the Quality Function Deployment (QFD) methodology (Brown, 1991; Maier, 1993; West, 1991), a product realisation strategy developed in the motor industry, but since applied across a range of applications (cf. Bellagamba et al., 1993, Jacobs & Kethers, 1994). As figure 2.5 indicates, the basic tenet of QFD (also known as ‘House of Quality’ or HoQ) is to ensure that each customer requirement is addressed by a design element and, that no design forms part of the final specification unless relevant to some customer requirement. Hence, analysis of the main relationship matrix seeks to identify empty rows (i.e., unfulfilled requirements) and empty columns (spurious design components). Figure 2.5 also shows how the correlation matrix (or roof) establishes complementary and conflicting strategies for in this case, designs parameters; e.g., Built-In-Test-Equipment can be seen to have a positive impact on

---

3 We note that Impact Structures can support two types of impacts relationships, those derived from information in a specification document and those based on domain knowledge.

4 In Mason & Saeed (1998), we represent disjunction and conjunction as two distinct forms of causality relation which strictly places FTA among the higher-order/multiple-relation cross-reference classification.
Techniques and Tools for Traceability

Mean Time To Repair, but a negative impact on Mean Time Between Failure (owing to the increased number of components).

The relationship matrix shown in figure 2.5 essentially captures the level of information in a labelled, weighted, bipartite graph (but with symbols instead of numbers to denote strength weightings). This particular example is therefore strictly a lower-order application of a potentially higher-order technique. In practice, it would normally be extended to include other artifact types (test cases, implementation elements, etc.) by combining matrices end-to-end to form so-called "cities" of quality and by capturing different functions using multiple cell entries.

In addition to QFD, Fischer & Walker (1979), Davis (1990), Ince et al. (1993) and Moore (1993) provide further (mostly lower-order) examples of matrix based traceability. Matrices are also an integral feature of the Hatley & Pirbhai (1987) methodology for real-time systems development, as well as popular documentation standards such as ESA-PSS-05 (ESA, 1991).

![Figure 2.5 - ‘An Example Quality Function Deployment (QFD) Matrix’](image)

Most applications of cross-referencing trace artifacts in the horizontal and vertical dimensions. However, Dick (1999) uses matrices to manage revisions and variants using an extended lower-order approach. A Project/Features Index (PFI) lists past and current projects (x-axis), together with the features, or alternatively modules they incorporate (y-axis); alpha-numeric cell entries denote the variant (alpha) and revision (numeric) of modules used on each particular project. Figure 2.6 shows a hypothetical example cross-referencing component modules against aircraft projects; i.e., it captures the partial function incorporates: \((\text{Module} \times \text{Variant} \times \text{Revision}) \times \text{Project} \rightarrow A\) (where \(A\) is an implicit set of arcs). The main aim of PFI is to promote and help manage reuse. Where analysis shows sufficient
commonality exists, a previous requirements configuration can be simply copied as the baseline for a new project and then modified accordingly.

![Table with traceability information]

**Figure 2.6 - 'An Example Project/Features Index (PFI)'**

We note that the terms traceability *matrix* and traceability *table* are often used interchangeably (cf. Sommerville & Sawyer, 1997; Polack, 1990). However, whereas matrix cell entries may only indicate associations between up to two entity types, each row of a table is capable of representing relationships among \( n \geq 1 \) types (which is consistent with higher-order techniques). Furthermore, while matrices are essentially just collections of 'pointers' among identifiers, table entries normally contain actual information. Therefore, structurally at least, tables exhibit basic data modelling capabilities as is evident from the Relational Model considered in subsection 2.2.2.2.1.

A table can be said to comprise (in database terminology), an intension (column headings naming the entity sets concerned) and an extension (a set of occurrences consistent with the intension). Typically, the left-most column in a traceability table is used as an identifier on which the other column types are functionally dependent; if we adhere to the basic representation of cross-references used throughout (i.e., a set of \((x, y)\) node pairings into a set of arcs), then that column becomes the first element of pairs in the domain of each function. Examples and variations of tabular traceability approaches are described by Armstrong (1993), Hermens (1991), Jackson & Renton (1993), Mejzak (1990) and Polack (1990). Note also the concept of a *traceability-list* (Sommerville & Sawyer, 1997), a two column table in which the first column is an identifier (typically a requirement) and the second, a set of identifiers of related artifacts.

Tables are also the staple means of representing safety information, be it the results of a particular analysis technique (cf. FMEA and HAZOP), or summaries of assessment sub-processes. Indeed tabular structures dominate the range of admissible certification data listed in ARP 4754. This includes FHA,
PSSA and SSA summary tables, validation and verification 'matrices' (again, strictly tables) and the configuration index (detailing all physical elements of a system and their interconnections). Table 2.1 features an example PSSA table showing (intra-micro) vertical traceability between requirements and designs (together with a remarks column that itself includes cross-references to supporting documents). Another standard advocating use of tabular structures is Def. Std. 00-55 (Annex E) which proposes them as a means of representing safety arguments (with columns for claims, arguments and evidence/assumptions).

<table>
<thead>
<tr>
<th>Safety Requirement</th>
<th>Design Decisions</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The probability of “BSCU Fault Causes Loss of Braking Commands” shall be less than 3.3E-5 per flight.</td>
<td>Dual channel BSCU design.</td>
<td>The overall BSCU system can reasonably satisfy this requirement - See FTA page 'X'.</td>
</tr>
<tr>
<td>2. The probability of “Inadvertent Braking due to BSCU” shall be less than 2.5E-9 per flight.</td>
<td>Each BSCU system contains independent command and monitor channels</td>
<td>BSCU integrity can achieve this requirement - See FTA page 'X'.</td>
</tr>
<tr>
<td>3. The BSCU shall be designed to Development Assurance Level A.</td>
<td>Development of the Command Channel to Development Assurance Level A and the Monitor Channel to Level B</td>
<td>Development Assurance Levels assigned according to guidance in Section 5.4 of ARP 4754</td>
</tr>
</tbody>
</table>

Table 2.1 - ‘PSSA Safety Requirements & Design Decisions Table’ (source, ARP 4761)

Finally, we note that graphical and formal modelling techniques often provide their own built-in cross-referencing support. For instance, UML (Unified Modelling Language) includes the <<trace>> stereotype indicating dependencies among elements of different models (figure 2.7A). The fault tree transfer symbol is a further example (figure 2.7B); each transfer is bound to a particular event and composed of two types (comparable to forward and backward traceability): a triangle with a vertical line from its top shows ‘transfer-in’ of a fault tree section from another branch of the tree, whereas a triangle with a horizontal line from its top indicates the event is ‘transferred-out’. Similar to this are EXPRESS-G cross page symbols (figure 2.7C); relationships on separate pages terminate with a rounded box containing page and reference numbers. Page numbers indicate the location of the ‘to’ definition, whilst reference numbers distinguish multiple references onto a page. The ‘to page’ also contains page numbers of ‘from pages’ referring to a reference, whilst the ‘from page’ contains the name of the ‘to definition’. Other approaches include DFD numbering schemes (enabling navigation between process decomposition levels), the ELH ‘quit/resume’ formalism (which relates an abnormal event termination to a resumption event elsewhere in the hierarchy) and the Z A (delta) symbol (which indicates where an operation in one schema definition causes a change to occur in the state space of another) - figures 2.7D, 2.7E and 2.7F respectively.

All the techniques in figure 2.7 differ slightly from previous higher and lower-order approaches. Specifically, they have fixed parameters in terms of purpose, and the association and node types they connect. They also carry varying degrees of semantic force; e.g., whilst the EXPRESS and FTA

---

5 Stereotypes are an extensibility mechanism for defining new classes on top of the pre-defined UML kernel (Muller, 1997).
Techniques and Tools for Traceability

notations mean simply 'see page x', ELH 'quit and resumes' (analogous to GoTo statements) and Z deltas (similar to #include in C++, meaning 'look here' for schema variables, etc.) are actually control statements with inherent cross-referencing.

Tracing across UML models using «trace» stereotype

Figure 2.7 - 'Cross-Referencing Techniques of Modelling Notations'

Other applications of cross-reference based traceability schemes include those maintaining some form of explicit requirements labelling, numbering or indexing (Evans, 1989; Jenkins, 1994; Mays et al., 1985; Ramsay & Bernsen, 1995; and Yu, 1994), as well as those expressing and maintaining specified relationships between keyphrase dependencies (Jackson, 1991).

2.2.1.4 Evaluation

In evaluating cross-reference based traceability we consider its potential from two perspectives: i) as a framework for representing and navigating the underlying structure of relationships between project artifacts; and ii) as a means for reader orientation within complex, integrated texts. The first perspective concerns a basic practitioner need - to identify relationships among project artifacts. Such analysis is potentially an end in itself (e.g., where the goal is to verify non-existence of circular dependencies); else
techniques and tools for traceability

It is the means of achieving an end such that consideration of artifact detail follows the traceability analysis (e.g., determining which design nodes relate to a particular requirement and then acting to modify them as a result). Practitioners of the second perspective meanwhile are again concerned with navigation, but only in a 'reader' sense and only of final deliverables (or interim baselines). Hence they use cross-references as embedded pointers to navigate within and between artifacts along paths pre-defined by the writer (developer). Together, the two perspectives provide an insight into the role of cross-referencing in the context of a development project. Further consideration should be given to the level of support for horizontal, vertical, revision and variant traceability.

On the first perspective, directed graphs give a succinct visual representation of dependencies between nodes (artifacts) and are of practical benefit irrespective of whether the traceability activity is a means to an end, or an end in itself. They are however especially useful for highlighting instances of circular dependencies and general over use of cross-references. At a coarse granularity, graphs can also be used to aid reader orientation; e.g., the preface of a text with sections aimed at specific subsets of its readership may include a graphical steer showing suggested reading paths for analysts, designers, safety engineers, etc. Matrices derived from graphs offer an informationally equivalent though more concise abstraction of dependencies and are also amenable to automated analysis. Reachability matrices are particularly beneficial when preparing or checking artifacts containing multiple cross-references as they can help identify instances of poor structuring or unreachable nodes. They can also support change management by helping to highlight nodes along affected paths and to reaffirm their continued reachability once the changes have been made.

On the second perspective, we regard actual 'in-document' cross-references as a feature used mainly in presenting results to end-readers engaged in content analysis, rather than for the analysis of either artifact-relationships or document structures. Besides being vital to document navigation, cross-references are compatible with human thought processes. Indeed, readings from the social sciences suggest a purely linear exposition actually conflicts with the brain's non-linear character (Buzan, 1989). However, multiple cross-references (either high or low level) can disrupt the flow of a document and prevent transfer of ideas from writer to reader. The problem lies in maintaining awareness of navigational paths, whilst understanding the information in course. As a result, errors may be concealed which with safety related material (notably safety cases), can compromise the integrity of a system. We note that recent work on web-based technologies indicates the need to base document structures on conceptual models that take into account why and by whom they will be used (Smith et al., 1997).

In principle (and despite the inability to describe properties of nodes), cross-referencing is capable of supporting traceability across all the dimensions discussed in Chapter One. Inevitably, restrictions on type mean lower-order approaches struggle to go beyond simple horizontal traceability, or at best, vertical traceability using bipartite nodes (without that is, recourse to overloading and the information loss that occurs when disjoint node types are treated as a single supertype). However, the multiple node

6 Exponents of this perspective might include requirements engineers and designers.
7 This perspective is appropriate to requirements review and safety assessor roles.
Techniques and Tools for Traceability

and relationship types of semantic networks do allow for representation of complex vertical traceability paths, as well as (potentially) supporting revision and variant traceability.

2.2.2 Conceptual Data Modelling

This subsection provides an introduction to conceptual data modelling and the support it provides for traceability.

2.2.2.1 Foundations

Relative to cross-referencing techniques, conceptual data models offer practitioners a greater range of ‘tools’ (or concepts) for developing an effective traceability approach. Structural modelling constructs include entity types (or classes), relationships, attributes and domains, whilst many data models also support the established abstraction principles of classification and aggregation; note that in moving from a cross-reference based to a data-modelling approach, we are effectively replacing nodes with entities and arcs with relationships. Most conceptual data models further include means for stating constraints and manipulating the data sets of populated models. Indeed strictly, a data model is defined by this trio of components - structural aspects, means to express constraints and manipulative operators (Date, 1995). Recall that these form the basis of criteria used to evaluate traceability techniques discussed in this chapter and as such, biasing the evaluation in this way may appear somewhat unjust. However, the literature suggests that data modelling and traceability have now become inexorably linked in that data modelling features map readily to the requirements for effective traceability tools (Riddle & Saeed, 1999b). In the following paragraphs, we briefly introduce key structural constructs, constraints and operators.

2.2.2.1.1 Structural Constructs

• Entities

Entities capture real-world concepts, whether they be physical (e.g., a fuel tank), or abstract (e.g., a requirement for a fuel tank). All entities with similar properties are assigned to a particular entity type or class (in object-based terminology). Note, entity types, classes and indeed ‘relations’ (as described in the original data model by Codd, 1970) are analogous to mathematical sets.

• Attributes and Domains

Entities (and relationships) may be described in terms of their characteristic features, variously referred to in the literature as attributes or properties. For instance, possible characteristics of requirements could include a unique identifier, author and specification. Individual values for these attributes are termed scalars, the smallest semantic unit of data (Date, 1995), while a named set of scalar values is termed a domain (also known as a value-set). Most of the models in the paragraphs that follow include primitive domains for integer, string, boolean, etc. From these, we may wish to define more specific value-sets; e.g. the domain of integrity levels according to ARP 4754 is precisely \{a, b, c, d, e\}. Thus, domains are simply pools of values from which attribute values are drawn.
• Relationships
The data modelling literature lacks a clear consensus on relationships, what they are and how best to represent them. Peckham & Maryanski (1988) contend they can be modelled as entities, attributes, independent connections or even as functions. A relationship is modelled as an entity if it is a distinct concept whose properties describe the actual association, rather than one of the entities being related - e.g., an Interface type between instances of a Module entity. Alternatively, relationships may be represented as attributes if the attribute of one entity points to or is derived from another; e.g., a Module entity with an attribute condition relating to the Condition type. Relationships can also be expressed as simple connections described using a verb phrase or 'claim', e.g., a realised by relationship between Requirement and Design entities. Similarly, connections may be labelled with rolenames indicating the purpose of each entity participating in a relationship. Rolenames are especially useful in clarifying reflexive relationships, or in situations where two entities participate in multiple relationships. Connections can also be expressed as entities if we have cause to describe them using attributes, or else wish to relate them to other connections; e.g., where a Requirement is realised by a Module and where this claim is itself related to a TestCase entity via a validated by connection. Finally, and though less common, we note that some textual modelling languages specify relationships through function definitions.

• Classification and Aggregation
Classification and aggregation are both abstractions originating from research in Cognitive Psychology, with Smith & Smith (1977) being the first to apply them to conceptual data modelling. Classification has two attendant viewpoints, namely generalisation and specialisation. Generalisation describes the factoring of common features among related entities to form a generic high level type, while specialisation (often seen as the basis of reuse) relates to the capture of features not already distinguished in existing types; e.g., Systems and Components may both be classified as specialisations of a generic Module entity. The generalised type is often termed a supertype or superclass and the specialised type or class, its subtype or subclass. Given that entity types and subtypes can be said to correspond to sets and subsets, then generalisation can be said to equate with the inclusion relationship. Similarly, multiple generalisation (where a subtype incorporates features from \( n > 1 \) supertypes) may be thought of as the intersection of two sets that are not subsets of the same superset.

Aggregation is simply a special and stronger (in the sense of coupling) form of relationship. Such relationships are transitive, bi-directional, asymmetric and possibly reflexive and are used to show that one kind of entity (the whole) is composed-of (i.e., contains), one or more other entities (the parts); e.g., in the context of ARP 4754, an aircraft can be said to comprise several system modules, each made-up of many item modules that in turn contain multiple hardware and software modules. In other words, aggregation expresses the semantics of 'has-part/part-of' or 'has-component/component-of' associations. It is also worth noting that while seldom treated as such in the literature, relationships between entity types and their attributes are merely a particular form of aggregation.
2.2.2.1 Constraints

Most conceptual data modelling techniques support means for expressing constraints over model constructs (a feature largely absent from cross-referencing approaches). Notwithstanding domains (discussed previously), the most basic form of constraints are those reflecting 'real-world' restrictions on relationships between entities. Such constraints are said to specify cardinality and participation. Cardinality describes the number of possible relationships for each participating entity; e.g., a single requirement statement may produce (and therefore relate to) several design artifacts. This is usually expressed as a ratio, in this case one-to-many. A more precise constraint may limit entities to some lower or upper bound; e.g., 'good practice' may restrict the number of derived requirements spawned by a source requirement to thirty.\(^8\) Conversely, participation constraints describe whether the existence of some entity depends on it being related to another entity through a relationship. Such participation is referred to as either total or partial, or more commonly, mandatory or optional.

The other type of restriction of interest to traceability practitioners can be broadly categorised as integrity constraints (or static dependencies) which typically verify consistency and completeness of a data set. Consistency checks are necessary, for example, to maintain uniqueness among elements of sets (such as requirement identifiers), or to ensure referential integrity where models are related via a common entity (cf. Klein, 1993a; and Pearson et al., 1998). Conversely, completeness checks are necessary to verify 'required data'; e.g., in the structure for recording failure behaviours by Pearson et al. (ibid.), a constraint ensures that for each Failure, there is at least one Error leading to that Failure, and at least one Fault stated as a consequence. A proviso of integrity checks for completeness is that they should not inhibit partial population since the nature of the aerospace domain for example, means engineers often have to work with incomplete information. Support for specification of integrity constraints is widespread (though by no means universal) among conceptual data models. The most leverage comes from those enabling formal definition of invariants using predicate logic, set and boolean constructs.

2.2.2.1.3 Operators

The final issue of concern to traceability practitioners and again a deficiency of cross-reference based approaches, is support for manipulating the data set of a populated model; i.e., the ability to produce 'new' information from existing elements. The first form of manipulation relates to means of specifying queries over a data set, to select and retrieve a subset of elements according to specified criteria using either standard or user defined query expressions. Another form of manipulation describes means of specifying expressions projecting views on the data model and which typically evaluate to the complete data set for a subset of elements reflecting (for instance) the information needs of particular stakeholder groups. In both cases, manipulation merely implies packaging (or re-packaging) of existing information. A further basis for manipulation (and a more authentic 'take' on newness) relates to the use of deductive logic to infer new propositions; i.e., rules operating over a subset of elements within a model (or group of models) that derive information populating other elements. It should be noted that in each case, the ability to traverse complex hierarchical structures is of paramount importance. Moreover,

---

8 More may suggest the original requirement is either vague or trying to state too much.
while most conceptual data models include some basic operators, they tend to be less well developed than means of specifying constraints and structural features.

### 2.2.2.2 Representative Conceptual Data Models

Having introduced the foundations of conceptual data modelling approaches, we now consider a number of actual models embracing some or all of the constructs discussed (i.e., entities, relationships, constraints, etc.). The ordering is arbitrary, although we begin with the Relational Model since it is widely regarded as being the very first data model, and conclude with O-Telos and UML/OCL which together provide a basis for developing the MATrA framework.

It will be shown that conceptual data models may be represented using a graphical and/or a lexical formalism; graphical representations are effectively higher-order graph-theoretic approaches (i.e., multiple-overlaid graphs capable of capturing $n > 1$ node and relationship types), but with a range of additional capabilities. Both employ a finite set of pre-defined symbols (graphical icons and words respectively), with rules on how they may be composed.

#### 2.2.2.2.1 Relational Data Model

Based on set theory and predicate logic, the relational model (Codd, 1970) supports data structuring, integrity constraints and manipulation operators. The data are structured as tables of records (one table per entity type), with each row or tuple (horizontal subset) corresponding to an instance of a record and each column (vertical subset), an attribute (whose value is drawn from a domain) describing a particular facet of the tuples. However, neither classification nor aggregation (other than entities being composed of attributes) are a feature.

The relational model enforces constraints on data integrity through keys. Specifically, the primary key is the (minimum) combination of columns (attributes) within a table necessary to ensure uniqueness of each tuple. Mappings between tables are handled through foreign keys, such that the value of a set of attributes in one table matches those of the primary key in another. The latter is a major limitation of the model; handling logical relations implicitly through shared values means associations can neither be named nor given attributes. It also means the semantics of relationships are embodied in query operations and that users of tools implementing the model must know which attributes define inter-relational connections in order to extract instances of such mappings.

Further, the leanness that comes from expressing data in 'third normal form' (a desirable model property that removes repetition and hence scope for inconsistencies) results in fragmentation of the tables. This is especially true where the multiplicity of association is $M..N$, or when relating objects of the same type (e.g., to support horizontal traceability). As a result, there is likely to be some divergence between the entities and relations as they exist at the conceptual level and the collection of tables representing them. However, despite these limitations, manipulative aspects of the model are strong, supported as they are through a relational algebra which defines a range of operators over the data. These comprise traditional set operations for union, intersect, difference and Cartesian product, as well
as unique relational operations for projection, join and divide.

### 2.2.2.2 Entity-Relationship Model

As its name suggests, founding constructs of the Entity-Relationship (E-R) model (Chen, 1976) are entity and relationship types. The former are denoted as rectangles and the latter as rhombi (both name bearing) on relationship arcs connecting entities. Attributes are represented as annotations on both entities and relationships, although the model lacks a convention for denoting either unique identifiers or domains. Existence dependencies (termed weak entities) are enclosed within a double rectangle, but in Chen's original work at least, neither classification nor aggregation (save for entities and their attributes) is supported. Cardinalities of the form one-to-one, one-to-many and many-to-many can be specified, while optional and mandatory relationships are denoted using single and double relationship lines respectively. On the issue of integrity constraints, the E-R model includes a set of in-built rules corresponding to those for foreign keys in the relational model; in-built because a pure relational system requires formulation of explicit foreign key rules, whereas E-R demands only that users state the kind of relationship involved. Finally, in terms of operators, the model basically provides a subset of those in Codd's work (subset in the sense that there is no explicit join, for example). However, this aspect is generally less clearly defined than structural aspects.

A number of enhanced or extended E-R models (EE-R) have emerged, including those by Czejdo et al. (1992) and Gogolla (1994), while Gogolla & Hohenstein (1991) and Parent et al. (1989) are among those seeking to give the model a formal semantics. Additional concepts proposed by EE-R models include classification (normally denoted by a rectangle within a rectangle), as well as the notion of table types - preliminary text listings of attributes for each entity. Table types may be optimised through normalisation and thus map intuitively onto the relational model (with some loss of subtypes and weak entities). Figure 2.8A shows a simple example of the ER approach illustrating a subset of the principles found in Chen's original model. Elements from this example will be used to demonstrate all the techniques in the subsections that follow.

### 2.2.2.3 Object-Role Modelling (ORM)

Object-Role Modelling (Halpin, 1998) also known as the Natural language Information Analysis Method (NIAM) is a graphical modelling technique in which 'facts' or *predicates* are described as combinations of objects (entities), attributes and roles (relationships). Objects are represented using ellipse icons and attributes as circles (both name bearing); object identifiers (termed labels in ORM) are shown as dashed circles, while a plus sign (+) within an attribute circle indicates a calculable sequence number (typically system generated). Roles are analogous to relational tables, or columns within tables to be precise; i.e., they are the foreign keys to the entities being related. They are represented by adjacent (name bearing) rectangles and connect objects to objects and objects to attributes via solid lines (no separate notation exists for aggregation), allowing the relationship of an attribute to its object to be precisely defined. Finally, classification is represented using a 'heavy' arrow pointing from the supertype.
ORM provides a number of symbols for capturing constraints. For instance, domain constraints can be listed and attached to attributes in braces (if values are ordered, the range may be declared by separating first and last values with ". . . ; e.g. \{a_1 \ldots a_n\}). Cardinality meanwhile is denoted using double-headed arrows; with one-to-many relationships the arrow is on the 'many' side, whereas for one-to-one relationships, it appears on both sides. If the relationship is many-to-many, arrows span both halves of the role rectangle (i.e., both halves are required to identify each occurrence in the relationship). A circled 'U' symbol denotes instances where two or more attributes or relationships are required to establish uniqueness, while mandatory role (participation) constraints are designated by placing a solid circle adjacent to the appropriate object or attribute. Many further symbols exist for the specification of additional constraint types, including frequency constraints (imposed over roles, meaning instances must 'play' a role \(n\) times) and ring constraints (indicating that binary relations formed by the role population must be irreflexive, intransitive, acyclic, asymmetric, antisymmetric, or symmetric).

ORM has evolved through several iterations, including a number of extensions supporting query operators. Of note are RIDL (Reference and IDEa Language), a hybrid declarative and procedural language and ConQuer (Conceptual Query) which enables ORM models to be queried without knowledge of the underlying schema (unlike the relational model).

It can be seen that ORM is a highly expressive language, although the corollary is that diagrams can often appear cluttered. Figure 2.8B shows a simple example illustrating a subset of the principles discussed.

### 2.2.2.2.4 GEM

The General Entity Manipulator (Zaniolo, 1983) is a textual modelling language that extends Codd’s relational model. GEM is based on the elements entity and attribute; entities are made up of attributes which may be atomic, set-valued (i.e., elements from a domain), a generalisation list (means for classification allowing definition of sub-entities), or a reference relating to another entity (providing support for aggregation). GEM has some similarities with object-based approaches in the sense that entities can conceptually contain other entities (as opposed to having foreign keys that are pointers to those objects). The dot notation is used as a means of referring to the ‘join’ paths necessary to access contained entities. Participation of reference attributes is mandatory (i.e., a value must be supplied) unless explicitly defined as ‘null allowed’. GEM also includes the notion of alternative attributes allowing for example, instances of a Module type to populate either a weight attribute or a language attribute depending on whether they represent a physical or logical entity.

Like the relational model, designated key attributes enforce uniqueness, while domain rules restrict set-valued attributes. Essentially, these along with the treatment of null values constitute the extent of support for constraints in GEM. However, a simple but powerful sub-language (based on QUEL) provides a range of operators for specifying queries and updates over the data model. Again, figure 2.8C presents a basic example illustrating a subset of GEM constructs.
Techniques and Tools for Traceability

Figure 2.8 - ‘Examples of Conceptual Data Modelling Techniques - A (ER Model); B (ORM); C (GEM); D (IDEFIX); E (DAPLEX); F/G (EXPRESS); H (O-Telos)’
2.2.2.2.5 IDEFIX

IDEFIX (FIPSP, 1993) is a graphical, quasi-relational modelling approach developed by the United States Air Force. Basic constructs are the entity, attribute and relationship; entities are denoted by round-cornered or square-cornered rectangles (with names appearing above). The former indicate dependent entities whose unique identifier includes at least one relationship to another entity, whereas the latter signify independent entities whose identifiers are not derived from other entities. In both cases a dividing line separates identifier attributes (above the divide) and non-identifier attributes.

Relationship lines (bearing an appropriate name) associate entities. In addition, the foreign key (fk) 'implementing' that line features as an attribute of the corresponding entity. Contrary to most modelling notations, different line styles and symbols are used to describe different combinations of participation and cardinality. For instance, a dashed line between two entities A and B, with a solid circle adjacent to entity B has the semantics 'one to zero or more'. Conversely, a solid line between two entities A and B, with a solid circle and a 'P' symbol adjacent to entity B (where B is dependent) has the semantics 'one to one or many dependent'. All told, IDEFIX specifies no less than twenty four symbol combinations and we therefore refer readers to (FIPSP, 1993) for further details.

IDEFIX also supports supertype (generic entity) and subtype (category entity) classification through the notion of 'categorisation relationships'. However, neither aggregation (except through attributes), nor operators are a feature. Moreover, semantic anomalies mean certain situations may be represented by more than one set of symbols, while the same symbol can mean different things according to context. This both reduces readability and makes the notation difficult to learn. It is also worth noting that IDEFIX effectively imposes third normal form and therefore assumes all attendant dis-benefits of the Relational Model. Again, figure 2.8D shows a simple example of the approach illustrating a subset of these principles.

2.2.2.2.6 DAPLEX

DAPLEX forms the data definition component of the Functional Data Model (Shipman, 1981). Although arguably the best known example, it is nevertheless just one of several attempts to construct a modelling approach based on functions rather than relations. In general, it can be said of such models that the basic navigational ('path-following') style is similar to that of object-based approaches in terms of addressing objects that are functionally related to other objects, that are functionally related to other objects and so on (Date, 1995).

The most striking features of DAPLEX are its simplicity and syntactic leanness. Whereas most conceptual data models feature a range of modelling constructs, DAPLEX relies on just two, the function and the entity (i.e., no relationships, attributes or value-sets). A function declaration can take one or more parameters and return a set of entities of a given type. Entities are specified using functions that take no parameters, whilst attributes are defined as functions taking the entity to which they belong as a parameter. Relationships are also represented as functions. A number of built-in types are provided including entity, and simple types such as integer and string. No explicit means are provided for
representing either classification or aggregation, although users can define their own functions for such purposes.

Constraints are specified using the model's functional programming language, though even basic restrictions are far from straightforward to impose. Finally, a powerful query sub-language enables manipulation of the data-model using English-like statements and expressions. Figure 2.8E presents a simple example illustrating the main principles of DAPLEX.

### 2.2.2.2.7 EXPRESS/EXPRESS-G

EXPRESS (Schenk & Wilson, 1994) is a structured textual language based on an extended entity-relationship formalism. Its main constructs are entities and attributes, the latter specifying a data type (either user defined or chosen from one of the in-built data types such as integer, string and boolean, etc.), a cardinality and also a possible value-set; relationships between entities are subsumed in the domains of attributes. We note also that the main abstraction concepts of classification and aggregation are both supported. Attributes and derived attributes can be designated unique, whilst entities, attributes and relationships may be constrained using rules written in a procedural language. Finally, the query language EQL defines a number of operators for analysing EXPRESS data structures.

EXPRESS-G meanwhile provides a graphical means of representing models specified in the language's textual form, although only a subset of constructs are supported, including entity, relationship and cardinality. Rectangular boxes represent entities, whilst their attributes are shown as a labelled arcs branching off and connected to a domain type - either in-built (denoted by a solid rectangular solid box with a double vertical line at the end) or user defined (denoted by a dashed rectangular box); dashed and solid arcs indicate whether an attribute is optional or mandatory. The destination of relationships between entities, or between entities and their domain types is conveyed as a circle at the end of an arc, while the convention for inheritance uses bold arcs. Asterisks denote a constraint over an entity or attribute (although the constraints themselves are not shown), whilst the characters 'DER' enclosed in parentheses indicate derived attributes.

We return to EXPRESS in Chapter Three (subsection 3.2.2) when considering work by ESPRIT project SEDRES (Johnson, 1997) and also in Chapter Seven (7.4.14) when discussing future work. In the meantime, figures 2.8F and 2.8G respectively show a subset of EXPRESS and EXPRESS-G conventions.

### 2.2.2.2.8 O-Telos

O-Telos (Jarke et al., 1995) is of particular interest in the context of this thesis as object management features of its implementation in ConceptBase are used throughout to provide a 'flavour' of tool support for the MATrA structures. O-Telos is an extension of Telos (Mylopoulos et al., 1990) and includes the key structural mechanisms of specialisation, aggregation and association, as well as means to specify

---

9 It is debatable whether the thickness of a line should carry semantic force, although the same criticism also applies to ORM (discussed in subsection 2.2.2.2.3) and also to the graphical form of O-Telos (to be considered in 2.2.2.2.8).
Techniques and Tools for Traceability

constraints and deductive rules. Thus, O-Telos constitutes what its authors term a deductive object base (DOB).

A DOB is a triple, \((OB, R, IC)\), where \(OB\) is the extensional object base and \(R\) and \(IC\) are deductive rules and integrity constraints respectively. Given the set of object identifiers \(ID\) and the set of labels \(LAB\), an extensional object base is formally defined as a finite subset, \(OB \subseteq \{P(oid, x, lab) | oid, x, y \in ID \land lab \in LAB\}\); elements (propositions) \(P(oid, x, lab, y)\) of the object base are referred to as 'OB objects', where \(oid\) defines the object identifier, \(x\) is the object source, \(lab\) the name, and \(y\) the destination.

An O-Telos object base distinguishes four kinds of objects:-

- \(\text{Individuals}\) - a real world object is represented as \(P(oid, oid, lab, oid)\), where object identifier, source and destination have the same oid\(^{10}\);
- \(\text{Instances}\) - a relationship between an object \(P(oid_o, oid_o, obj\_name, oid_o)\) which is an instance of a class \(P(oid_c, oid_c, class\_name, oid_c)\) is represented by \(P(oid_rel, oid_o, in, oid_c)\) - i.e. the relationship is an object with \(oid_o\) as its source, \(in\) as its label, and \(oid_c\) as its destination;
- \(\text{Specialisations}\) - specialisation of a class \(P(oid_{cl}, oid_{c1}, class\_name, oid_{cl})\) by a class \(P(oid_{c2}, oid_{c2}, class\_name, oid_{c2})\) is represented by an object \(P(oid_{rel}, oid_{c2}, isA, oid_{cl})\);
- \(\text{Attributes}\) - \(P(oid, x, lab, y)\) states that object \(x\) has a relationship \(lab\) to an object \(y\).

O-Telos supports an infinite number of modelling levels. At the lowest level, real-world objects are modelled as individuals or Tokens. In turn, Tokens instantiate SimpleClasses which are likewise instances of MetaClasses that are themselves instances of MetametaClasses, and so on\(^{11}\). MATrA is defined in ConceptBase using the Token, SimpleClass and MetaClass levels.

Both textual and graphical representations are supported in O-Telos\(^{12}\). The textual form featured in this thesis employs a frame syntax which uses object-labels for identification: all objects with a particular object as their source are grouped around that object. For each attribute, the frame description includes labels of attribute classes. Meanwhile, the corresponding graphical notation resembles a semantic network, with \(\text{Individuals}\) represented as nodes, \(\text{Instances}\) as dashed arcs, \(\text{Specialisation}\) using bold arcs and \(\text{Attributes}\) as 'solid' labelled arcs.

O-Telos also provides an assertion language to express DOB integrity constraints and rules. These are defined as instances of the pre-defined constraint and rule classes and included as attributes of the class to which they are attached. Constraints take the form \(<\text{quantification}> <\text{formula}>\) and rules, \(<\text{quantification} > <\text{condition}> \implies <\text{result}>\), where quantification involves evaluation of universal and existential operators over O-Telos classes, formula and condition are logical combinations of literals describing relationships among O-Telos objects and result is a literal describing the logical consequence when a condition

\(^{10}\) The object model of O-Telos includes around thirty in-built constraints that maintain integrity of an object base and ensure (among other things), that each object has a unique identification and that each referenced object exists.

\(^{11}\) The uppermost layer of any object base (irrespective of the number of levels) is pre-defined by the objects \(\text{Object}\) and \(\text{Class}\).

\(^{12}\) In ConceptBase, the graphical notation may only be used to browse elements.
holds. Constraints are mainly used in this thesis to model value-sets; e.g., a class AssuranceLevel of type integer with values in the range 1 to 4 could include the constraint: $\text{AssLevel} \in \{1, 2, 3, 4\}$.

Finally, O-Telos defines operators for updating and querying an object base. Specifically, Tell and Untell operations allow for adding and deleting of objects respectively, while Ask permits their retrieval. Objects retrieved using Ask are represented as instances of the QueryClass and as a specialisation of one or more O-Telos classes. Constraints can be used to restrict possible answers by specifying particular attributes and attribute values.

Again, figure 2.8H features a simple example incorporating a subset of O-Telos constructs expressed in the textual frame syntax. An illustration of the graphical representation appears in Chapter Three (subsection 3.2.1).

2.2.2.2.9 Unified Modelling Language (UML) & Object Constraint Language (OCL)

We conclude our discussion on data modelling approaches by considering the Unified Modelling and Object Constraint Languages. The level of detail is necessarily greater than for previous techniques as they provide a foundation for the modelling of MATrA structures in Chapters Four and Five.

- **Unified Modelling Language**

The Unified Modelling Language (Rational Software Corporation, 1997a; Muller, 1997) adapts and extends modelling techniques developed by Booch (1994), Rumbaugh et al. (1991) and Jacobson et al. (1993). UML defines nine different types of diagram, each one capturing a different system viewpoint. However, we concentrate exclusively on Class Diagrams as these are used to represent all MATrA structures.

The main elements of Class Diagrams are class and relationship icons. Individual classes are represented as solid outline rectangles, with a mandatory compartment for the name, plus optional compartments in which to list attributes and operations (see figure 2.9). Note that operations are primarily used in software development to describe parameters of member functions and are therefore not of concern to us in modelling the traceability structures.

The simplest form of relationship depicted on Class Diagrams represents a dependency between (generally two different) classes and is denoted by a connecting line. Such associations may be labelled with either a passive or active verb (e.g., produces in figure 2.9), with the direction it should be read indicated using a small arrowhead. Alternatively, both ends can be labelled with rolenames that describe how each class ‘sees’ the other (e.g., architecture and build_element from the Design and ImplementationModule classes in figure 2.9). Reflexive associations linking classes of the same type can also be defined; e.g., an instance of Stakeholder with rolename responsibility_principal may be associated with another instance with rolename responsibility_holder (again, see figure 2.9). UML further supports
Techniques and Tools for Traceability

Multiplicity adornments for ‘1’ (one and only one), ‘0..1’ (zero or one), ‘0..*’ (from zero to any integer), ‘1..*’ (from one to any integer) and ‘M..N’ (from M to N integers).

Dependency associations can themselves be represented as classes (complete with attributes, operations and constraints) and so participate in relationships with other classes (including other dependency associations). We use this particular approach to represent dependencies between MATrA structures (and elements) since it also provides a useful placeholder for decision rationale. In modelling dependencies in this way we chose to use aggregation, the semantics of which are that the claim being expressed by a dependency is ‘composed-of’ the adjoining classes; see for example Validates and RealisedBy in figure 2.14.

UML denotes aggregation associations using a small, hollow diamond situated next to the aggregate class (as depicted by the above-mentioned Validates and RealisedBy dependencies and also the reflexive association on ImplementationModule). A variation of this formalism is used where attributes must participate in relationships with other classes. This is termed composition and denoted through a small, black diamond adjacent to the aggregate (as shown between Requirement and Stakeholder in figure 2.9). Composition also implies a multiplicity of zero or one on the aggregate side. Note, the nature of concepts modelled in this thesis, i.e., traceability structures composed of elements which in turn are composed of other traceability structures and elements, makes aggregation the dominant association type used throughout.

Finally, UML denotes classification using an arrow pointing from the specialised class towards the general class; e.g., in figure 2.9, Requirement, Design and ImplementationModule are all forms of Artifact. The generalised class is termed a superclass and its specialised counterpart, a subclass such that a subclass inherits all attributes specified in its superclass, together with any relationship dependencies that the superclass has against other classes. A subclass then specialises the superclass through addition of its own attributes and associations. Note, the Artifact class in figure 2.9 is abstract meaning it is not instantiated directly, but instead is used to (transparently) manipulate instances of its subclasses.

Figure 2.9 will now be used to provide a context for consideration of OCL. It must be stressed that our interest in the figure is purely as a platform for exhibiting the basic modelling constructs used in developing MATrA (and not the content of the schema itself).

---

13 Where two classes are related by more than one association, we sometimes adopt the readability convention of using a single association annotated with multiple rolenames and multiplicities. For example: 

14 Note a dashed line can also be used to attach an ‘Association Class’ to a connecting line denoting a dependency between two classes (for more information, readers are referred to Muller, 1997).
Techniques and Tools for Traceability

Figure 2.9 - ‘Example UML Class Diagram’

- **Object Constraint Language**

Notwithstanding multiplicity, UML itself has no actual syntax for specifying constraints; natural language, pseudo-code and formal expressions can all be used on condition that definitions are enclosed in braces adjacent to the appropriate model elements. However, version 1.1 prefers use of the Object Constraint Language (Rational Software Corporation, 1997b; Warmer & Kleppe, 1999) which extends UML in much the same way as Syntropy (Cook & Daniels, 1994) previously extended OMT.

OCL is a precise (and strongly typed) language with a simple, non-symbolic syntax for expressing constraints over elements of UML models. It can be used for a range of different purposes, including specification of pre and post-conditions on operations as well as detailed behavioural constraints such as guards on state transitions. However, as regards means of representing the traceability structures, we are solely concerned with specifying rules and invariants over elements of Class Diagrams.

Before discussing these aspects further, we introduce some basic OCL conventions. As figure 2.9 shows, Class Diagrams express a vocabulary of classes, attributes and associations describing the subject being modelled. OCL uses these in conjunction with its own pre-defined basic (integer, real, string and boolean) and collection (set, bag and sequence) types to form expressions. OCL types also have pre-defined operations, e.g., those over basic types include real and integer modulus, boolean implication and string concatenation (we consider collection operations later). Several meta-types that apply to all objects are also available, including oclIsKindOf (which evaluates to true if a type t is either the direct type or one of the supertypes of an object); OclType (which evaluates to the type of an object); and most importantly of all, allinstances (which gives the set of all instances of a type and its subtypes).

An OCL expression is stated within the context of an instance of a specific type. The name self is used to refer to the contextual instance. The type of contextual instance of an OCL expression is written with
the name of the type underlined as follows\textsuperscript{15}:-

\begin{verbatim}
Requirement self.requirementNo   -- evaluates to the value of the requirementNo attribute
\end{verbatim}

Another important concept when stating expressions is the convention for navigating Class Diagrams. This enables us to refer to other objects (and their properties) that are either directly or transitively related to a context class. Navigation is accomplished using the rolename of an opposite (destination) association end, i.e. self.rolename. For example, starting from the context of Design in figure 2.9, we can state:-

\begin{verbatim}
Design self.build_element   -- navigates to objects of type ImplementationModule
\end{verbatim}

Evaluation of such expressions results in the objects on the other side of the rolename association (in this case, of type ImplementationModule). If multiplicity is either '0..1' or '1', the value of the expression is an object, otherwise (as it is here), the result will be a collection or more specifically, a set\textsuperscript{16}.

If rolenames are not specified on a Class Diagram, navigation can be accomplished using the names of types at association ends. The convention is to refer to them using lower case for the initial character. Note however that if this results in an ambiguity, as for example with a reflexive association, then rolenames are mandatory. Again, from figure 2.9, we can state the following:-

\begin{verbatim}
Requirement self.design   -- evaluates to objects of type Design
\end{verbatim}

As previously indicated, the OCL collection type comprises set, bag and sequence subtypes. These support a number of pre-defined operations, among them size (cardinality), isEmpty and notEmpty (for boolean evaluation of a null set), includes and includesAll (for the '\texttt{\in}' and '\texttt{\subseteq}' set operators) and select and reject (for criteria based inclusion and exclusion of elements from operation results), together with the existential and universal qualifiers, exists and forAll ('\exists' and '\forall'). Note each collection subtype also has its own specialist operations - e.g., those for set include difference, intersection and union.

A collection operation is accessed using an arrow '->', followed by the operation name. For instance:-

\begin{verbatim}
Requirement self.design->size   -- returns the number of Design types of the Requirement self

Requirement self.design->isEmpty   -- evaluates to true if the set of Design types of Requirement self is empty

Requirement
self.design.build_element->includesAll(self.realisedBy.target)
   -- evaluates to true if the set of ImplementationModule types reachable
   -- via Design includes all ImplementationModule types reachable via
   -- the RealisedBy class
\end{verbatim}

\textsuperscript{15} An alternative to this notation precedes the contextual instance type with the keyword context.
\textsuperscript{16} Unless that is, the association carries an '{\texttt{[ordered]}}' adornment, in which case a sequence results.
Techniques and Tools for Traceability

ImplementationModule
self.realisedBy.validates.target->select (self.realisedBy.validates.target.testResult = #fail)
   -- evaluates to TestCase types whose testResult attribute has the
   -- value #fail

Note the same class can be treated as both an object and as a set, depending on whether the self.object
->set-property or self.object.object-property convention is being applied.

The forAll operation can be used to constrain all elements of a collection. Again we use the arrow syntax
with a boolean parameter; the result of evaluation is true iff the expression holds for all elements. For
instance, in the context of a Requirement:-

Requirement
self.realisedBy.validates.target->forAll (t | t.testResult = #pass)
   -- evaluates to true if the testResult of each TestCase is #pass

Likewise, the exists operation can be used to determine whether a constraint holds for at least one
element of a collection. Again, in the context of a Requirement:-

Requirement
self.realisedBy.validates.target->exists (t | t.testResult = #pass)
   -- evaluates to true if the testResult of at least one TestCase is #pass

As previously stated, the oclIsKindOf operation results in true if a type \( t \) is either the direct type or one of
the supertypes of an object. For instance:-

Requirement
self.oclIsKindOf(Artifact)  -- evaluates to true because Requirement is a subtype of Artifact

Remember also that evaluation of the oclType and allInstances operations will result in the type of an
object, and the set of all instances of a type (and its subtypes) respectively; for example we can combine
these operations to state the following expression testing the abstract property of Artifact (i.e., to test that
Artifact has no instances):-

Artifact
self.allInstances->select (oclType = Artifact)->isEmpty
   -- evaluates to true because Artifact is designated abstract

The conventions outlined above will now be used to express two simple invariants over elements of the
Class Diagram in figure 2.9. An invariant is a boolean expression which either limits the value of an
attribute or association role, or else states a relationship between the values of attributes and association
roles. The result must evaluate to true for all instances of the associated class at any point in time.

Our first example restricts values for the sizeMb attribute of ImplementationModule to less than 10Mb

\[^{17}\] The \# syntax is used when referring to enumerators of enumerated types.
(i.e., it defines the domain of permissible values). This can be expressed as follows (note use of the keyword invariant):-

```
ImplementationModule invariant
self.allInstances->forall(i | i.sizeMb < 10)
```

In the above, the allInstances operation is used to obtain the set of all instances of self, in this case ImplementationModule; forAll then constrains the appropriate attribute for all elements of that particular collection.

Our second example ensures that Requirement identifiers are unique. Again the forall operation is used, this time over the Cartesian product of all instances of self (i.e., Requirement):-

```
Requirement invariant
self.allInstances->forall (r1, r2 | not (r1.requirementNo = r2.requirementNo and r1 <> r2))
```

In keeping with the style to be used in Chapters Four and Five, we use the proof by contradiction principle to express this invariant (i.e., negate the condition we wish to hold).

The UML and OCL conventions introduced above are for reader orientation and hence consider only a subset of their respective language features. Nevertheless, they represent the main constructs used in expressing the MATrA traceability structures. For a more exhaustive consideration of UML and OCL, readers are referred to (Rational Software Corporation, 1997a and 1997b) respectively.

2.2.2.10 Other Conceptual Data Models

The previous discussion provides an overview of conceptual data modelling approaches spanning over a quarter of a century. Some older examples, e.g., GEM and DAPLEX are of little more than historical interest nowadays, although age is not necessarily an accurate barometer given that the two oldest approaches considered (namely the E-R and Relational models) remain prevalent to this day. Space prevents the inclusion of further examples, however interested readers may refer to works on Taxis (Nixon et al., 1987), SDM (Hammer & McLeod, 1981), SAM* (Su, 1986) and SHM+ (Brodie, 1984).

2.2.2.3 Traceability Enhancements

From a traceability perspective, the means by which data-modelling techniques build on basic set theoretic concepts can be seen to include structural, integrity and manipulative aspects. Structurally, all of the techniques offer rich graphical and/or lexical representations of the underlying mathematical concepts. The representations themselves are generally more expressive than graphs in the sense of capturing multiple overlaid relations, as well as enabling practitioners to describe domain elements in greater detail (using attributes) and to express stronger (in terms of coupling) set-theoretic relations such as multiple inheritance.

In addition (and to varying degrees), all featured notations further support the expression of constraints, ranging from multiplicity and participation, to entity integrity. Sometimes the mechanisms are in-built
Techniques and Tools for Traceability

(e.g., referential integrity in the relational model), while more specific user-defined restrictions can often be stated using some form of assertion sub-language (as in O-Telos). Many techniques further provide data manipulation operators, including the relational model with its select, project and join constructs and OCL with its array of built-in functions over set, bag and sequence types. It should be noted that whilst all of the featured techniques are structurally rich (relative to cross-referencing), some, such as the E-R model, require charitable reading to conclude that they are in fact data models; the integrity and operator constituents are often either weak or else not clearly defined in the literature.

Thus a degree of care is necessary when assessing the full extent of support for traceability afforded by data-modelling approaches. Those allowing the definition of sets of artifact and relationship types, with basic support for integrity constraints and some means of manipulation (e.g., the Relational, E-R and IDEF1X models) afford a basis for what is often referred to as 'syntactic traceability' (Pearson & Saeed, 1995). However, formalisms such as O-Telos and UML/OCL that combine rich structural abstraction mechanisms with a predicative sub-language, potentially permit what may be termed 'semantic traceability' (Palmer & Evans, 1994). In other words, they support precise formal definition of complex artifacts, their relationships and constraints imposed over them, as well as rules for reasoning with the data. This is the most sophisticated form of traceability achievable. It is also the most ambitious, requiring that each artifact and dependency is given a formal semantics. With MATrA, we are seeking to provide a foundation towards semantic traceability for the aerospace domain.

2.2.2.4 Traceability Applications

We purposely limit discussion on applications of conceptual data-modelling in this subsection since it constitutes a key aspect of the thesis and is demonstrated extensively throughout Chapters Four to Six.

There are numerous examples of data modelling approaches to traceability in the literature, although the emphasis is very much on presenting structural aspects, with little space devoted to semantic issues. This is perhaps understandable given that establishing the kinds of information and relationships to record is fundamental to any traceability approach and is the foundation on which the other aspects are built. The bias towards structural issues may explain why authors often present their work using arbitrary 'box-and-line' notations (which lack support for both constraints and operators), rather than using recognised data modelling techniques (cf. Laubengayer & Spearman, 1994; Ramesh, 1994). However, there are many examples of models based on the approaches considered in subsection 2.2.2.2, including Canfora et al., (1995), Eberlein et al. (1997), Herzog & Törne (1999), Mason (1996) Oliver, (1994) and Pohl (1996).

Work on data modelling and traceability tends to be situated in the horizontal and/or vertical dimensions, as is the case with these examples. Again, this reflects the more specialised nature of the variant and revision axes. However interested readers are referred to work on data modelling for variants by McKay et al. (1996) and on revisions to Rolland (1994b).
2.2.2.5 Evaluation

In evaluating conceptual data-modelling as the basis of a traceability approach, we must consider what practitioners gain, both directly and indirectly, from using such techniques.

The main product of data modelling for traceability is a logical representation of project artifacts in terms of entities, relationships and well-formedness constraints. As we demonstrate in Chapters Four and Five, the expressive capabilities of data-modelling techniques allow practitioners to represent artifacts employing text-based, graphical and even formal notations in a way that with appropriate tool support, enables horizontal and vertical traceability between them; we note this normally requires links from the data-models into appropriate tools, thereby removing the need to capture information twice - once in the tool itself and once in the data-models (an issue addressed in Chapter Three). Data-modelling further allows revisions and variants to be structured in a way that supports traceability, while depending on which notation is being used, standard operations may also be produced for projecting different views of the data.

However, this perspective belies the fact that developing a conceptual data model (or set of models) requires deep understanding of the artifacts involved in a project and the life-cycle processes that produce them. As such the modelling task itself is actually a project within a project, comprising a requirements phase to elict user needs, a design phase to develop the models themselves and an implementation phase in which the necessary tool support is produced (see subsection 2.3). Thus, improved understanding of notations used to express the artifacts being modelled, of the processes employing them and of the relationship between notations and process activities can be said to be an indirect benefit of the modelling activity.

It is important the resultant data model(s) are continually re-evaluated throughout a project. Indeed, since inputs and outputs from process activities ultimately map to model entities, any modification of 'live' data-models may indicate a growing gap between the perceived process, and that being enacted. This in turn might suggest the need for process re-definition, thereby completing a product/process feedback loop. Therefore, practitioners further benefit indirectly from the fact that data models supporting traceability make the process model more visible, enabling greater scope for process monitoring and improvement.

Finally, development of the data model(s) involves close interaction with a cross-section of stakeholder groups. This is necessary to establish not only what artifacts need to be modelled and the required ways of tracing between them, but also who will employ what information. This is especially important as project data is often used by people other than those involved in its capture. Thus, another indirect benefit of the data modelling activity is that it helps establish a control framework for accessing information, as well as determining 'ownership' and responsibilities for its maintenance and management.
2.2.3 Applicability of Techniques

Clearly, while generally inferior to data-modelling approaches, cross-referencing is still of value since the degree of sophistication an organisation requires from traceability will broadly correspond to its level of maturity\(^\text{18}\). At lower levels, a properly maintained if rudimentary cross reference-scheme is infinitely preferable to sophisticated, but neglected data-models where information becomes progressively less useful and there is little incentive to either update or use it.

It is further clear that cross-referencing techniques can benefit from tool support, whereas data-modelling requires it in order to be practical. However, tools are also essential given our domain of interest, where aerospace practitioners must often capture and analyse vast data sets. We address the issue of tools in the following section.

2.3 Tool Support For Traceability

Before considering tools supporting traceability directly, we briefly mention some other tools that provide limited traceability capabilities as a by-product of their stated purpose. These include the KJ editor (Takeda et al., 1993), the T tool test case generator (Sodhi, 1991) and AGE (Keys, 1991), as well as code analysers such as Microscope (Ambras & O'Day, 1988), MasterScope and CScope (summarised by Devanbu et al., 1991). Traceability is also an inherent feature of some languages used in the development process (cf. Bell et al., 1977 and Davis & Vick, 1977; Dubois, 1994) and in the notion of parasitic languages described by Hill (1996). Also worthy of mention are tools supporting automatic transformation of specifications into source code; research in this area dates back to (Balzer et al., 1983), whilst other works include Fraser et al. (1991), Reubenstein & Waters (1991), Börstler & Janning (1992), Duke & Harrison (1995) and Goguen (1996b).

All remaining tools considered in this section are themselves, or are built on-top of, some form of database. We note that imperative programming languages such as C++ and logic programming languages such as Prolog - both potential alternatives to a database approach - are undermined by lack of specialist capabilities for file handling and recovery.

A number of guidelines and heuristics exist for the evaluation of CASE tools generally (cf. Anderson, 1989; IEEE, 1992; Mosley, 1992; Sodhi, 1991; Thompson, 1994) and traceability tools in particular (cf. Gotel, 1995; Arango et al., 1991; Brown, 1993; Edwards & Bergstein, 1993; Fiksel & Hayes-Roth, 1993; and Kelley 1990). However, our assessment is based on work by Riddle & Saeed (1999b) as it is specific to the aerospace domain. Their 'wish-list' of features for an ideal traceability tool can be summarised as support for:-

- **Elicitation** - i) varied means of extracting data (such as forms/templates); ii) integrity checks (to be applied to new data or triggered by updates); and iii) tolerance of incomplete information.

- **Expression** - i) representation of complex hierarchical structures; ii) specification of dynamic

\(^{18}\) Maturity as implied by the evaluation frameworks introduced in subsection 1.4.6.2.
Techniques and Tools for Traceability

dependencies (enabling propagation of updates); and iii) features for extension of the underlying information schemas.

- **Analysis** - i) means of traversing complex hierarchical structures/transitive relations; ii) standard queries (including checks for consistency and completeness); iii) an expressive query definition language; and iv) predefined report formats.

We also address support for the traceability dimensions discussed in Chapter One.

### 2.3.1 General Purpose Tools

Specialist traceability tools or CASE tools providing explicit traceability support have only recently emerged. Prior to that, practitioners had little alternative but to use general purpose technologies. For cross-reference based schemes, that meant basic text editors (Davis, 1990; Kelley, 1990), UNIX *Nroff* and *Troff* text processing macros (Ni *et al.*, 1994; Yu, 1994), spreadsheets (Dean, 1992) and in particular, hypertext (Conklin, 1987) and hypermedia (Grenback & Trigg, 1994) technologies19.

#### 2.3.1.1 Hypertext

Hypertext in general provides a natural means of presenting inter-related information for browsing and on-line document analysis. However, lack of user operations on the underlying structure (database) has traditionally made it unsuitable as a basis for interactive development tools and as such, the Riddle and Saeed elicitation criteria are largely inapplicable.

From (Halasz & Schwartz, 1994) we have identified some further general criticisms of hypertext with regard to expression and analysis, including: i) weak support for link typing; ii) lack of support for consistency checks; iii) weak support for view-based presentation and lack of support for view-types; iv) lack of support for inferred links; and v) lack of support for queries and reports using selective retrieval20. We also note the general absence of mechanisms for revision control (and by implication, revision traceability).

Despite such failings, hypertext is adept at representing hierarchical document structures, as well as in handling coarse and fine-grained inter and intra-document relationships. Also (and in contrast to its graph theoretic base), hypertext systems normally provide limited node typing and can therefore support both horizontal and vertical traceability using relationships between nodes of the same and different types.


---

19 Hypertext refers to text only systems and hypermedia describes systems supporting multiple-media.

20 Note work by Pohl & Haumer (1995) - on which the MATRA structure in 4.2 is based - seeks to address issues such as view typing and selective retrieval.
Techniques and Tools for Traceability


It is also worth noting that generic mark-up languages such as HTML and its sibling XML (which both provide annotations for the digital representation of hypertext documents using tags\(^{21}\)) are now being challenged by domain specific variations. For instance, SML, a mark-up language for safety engineering includes tags for hazards, safeguards, failures and other key safety related terminology. Such languages further improve traceability, in addition to making the analysis process potentially more efficient (Fan & Yih, 1999).

2.3.1.2 General Purpose Database Management Systems

To provide tool support for data-modelling approaches, practitioners have typically employed commercial database management systems (DMS). Of these, relational databases remain the most popular despite innovations in the object-oriented and deductive fields. We base material presented in this subsection on work in DCSC investigating use of relational (Mason, 1996) and deductive object-oriented databases (Riddle & Saeed, 1997; Stephenson, 1997; Mason & Saeed, 1998; and Pearson et al., 1998) as potential platforms for traceability tools in the aerospace domain. Note that Chen et al. (1993) provide a general set of criteria against which to evaluate DMS for use in systems engineering.

2.3.1.2.1 Relational Database Management Systems (RDMS)

RDMS are based on implementations of Codd’s Relational Model developed in 1970 (and described in subsection 2.2.2.2.1). The most significant development since then has been adoption of SQL (Structured Query Language), a non-procedural approximation of the underlying relational algebra, as the official standard relational query language. Examples of RDMS include Informix, Sybase, DB2, Ingres and Oracle.

Most commercial RDMS support means to develop form-based interfaces that shield users from the underlying SQL syntax. Some systems allow these forms to include derived attributes based on arithmetic operators or aggregates. Constraints preserving domains as well as referential and entity integrity are also normally enabled, as is the definition of triggers\(^{22}\); constraints are usually attached to forms in order to validate input at source. Support for partial completion of forms typically depends on the status (i.e., optional/mandatory) defined for individual attributes when creating the underlying schema.

As regards expression, RDMS permit the representation of complex, hierarchical structures. However, ‘shoe-horning’ them into tables expressed in third normal form leads to fragmentation of the data and often creation of many intermediate tables (requiring query via outer-natural-join to prevent information loss). The notion of dynamic dependencies, enabling propagation of changes caused by updates, are handled through transactions. Essentially, a transaction is a sequence of operations that transforms the

\(^{21}\) Tags are used to structure text into headings, paragraphs and links etc., e.g., \(<h1>\) Heading \(<h1>\).

\(^{22}\) Events initiating triggers are typically updates, deletions, or insertions of tuples, with the actions to be executed on triggering defined as standard SQL statements.
database from one consistent (initial) state to another via a series of potentially inconsistent states. One notable weakness of RDMS expression-wise is that once populated, edits to the underlying schema are often either prohibited, or else difficult and cumbersome to perform.

As previously indicated, SQL provides the basis for defining standard query operations and for projecting virtual tables (or views) over a repository. However, RDMS have no inferencing capabilities and are largely unable to support recursive queries that compute the transitive closure of a relation. As Mason (1996) demonstrated, this is potentially a major weakness in terms of their potential application to the aerospace domain, although the SQL3 standard (Elmasri & Navathe, 1997) does now include an explicit recursion feature. Alternatively, SQL statements can be embedded in a host programming language such as C++ or Pascal, with recursion supported by means of loops. Finally, most RDMS systems include a report-generator for specification of pre-defined report formats, with the content again determined using query language operations.

Examples of traceability tools built on RDMS include those described in Buus et al. (1997), Cockram et al. (1998), Dorfman & Flynn (1984), Flynn & Dorfman (1990), Garcia (1994), Mason (1996), Neely & Hartley (1993), Patel et al. (1993) and Watkins & Neal (1994). These works demonstrate horizontal and vertical traceability capabilities, whereas variants can be handled by RDMS using appropriate attributes (Silva & Agusta, 1998); versioning is not normally supported in relational systems.

2.3.1.2.2 Object-Oriented Database Systems (OODS) and Deductive Database Systems (DDS)

Two approaches seeking to address the problems inherent in RDMS are object-oriented databases (OODS) and deductive databases (DDS). OODS date back to the late 1970s, with commercial products first emerging in the late 1980s. They provide rich facilities for representing both structural and behavioural information by combining features from database technology with those of object-oriented programming languages; examples include O2, ObjectStore, GemStone, Ontos and Versant. For a comprehensive review of OODS, readers are referred to McFarland et al. (1997).

In contrast, development of precursors to deductive databases can be traced back to the late 1950s and to subsequent advances in logic programming in the 1970s. This was followed by work towards efficient implementation of recursion and reinterpretation of the conventional model-theoretic perspective on databases in proof-theoretic terms (both during the 1980s). Thus, DDS combine database technology with the formal reasoning capabilities of logic programming; examples include LDL and Coral. Interested readers are referred to Minker (1988) for an account of the development of DDS and to Grant & Minker (1992) for an early assessment of the impact of logic programming on databases.

Note, while the potential exists, we are not aware of any work describing traceability tools based

---

23 Thus, where the transaction fails to complete, a rollback operation must be applied to restore the database to its initial state. As indicated in 2.3, a lack of such facilities undermines potential use of programming languages as a basis for traceability tools.
Techniques and Tools for Traceability

specifically on either object-oriented or deductive databases. They are therefore not considered further in this thesis.

2.3.1.2.3 Deductive Object-Oriented Database Systems (DOODS)

Both OODS and DDS have inherent weaknesses. For instance the former are seldom based on formal semantic models and lack declarative query languages, whilst the latter have poor data structuring mechanisms and limited facilities for update and I/O operations. Deductive Object-Oriented Database Systems (DOODS) attempt to combine these two paradigms in a manner which utilises their respective strengths to overcome these respective weaknesses, but without compromising the characteristic benefits of either. The resulting systems provide a rich modelling capability combined with a formal mathematical foundation; examples include Coral++ (Srivastava et al. 1993), Rock & Roll (Barja et al., 1995) and ConceptBase (Jarke et al., 1995), an implementation of O-Telos (described in subsection 2.2.2.2.8) that provides a target platform for the MATrA structures. Note that since the features of DOODS tend to vary significantly, we concentrate on ConceptBase for the purpose of this evaluation.

An immediate criticism of ConceptBase is its lack of either a form based or template interface, though this is probably due to being developed for research rather than commercial use. However, data can be loaded from pre-edited files and therefore a template of sorts could be provided as a commented text file if required; work in DCSC towards development of a JAVA based interface to ConceptBase for a Requirements-Engineering tool (Sukamaran, 1999) is also noted. In terms of constraints, the tool provides good support for integrity checks which are encoded as attributes of classes (although the restriction of operators to universal and existential quantifiers is a limitation); new and updated instances must conform to these constraints which can enforce various domain, real-world and entity integrity restrictions. Note, new instances of a class need not by default have values associated with the attributes defined, so enabling partial population.

The underlying object model supports definition of complex, hierarchical structures using O-Telos' mechanisms for specialisation, aggregation and association (described in section 2.2.2.2.8), while the tool's axiomatic basis means querying such structures is not problematic. ConceptBase further enables definition of deductive rules (again, as attributes of a particular class) that can be used to derive dynamic dependencies, allowing propagation of changes to related objects. As regards extendability, the underlying schema can be arbitrarily changed provided constraints and dependencies are not violated.

Standard queries can be defined via the special QueryClass construct (as introduced in section 2.2.2.2.8); these become classes within the database which may be selected from a menu and applied at any time. QueryClass can be thought of as generalising the notion of relational views in that its instances are answer objects to the query. This form of 'lazy evaluation' is one of the tools strengths and is often preferable to expressing constraints on the classes themselves which can be overly restrictive.

---

24 In providing a flavour of automation for MATrA structures, we focus on representation of the base classes and hence on object-oriented rather than deductive features of ConceptBase (see subsection 2.4).
Conversely, one area where ConceptBase is particularly weak is its reporting facilities, with no means for defining standard report formats or templates.

It should be noted that application of DOODS has largely been confined to academia, including some work on the development of traceability tools (cf. Eberlein et al., 1997 and Pohl, 1996).

### 2.3.2 Commercial Traceability Tools

This section analyses three commercial traceability tools, namely DOORS (QSS, 1998), RTM (Integrated, 1997) and RDD-100 (Ascent, 1997). We base our appraisal on a combination of 'hands-on' evaluation and discussions with various user groups, including experienced\(^{25}\), intermediate\(^{26}\) and beginner\(^{27}\) levels, together with findings in surveys by INCOSE (1999) and Riddle & Saeed (1999a, 1999b). It should be stressed that as commercial tools, DOORS, RTM and RDD-100 are constantly evolving and therefore any appraisal is likely to become rapidly outdated. We therefore indicate which particular releases were considered, whilst highlighting subsequent developments of note where appropriate.

#### 2.3.2.1 DOORS (Dynamic Object Oriented Requirements System)\(^{28}\)

DOORS is a dedicated requirements management tool capable of interfacing with the likes of Statemate, Teamwork, Rational ROSE and Word. It also provides a C-like scripting facility - the DOORS eXtension Language (DXL) - which may be used (among other things) to construct custom interfaces with non-supported tools. However, relative to this thesis at least, a more significant means of interfacing with other CASE tools, or to be precise, capturing their project data within the DOORS environment, is the 'DOORS Connect Programme'; we return to this issue below.

In essence, DOORS adopts a document-centric approach; i.e., it presents the information via a word-processor-like interface, with requirements displayed as hierarchically organised 'document' segments. Requirements may be captured in DOORS in several ways; e.g., automatic parsing mechanisms can read information from multiple file formats (including ASCII and RTF) so that structures, attributes and links may be set up without manual input. The parsers operate by analysing text and identifying requirements according to keywords. A manual mark-up facility (using mouse highlighting) is also available, while requirements and links can be further loaded in batch mode directly from file. Finally (notwithstanding the above-mentioned DOORS Connect Programme), there is the template based approach, where DXL scripts generate data entry forms with pre-defined headings for users to populate; templates are supplied for standard format requirements such as Mil-Std-498 (DoD, 1994).

DOORS uses a proprietary object-oriented database which is reflected in the way information is structured. A DOORS 'project' contains a set of 'formal modules', each one typically containing a

---

\(^{25}\) Aerospace practitioners.

\(^{26}\) Academic colleagues in the BAE Systems Dependable Computing Systems Centre.

\(^{27}\) Undergraduate and post-graduate MSc. students (cf. Duvall, 1997; Brown, 1999; Casemore, 1998).

\(^{28}\) Considers release 4.0.
specific requirements sub-group (e.g., aircraft-level, system-level, item-level, etc.); traceability is realised through links ("link modules") both within and between modules. DOORS handles individual requirements as discrete objects (which it allocates unique identifiers) organised under numbered sub-headings. Each object is described by typed attributes, some of which are standard and may be attached automatically (e.g., date of creation/last update, creator, etc.), whilst others are defined by the user (note that links can also have attributes). However, the use of headings and sub-headings naturally encourages a hierarchical structure which though supporting inheritance between levels, is not always appropriate. It should also be noted that the absence of an underlying graphical schema underlines the need for a clear understanding of entities and relationships involved in a project before creating the actual objects and links; that said, extensions (or changes) to the database are easily performed.

As indicated previously, the DOORS Connect Programme is an important feature of the tool allowing users to capture data from other CASE tools inside the DOORS environment. Technically an interface to a particular CASE tool is achieved through a ‘surrogate’ formal module which shares common (unique) identifiers with the data set of that CASE tool. Surrogate formal modules are populated automatically by directly importing data into the DOORS database. Data is synchronised between DOORS and the CASE tool by updating the surrogate module with changes made to the data set of the CASE tool. As will be seen from Chapter Three, a similar approach is used for development of the MATrA framework described in this thesis.

For simple analysis, the Traceability Explorer offers an on-line-browser tool showing all incoming or outgoing links for a particular module. Meanwhile the ‘Traceability Wizard’ allows selection of forward and backward traversal paths over links between modules. Custom analysis is achieved using filters or DXL coded queries; filters are constructed by combining filter criteria (such as operations on attributes), the results of which can be saved as views. In addition DOORS provides a ‘View Wizard’ to guide selection of filter operations, such as the columns to display and sorting operations to conduct over filter results. Views further provide the basis for producing printed reports; a view is used to select the information required which is then included in the report definition. Basic consistency checks (e.g., on attributes) may be conducted by filtering, although more complex checks (e.g., of links) require DXL code (note as a procedural language, recursive querying of hierarchical structures is relatively easy). Several standard DXL queries are provided which may be adapted or extended as needs dictate. In terms of output, DOORS provides a number of report generation templates that comply with software engineering standards (including Mil-Std-499b and ESA PSS-05), while further reports can again be created using DXL. It is also possible to extract a detailed audit trail detailing the ‘who, what, when and how’ of changes to objects for revision control purposes.

To summarise, DOORS is mainly a tool for managing textual requirements and specifications. Consequently vertical or horizontal traceability to non-textual artifacts comes either through interfaces

---

29 The ability to capture information expressed in other tools through the DOORS Connect Programme and to represent textual data from all project phases (not just requirements) - including design and testing - has recently lead to DOORS being marketed as an 'information management tool'.
into other development tools, or else through data stored as surrogate formal modules within DOORS itself. For artifacts stored as DOORS modules (including surrogate modules), vertical and horizontal traceability links may be inserted between any two object classes. DOORS also enables revision traceability through change histories and while no explicit support for variants exists, this may be partially offset through the use of appropriate object attributes.

2.3.2.2 RTM (Requirements & Traceability Management)

RTM like DOORS, is intended to form the requirements management component of a life-cycle wide CASE tool environment. Pre-defined interfaces are provided for a range of development tools (including Teamwork and Statemate), as well as text processing tools (such as MS Word). For tools without pre-defined bridges, RTM provides a C and C++ driven API (Application Programming Interface), enabling users to develop their own custom interfaces to third party tools. In addition, later releases include a mechanism for representing data transferred from CASE tools directly into RTM using concepts similar to the DOORS Connect Programme discussed in 2.3.2.1.

Normally, the first stage in using RTM is to define an information model known as the Class Definition Diagram (essentially an ER schema for the underlying ORACLE relational database). Its purpose is to capture information elements to be used on a project, the relationships between them and the rules governing these relationships. Essentially, a 'class' equates to an entity (in the entity-relationship sense), each one defined as a set of attributes and each attribute specified from a range of types (e.g., alpha-numeric, character, etc.). It should be noted that since changes to the schema once populated are difficult to make, developers are under pressure to get the schema right first time. We also note that (unlike DOORS) the need to construct a Class Definition Diagram means RTM cannot be used 'out-of-the-box'.

There are several ways of entering requirements into RTM; e.g., the Capture tool provides a language parsing mechanism for automatic identification of requirements within a source text (by keywords, unique identifiers, etc.). Users can further identify requirements interactively by highlighting appropriate document sections. Requirements and links can also be entered in batch mode by importing the data from file (such as ASCII text), or failing that, created manually using a form based editor with pre-defined fields taken from the Class Definition Diagram.

For structuring, RTM enables links to be established between any type of class in any direction via two relationship types - genealogical and generic - each representing a different aspect of requirements flowdown. Genealogical flowdown describes the decomposition or refinement of a requirement where one or more subsidiary requirements are derived from an existing higher level requirement (with flowdown links inserted automatically). Generic flowdown meanwhile supports the definition of relationships between different information types, e.g., from requirements to acceptance tests. The tool further allows users to identify inconsistencies such as orphan (i.e. unlinked) elements.

30 Considers release 4.0.
RTM provides three basic interfaces for viewing, analysing, and maintaining link and traceability information: i) the Trace tool displays links between individual items in different classes as a cross-reference matrix; ii) the Visual Network Tool graphically traces relationships between any linked classes using a tree format; and iii) the DocTool uses scripts expressed in an SQL-like query language to generate reports based on information selected for inclusion. The DocTool also provides an alternative means of generating inconsistency reports.

RTM supports configuration management by establishing separate database objects for each revision of a requirement, with further support for establishing and comparing project base-lines. In addition, the tool enables propagation of links to new revisions of requirements. This is achieved either automatically or by manual intervention, i.e. existing links are broken and must be physically re-established by the user (preventing situations where the revision is linked, for instance to an irrelevant test). Alternatively, all links are copied to the new requirements and deleted from the old revision.

To summarise, RTM is a tool for textual requirements management and as such, traceability to non-textual artifacts again comes mainly via interfaces with appropriate tools. For artifacts represented within RTM itself, both vertical and horizontal traceability are supported, although recursive queries are potentially problematic owing to the tool's underlying relational architecture (i.e., the depth of recursion involved in a query must be known in advance). As with DOORS, variant traceability is not explicitly enabled, although again, it can be achieved using appropriate attributes. Revision traceability is however supported (the RTM practice of establishing a separate object for each modification being arguably more effective than DOORS' change histories).

2.3.2.3 RDD-100

Whilst DOORS and RTM focus on requirements management, RDD-100 (which is built on the 'Visual Works' object-oriented database) provides an extensive systems engineering tool-set that in principle, reduces the need for tool integration. That said, the ASCII interface provides a basic bridge to a range of CASE tools including Teamwork, Software Through Pictures and Statemate. There is also a bi-directional interface between MS Word and the RDD-100 text editor, RDD Word.

In addition to a standard form based approach, RDD-100 accepts input in a variety of ways. Among them: i) a parser tool for identifying single or compound requirements in source documents based on key words and unique identifiers; ii) manual selection and extraction of requirements text from ASCII format documents; iii) the RDD-100 Command Reader and Report Writer facilities which can capture and create requirements in batch mode; iv) the RDD Word facility which may be used to associate MS Word formatting with entity instances in RDD-100; and v) manual entry using various diagram formats, including Function Block, Data Flow and Behaviour Diagrams. Incomplete information is tolerated provided that integrity checks have not been violated. These are encoded using the consistency

\[^{31}\text{Considers release 4.1.1}\]
Techniques and Tools for Traceability

checking mechanism and supplement standard, pre-defined constraints for consistency and completeness.

RDD-100 includes a wide range of standard system engineering entity and relationship types. These can be extended with the addition of user defined elements as needs dictate. The types are categorised in a hierarchical manner, with inheritance between different levels. RDD-100 allows users to interrelate elements between hierarchies, as well as supporting reflexive relationships. It should be noted that the linking of elements in RDD-100 is bi-directional in the sense that when a link between two elements is established, an inverse link is automatically inserted. Extensions to the database (i.e., the addition of new element and link types) are best accomplished when the database is un-populated, otherwise the complete data set must first be exported to a schema copy and then re-imported into the extended schema.

RDD-100 provides two main mechanisms for analysing the database, namely Custom Hierarchies and Behaviour Diagrams. Custom Hierarchies provide a graphical or textual view of entities and relationships based on standard pre-defined templates. For more complex analysis, Behaviour Diagrams can be used as a form of visual programming tool, exploiting facilities offered by language constructs such as loops and recursive procedure calls. One of RDD-100's real strengths meanwhile lies in its ability to produce reports, including pre-defined formats for a range of recognised documentation standards (e.g., Mil-Std-498); further, user specified styles can be defined using RDD Word. Again, behaviour diagrams are used to select information and to produce the final printed reports.

RDD-100 also provides users with the ability to track changes to data during the course of a project (using the 'creation date' and 'modification date' attributes). Revision control is accomplished through the Model Configuration Management Facility (MCMF) which as its name suggests, tracks configuration of the system throughout development and enables management not only of item revisions, but also variants.

To summarise, RDD-100 is a systems engineering tool which in addition to various functional, data and behavioural modelling, provides support for requirements traceability. On the dimensions referred to throughout, it is clear the ability to link any user defined or in-built element types (including elements of the same type) enables both horizontal and vertical traceability. RDD-100 also supports revision traceability (although like DOORS this does not extend to the maintenance of separate objects) and variant traceability via the MCMF facility.

2.3.2.4 Additional Commercial Traceability Tools

In addition to the above, a host of other commercial tools now offer support for traceability. These include: i) SLATE (System Level Automation Tool for Engineers), which like RDD-100, is intended more as a systems engineering tool; ii) Requisite Pro, a tool aimed at the requirements management market and a direct competitor of RTM and DOORS; iii) Vital Link, another requirements tool which like DOORS manages traceability via a document-centric interface; iv) XTie-RT an entry level
requirements tool with basic traceability facilities; and v) Cradle, another systems engineering tool with traceability support (this time based on the Yourdon methodology).

2.3.3 Comment on Tools

In the past, there has been a tendency in developing traceability tools (particularly those managing textual requirements) to simply automate paper based techniques (cf. Horowitz & Williamson, 1986, Lefering, 1993; Queille et al., 1994 and Singh & Han, 1996); the literature suggests this is no longer the case. However, despite advances reported by Dömges & Pohl (1998), INCOSE claim that traceability remains the on-going tool related issue among systems engineering practitioners.

It has long been felt that traceability approaches (and therefore tools) should be capable of reasoning about the artifacts they link instead of merely creating navigable paths between them (Ramesh & Edwards, 1993). However, in order to do so, models on which the tools are based must also capture the semantics of the artifacts they represent; we address this issue in the development of MATrA structures.

White (1993) highlights a lack of tool support for the tracing of behavioural and non-functional requirements such as reliability and fault-tolerance. Save a few exceptions (Boyd, 1993; Haveman & Pearson, 1997; Landes & Studer, 1995; Mylopoulos et al., 1992; Prowell & Poore, 1998; and White, 1994a, 1994b), work on traceability largely excludes these areas (possibly because they are often satisfied by an entire system and therefore cannot be traced to a particular set of design elements). MATrA partly addresses this issue through its range of Workspace notations (subsection 3.3.2) and by primitives in the structure that maintains Workspace consistency (subsection 3.3.5). We also propose topics for further work in this area as described in subsections 7.4.1 and 7.4.4.

Another problem is that again, apart from a few exceptions (cf. Ni et al., 1994; Yu, 1994; Pearson & Saeed, 1996; Silva, 1998; Tran et al., 1997), work on tool support for traceability seldom considers the method by which the tools will be used. According to Jayaratna (1994), a method generally contains guidance on steps to be taken, how the steps are to be performed and why the steps must be followed in a specified order. Though not explicitly addressed by this thesis, MATrA raises a number of method related issues and is identified as an area for future work (see subsection 7.4.6).

Finally, and especially significant as regards this particular piece of research, is the problem of CASE tool fragmentation where separate applications (often with overlapping capabilities) are used for different development and assessment activities. A lack of tool integration (to be discussed further in Chapter Three) makes it difficult to establish traceability between, and maintain consistency across, artifacts produced by these activities; this thesis seeks to address both issues.

---

32 The need to define artifacts (nodes) in detail implies a data-modelling rather than a cross-referencing approach.
33 These concepts were introduced by the thesis argument in subsection 1.3 and are discussed in more depth in the subsections identified.
2.4 Techniques and Tools for MATrA

From the investigation of tools and techniques described above it was possible to establish an appropriate technical basis for MATrA. Cross-Referencing was immediately discounted as we wish to represent artifacts (i.e., nodes) as more than atomic elements.

Regarding data modelling, UML/OCL, EXPRESS and O-Telos are the only formalisms featured in this chapter which broadly support the range of foundations outlined in 2.2.2.1. As indicated, the UML/OCL combination was our eventual choice since it is widely used by BAE SYSTEMS and throughout the aerospace sector generally. This obviously made it easier to communicate ideas to practitioners than if we were using an unfamiliar language. However, for reasons outlined in a future work item (subsection 7.4.14), there are strong reasons to consider re-expressing MATrA in EXPRESS at a later date.

As previously indicated, we chose the ConceptBase implementation of O-Telos to provide a flavour of tool support. This was motivated by the fact that O-Telos supports both object-oriented and axiomatic constructs and hence is (relatively) close in spirit to the combination of UML and OCL. That said, we largely elect to concentrate on implementing structural aspects of MATrA, while noting that expressing the constraints and rules using O-Telos may be difficult as the range of set and boolean operators available in OCL is far greater. Therefore subsection 7.4.13 commits to investigate various commercial tools, of which DOORS (with its procedural extension language that can potentially replicate all necessary operators) is a leading candidate.

2.5 Chapter Summary

Following on from Chapter One which provided a user view on the need for traceability, this chapter has taken an applied perspective on how it can be realised using different techniques and tools.

Traceability techniques were classified as based on either a cross-reference or conceptual data modelling approach. We further differentiated between lower and higher-order cross-referencing, and thence higher-order multiple relation and higher-order multiple artifact techniques. Various means of automation were also considered, including hypertext, general purpose database and commercial traceability tools.

The chapter concluded with a brief summary of techniques and tools used to develop the MATrA framework which we consider in the following chapter.
Chapter 3  MATrA: Foundations and Fundamentals

3.1 Introduction

This chapter considers the main works that have influenced MATrA (foundations), together with the nature of their influence and/or perceived weaknesses we address in our thesis. It goes on to introduce the basic principles of MATrA (fundamentals), their purpose and composition.

3.2 Foundations of MATrA

In this subsection we consider three key works from current literature that have influenced the development of MATrA.

3.2.1 Novel Approaches to Theories Underlying Requirements Engineering (NATURE)

Our thesis argument in Chapter One (subsection 1.3) posited the notion of a traceability Workspace as a mechanism to support traceability across data from the various tools used by aerospace practitioners. A key constituent of this Workspace is a set of meta-models capturing elements of notations supported by these tools. The origins of our approach to representing such meta-models stems from work by ESPRIT project NATURE (summarised in [Pohl, 1996]), a three year programme with the aim to develop a framework, tool environment and models for process-centred requirements engineering.

The NATURE framework supports functional, data and behavioural modelling through the following quasi-standard techniques: Structured Analysis, Entity Relationship Models and OMT (including object and dynamic models). They further provide a hypertext model for recording informal representations and an argumentation structure (based on the IBIS model - Kunz & Rittel [1970]) for asserting rationale.

Figure 3.1 - ‘Project NATURE Entity-Relationship Meta-model’
NATURE partners capture elements of these techniques in the form of conceptual meta-models (i.e., they adopt a data-modelling approach) expressed using the O-Telos knowledge representation language (Pohl, 1996); to provide tool support, they implemented these models in the ConceptBase object management system. Figure 3.1 demonstrates an example of one such meta-model (expressed in the graphical O-Telos language) which captures information elements for Entity Relationship diagrams (ER_Model), specifically entity (ER_Entity), relationship (ER_Relation), attribute (ER_Attribute), cardinality (ER_Card) and role (ER_Role).

NATURE records associations between models structured in this way using a 'dependency' class with attributes 'to' and 'from'. Dependencies are typed (and organised as a taxonomy - the Dependency Model) in order to restrict instantiation of these attributes; examples of dependencies include 'elaborates', 'based-on' and 'formalises' (of taxonomy type evolutionary dependency) and 'compares', 'contradicts' and 'conflicts' (of taxonomy type content dependency). Though drawn from a survey of requirements engineering literature, many of the associations identified (including those above) have potential for wider application in linking 'down-stream' artifacts.

3.2.1.1 NATURE Influence

MATrA is perhaps closest in spirit to the work by project NATURE. We have been particularly influenced by their use of a conceptual modelling language to structure object-based representations of various requirements notations. Accordingly, in this thesis we use a similar approach to develop meta-models covering a representative subset of notations employed by aerospace practitioners (introduced in 3.3.2), including examples used for design and implementation, for developing heterogeneous systems, for stating real-time and non-functional properties, for safety analysis, and for product management, etc. Moreover, like NATURE meta-models, those in MATrA instantiate elements of a common meta-level structure (subsection 3.3.3) such that each model (ER_Model in the above) is composed of elements (ER_Entity, ER_Relation, ER_Attribute, etc.); this is similar to the approach employed by Pearson & Saeed (1996).

We have also been influenced by the NATURE way of creating associations between meta-model representations (subsection 3.3.6.3.2). Indeed, this thesis largely concentrates on meta-modelling aspects; whilst the case studies in Chapter Six demonstrate example associations between elements of these models, the approach used is essentially as described above. Moreover, the featured associations are fairly arbitrary and a detailed investigation (including a literature survey, together with practitioner consultation) will be required to establish a set of domain specific associations. We return to this issue in the concluding chapter (subsection 7.4.3).

Finally, whilst NATURE regard their framework as autonomous (each model has a graphical interface providing an appropriate 'look-and-feel' for the notation represented), aerospace practitioners are unlikely to willingly relinquish existing tools for bespoke alternatives. Therefore MATrA aims to integrate with, rather than replace such tools; this is the basis of our SEDRES influence.
3.2.2 System Engineering Data Representation and Exchange Standardisation (SEDRES)

In order to populate Workspace meta-models, a mechanism is required to transfer data from the models underlying various CASE tools used by practitioners. As previously indicated, our thesis focuses on a framework to support traceability of this data once ‘inside’ MATrA, with the transfer process itself treated as a ‘black-box’. However, in doing so it is necessary to show feasibility of such a process based on reputable research from the public domain. For this we turn to work by partners of ESPRIT project SEDRES (Johnson, 1997) and its follow-on SEDRES-2 (Johnson, 2000).

The objective of SEDRES has been to produce a usable and standardised data exchange capability for systems engineering tools, based on STEP (Standard for the Exchange of Product Data - ISO 10303), a framework for the unambiguous representation and exchange of computer-interpretable product data. Johnson (1997) summarises the design benefits of data exchange as follows: - i) support for the movement of data between tools; ii) the possibility of tool-independent data-storage (as opposed to proprietary formats); iii) support for verification and analysis of the emerging design across tool boundaries (something that is currently both difficult and labour intensive); and iv) greater scope for the way in which a design can be viewed (e.g., using data from one or more tools).

To realise these benefits, SEDRES developed a comprehensive information model capturing primitives relevant to systems engineering. The model provides, in conjunction with STEP framework services, what Herzog and Törne (1999) describe as an ‘infrastructure’ for data exchange. We now briefly describe the SEDRES concepts underlying this infrastructure using a simple scenario (from Harris & Candy [1999]) involving the transfer of information between two tools.

![Figure 3.2 - 'SEDRES Data Exchange Concept'](#)

From figure 3.2, it can be seen that SEDRES' export interface maps data from the internal data model of a source tool, onto the SEDRES model - its boundary of influence on MATrA - and thence into a flat ASCII file (or “Part 21” in STEP terminology) for transmission to a receiving or ‘sink’ tool. The import
interface of the receiving tool then maps this file onto the SEDRES model and hence onto its own internal representation. For further consideration of these concepts, readers are referred to Johnson (1998) and Johnson et al. (1999).

The SEDRES data model is represented in EXPRESS (the standard language for STEP) and divides into several loosely coupled Units of Functionality (UoF). These include the following (readers are referred to Herzog & Scerri (1998) for the details on each):

- The Requirements UoF contains entities for representing requirements, the system under specification and allocation of requirements onto systems, functions, behaviour specifications, etc.
- The Functional Architecture UoF contains entities for representing composite and leaf functions. These can be assigned to physical elements realising the specified functionality. Support for recording how and with which external functions a system interacts is also provided.
- The Physical Architecture UoF contains entities for the specification of physical system objects, including physical component and data link definitions, physical context definition and physical ports (defining the component interfaces).
- The Configuration Management UoF contains entities supporting version and variant management. A mechanism for object-grouping (or ‘packaging’) is also provided.

### 3.2.2.1 SEDRES Influence

The Functional and Physical Units of Functionality provided a steer in developing a model to verify consistency across the MATrA Workspace (subsection 3.3.5), while the Configuration Management UoF has influenced our means of managing versions (described in subsection 5.5).

However, SEDRES' main influence on MATrA has been at a conceptual level in providing evidence of the ability to map data from CASE tools onto information models. In confirming that this is indeed possible, SEDRES has enabled us to treat all such mappings at a very abstract level (subsection 3.3.4) and to concentrate on other aspects of the traceability framework.

It should be stressed that MATrA requires the support of a slightly different form of 'tool-to-model' mapping to that shown in figure 3.2. Rather than mapping a source tool model onto a generic information model (as illustrated), MATrA depends on the ability to map a source tool model onto a corresponding notation dependent structure. We note that SEDRES also has this capability, as evidenced by the fact the project has produced several working implementations of import/export interfaces. Similarly, recall from subsection 2.3.2.1 that this form of one-to-one mapping is possible in recent releases of DOORS using 'surrogate' formal modules (Telelogic, 2001).

### 3.2.3 Design Rationale Capture System (DRCS)

The origins of our approach to maintaining consistency across the MATrA Workspace stem from the Design Rationale Capture System - DRCS a (higher-order cross-referencing) technique for expressing...
argumentation over an emerging design (Klein, 1993a). We have been particularly influenced, not by means to record the arguments themselves, but by the way in which they are placed in context. Recall from 1.4.6.1 that early models of this type were criticised for being detached from their subject matter; i.e., they merely structured elements of an argument (claims, issues, positions, etc.) and not the artifacts over which they were being expressed. What sets Klein’s approach apart is the use of a top-down representation of fundamental system elements as a basis for expressing decision rationale.

DRCS is constructed from a decision rationale language which provides a vocabulary of assertions about a system and its components. The vocabulary includes a pre-defined set of entities, and claims relating them. Klein divides DRCS into a number of self-contained information structures (connected through common entities), each focusing on a particular aspect of development information.

The Artifact Synthesis structure (figure 3.3) records the physical system architecture. Basic entities include modules, attributes, interfaces and connections. Modules can have sub-modules or specialisations, while attributes can have values expressed over them. The DRCS vocabulary represents these features using claims such as has-attribute, has-interface, has-value, etc.

![Figure 3.3 - 'DRCS Artifact Synthesis Structure'](#)

The rationale language (figure 3.4) embodies a set of question-spaces proposed by Lee (1991). It includes the following structures for expressing various aspects of rationale over Artifact Synthesis components (abstracted into the claim entity in the figure):-

- The Evaluation structure (figure 3.4A) links requirement specifications to the versions that achieve them over achieves and best-achieves associations. Both can have absolute or relative priorities.
- The Version structure (figure 3.4B) captures alternatives for a decision-problem, with the chosen option represented by an is-best-option-for claim. Versions have a status (abandoned, suspended, or conflict), and again can be assigned an absolute or relative priority.

---

1 Note requirement usually represents 'attribute-has-value-constraint' relationships from the Artifact Synthesis (see figure 3.3).
The Intent structure (figure 3.4C) relates a decision-problem to both the claim that raised the issue and a strategy to resolve it over raises-issue and has-strategy relations respectively. Like requirements, decision-problems can have absolute or relative priorities.

The Argumentation structure (figure 3.4D) records beliefs pro and contra to a claim. Questions can be raised about the validity of a claim and answers asserted in response, while has-result and has-input may be used to link claims to the procedures that derive them and to inputs to those procedures.

3.2.3.1 DRCS Influence

The main DRCS influence has been on development of a model to verify consistency across the MATrA Workspace (subsection 3.3.5) which we base on the Artifact Synthesis structure. However, extensions are necessary as it lacks elements to record functional architecture and behaviour. The literature provides a sound basis for developing these extensions; in particular, works by Pyle et al. (1993), Oliver (1994) and Wilson & McDermid (1995) as well as the SEDRES model discussed previously.

Klein's rationale language component highlights an important trade-off between expressive power and usability. Put simply, discussions with practitioners indicate that these structures are too complex for use on real projects. Therefore as part of this work, a single optimised structure composed of fewer elements and capable of use in conjunction with almost any MATrA element has been produced (Appendix B, Part Two).

With the above influences in mind, remaining sections of this chapter set out the fundamental elements of MATrA.
3.3 Fundamentals of MATrA

This subsection introduces the fundamental principles or ‘building-blocks’ of MATrA. Where appropriate, we highlight our scope of interest.

MATrA is an object-based approach to tracing artifacts for the development and assessment of aviation electronics (avionics) systems. It is based on a set of interconnected ‘traceability structures’ specified using the Class Diagram view of UML, with integrity constraints over these structures expressed in the Object-Constraint Language (OCL). To provide a flavour of tool support, a subset of elements for each structure are embedded in the ConceptBase object management system which implements the O-Telos modelling Language. Alternative representations and tool support are considered in Chapter Seven.

3.3.1 MATrA Fundamentals Overview

MATrA consists of five main principles. For reader orientation, these are introduced below with the details to follow in subsections 3.3.2 through 3.3.6:

1. A Workspace of notation dependent structures (meta-models) representing project data transferred from CASE tools (see 3.3.2).

2. A Meta-class model to maintain consistency of definition across Workspace meta-models by providing a common underlying representation (see 3.3.3).

3. A tool2matra (tool-to-MATrA) mapping function to provide data transfer from bespoke and commercial CASE tools to the Workspace (see 3.3.4).

4. The Product Data Synthesis (PDS), a notation independent structure representing fundamental elements of the emerging system (e.g., components, functions, behaviour, etc.), together with associations describing their composition; it maintains Workspace consistency by preventing ‘bad’ data from CASE tools entering via tool2matra (i.e., preventing data being mapped to a notation dependent structure) unless the PDS contains corresponding data elements, and by preventing violations once the data is ‘inside’ (see 3.3.5).

5. A Framework Model that gathers together MATrA elements to allow common behaviour to be managed more easily. In addition, means to create relations among notation dependent structures, and between these and the PDS are introduced (see 3.3.6).

Note, Workspace and PDS structures are intended for use by avionics engineers, rather than by method engineers who design the procedures that avionics engineers follow. Thus, in confining our scope to modelling and implementation of the structures themselves, we do so in the knowledge that graphical interfaces are necessary to ‘hide’ these definitions and so render MATrA practical. As evidence of the feasibility of creating these interfaces, we again highlight work within the department (Sukamaran, 1999) and also by the NATURE project which implemented graphical interfaces to various information structures (expressed in O-Telos) using a combination of C++ and the Andrew tool-kit (Pohl, 1996).

---

2 In this thesis, the term method engineer refers to persons who construct dedicated methods for engineering avionics systems. Avionics engineers then apply these methods to produce avionics systems.
3.3.2 Notation Dependent Traceability Structures (Meta-models)

As previously stated, practitioners use a range of notations to model avionics systems (often in conjunction with a process as part of a technique or methodology). These notations fall into two broad categories: those with a well-defined syntax and semantics (such as Circuit Diagrams and Ada) and those that are less rigorously defined but which offer flexibility as a result (for example Use Cases and Scenarios). Practitioners also require the ability to conduct safety assessment over these models using established techniques for hazard analysis (e.g., Fault Tree and Failure Modes and Effects Analyses). In addition, the scale and complexity of avionics projects normally dictates use of further notations and techniques to aid product management (for instance, Gantt Charts and the Programme Evaluation & Review Technique).

To maintain a tractable scope, we only feature a small subset of notations used throughout the industry drawn from the requirements, design, implementation, safety assessment and product management domains. Selection has been guided not only by their relevance to avionics engineering (determined from the literature, standards and discussions with practitioners), but also by a desire to demonstrate different representation formats (textual, graphical, tabular and program code) and hence tackle the respective modelling challenges presented by each. Featured notations are therefore as follows:-

- **Natural Language Structure** - means to represent informal artifacts (typically constituent parts of other notation dependent structures) at variable and user determined granularities (section 4.2);

- **User Centred Requirements Structure** - means to model functional requirements from a user perspective as Use Case Models, and to describe them using Scenarios and Message Sequence Charts (section 4.3);

- **Real-Time Network Specification Language (RTN-SL) Structure** - means to model architectural designs where concurrent processing components exchange information and synchronise through shared data in the connections (section 4.4);

- **SPARK Ada Structure** - means to model artifacts expressed using the SPARK Ada software implementation language (section 4.5);

- **Fault Tree Analysis (FTA) Structure** - means to model the results of FTA, a deductive safety analysis technique that identifies causes of some hazardous event and allows calculation of a probability of occurrence for that event (section 5.2);

- **Failure Mode and Effects Analysis (FMEA) Structure** - means to capture the results of FMEA, an inductive safety analysis technique that considers the failure of components of a system and tracks their effects to determine the ultimate consequences (section 5.3);

- **Programme Evaluation and Review Technique (PERT) Structure** - means to manage the planning and control of projects which represents relationships between the timing of events and activities as a network (section 5.4).
When developing these structures, we must first establish the concepts to be modelled. Note that most MATrA structures either represent domain specific "dialects" of less rigorously defined notations, or else rigorously defined notations whose arrangement has been enhanced to reflect domain usage. In both cases the models proposed and the concepts they contain are assumed to correspond to those underlying existing bespoke tools. For well-defined notations without domain enhancements, we make assumptions about the concepts and elements underlying hypothetical COTS tool.

Having identified the relevant concepts for a given notation, a meta-model is created using the Class Diagram view of UML. We then add OCL constraints defining well-formedness and PDS consistency; note with the former, we are guided by the literature for well-defined notations and by our own analysis of constituent elements for flexible notations. PDS checks verify integrity of the tool2matra transfer function by stating appropriate invariants for Workspace elements that must hold following its invocation. However, their main purpose is to determine whether consistency is preserved (between the Product Data Synthesis and the traceability Workspace) following changes to the PDS. Finally, we provide a flavour of tool support by encoding base class elements for each structure in the ConceptBase implementation of O-Telos. For reader orientation, subsection 3.3.7 has a worked example developed using the above approach which follows the format used with our featured meta-models in Chapters Four and Five.

3.3.3 MATrA Systems Engineering Notation Meta-class Model

The Systems Engineering Notation Meta-class model (SENM) provides a common basis for describing (and readily extending) notations and techniques supported within the MATrA Workspace. It does so through meta-level class definitions whose instances define the structures and elements of notations featured in 3.3.2. Given that we are defining meta-level classes - i.e., classes whose instances are classes - we use the UML «MetaClass» stereotype.

![Figure 3.5 - 'MATrA Systems Engineering Notation Meta-class Model']
The model itself (figure 3.5) consists of seven classes organised in a specialisation hierarchy; at its root is the artifact (Artifact) class, whose sole instance is the (abstract) AerospaceEngineeringObject class (see subsection 3.3.6.3.1) which serves to generalise all Workspace structures and elements for ease of manipulation.

Artifact is specialised by trace artifact (TraceArtifact), an abstract class containing the has_property attribute of type artifact property (ArtifactProperty). ArtifactProperty subsumes all predefined types as instances, e.g., String, Integer, Boolean, etc., together with user defined types such as Voltage. Note, we could simply state the latter as an instance of ArtifactProperty and specialisation of Real. However, it is often necessary to compose attribute definitions from other ArtifactProperty types. For example, Voltage could be stated in terms of a voltage attribute of type Real and a unit attribute of type String (with default “V”). Thus the actual definition of ArtifactProperty (see table 3.1) is as an instance of MetaClass with attribute described_by (stated using a reflexive composition association) of type ArtifactProperty.

TraceArtifact is specialised by the traceability structure (TraceabilityStructure) and structure element (StructureElement) classes. These provide the basis for representing all MATrA notation dependent structures such that each traceability structure (or sub-type) is defined as an aggregation of structure elements (over rolename has_element). In addition, TraceArtifact propagates its aggregation association (has_structure) with TraceabilityStructure to both subtypes (so for example decomposition of a DFD Function StructureElement may be represented using a separate model). StructureElement also bears two reflexive aggregation associations (has_part and has_transitive_part), as most of its instances are actually composed of other elements.

TraceabilityStructure is further specialised to differentiate between structures for development (DevelopmentStructure), assessment (AssessmentStructure) and product management (ProductManagementStructure). The ability to distinguish these different ‘viewpoints’ is convenient for manipulation purposes, in particular for constraining the source and target of associations between traceability structures and elements.

O-Telos implementation of the SENM base classes from figure 3.5 is as follows:-

```plaintext
Artifact in MetaClass end
ArtifactsProperty in MetaClass with
attribute
described_by : ArtifactProperty
end
TraceArtifact in MetaClass isa Artifact
with attribute
has_property : ArtifactProperty;
has_structure : TraceabilityStructure
constraint
abstract_TA: forall t/Token s/SimpleClass (t in s) => not (t in TraceArtifact)
end
StructureElement in MetaClass isa
TraceArtifact with attribute
has_part : StructureElement;
has_transitive_part : StructureElement
end
TraceabilityStructure in MetaClass isa
TraceArtifact with attribute
has_element : StructureElement
end
DevelopmentStructure in MetaClass isa
TraceabilityStructure
end
AssessmentStructure in MetaClass isa
TraceabilityStructure
end
ProductManagementStructure in MetaClass isa
TraceabilityStructure
end
```
Notice O-Telos’ uniform treatment of aggregation relationships and attributes which are both represented using the `with attribute` convention. It is also worth commenting on the information loss from UML to O-Telos, in particular as regards multiplicity. This can however be stated by specifying appropriate constraints (not shown) as described in 2.2.2.2.8.

Table 3.1 clarifies the relationship between UML and O-Telos modelling levels and conventions. It does so by demonstrating specification and instantiation of `ConnectorPin` and `Connection (StructureElement)` types for a Circuit Diagram meta-model, to be featured in subsection 3.3.7.

<table>
<thead>
<tr>
<th>UML</th>
<th>O-Telos</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>«MetaClass» Level</strong></td>
<td><strong>Meta Class Level</strong></td>
</tr>
<tr>
<td><code>&lt;&lt;MetaClass&gt;&gt; StructureElement</code></td>
<td><code>StructureElement in MetaClass with attribute</code></td>
</tr>
<tr>
<td><code>has_part</code></td>
<td><code>has_property : ArtifactProperty</code></td>
</tr>
<tr>
<td><code>where has_part : StructureElement</code></td>
<td><code>has_part : StructureElement</code></td>
</tr>
<tr>
<td><code>described_by</code></td>
<td><code>ArtifactProperty in MetaClass with attribute</code></td>
</tr>
<tr>
<td><code>where ArtifactProperty</code></td>
<td><code>described_by : ArtifactProperty</code></td>
</tr>
</tbody>
</table>

**Class Level**

| `<<Structure Element> Connection` | `Connection in StructureElement, SimpleClass with` |
| `pin` | `has_part : ConnectorPin end` |
| `<<Structure Element> ConnectorPin` | `ConnectorPin in StructureElement, SimpleClass with` |
| `v : Voltage` | `has_property : Voltage` |
| `<<ArtifactProperty> Voltage` | `Voltage : Voltage` |
| `voltage : Real` | `voltage : Real` |
| `unit : String = "V"` | `unit : String = "V"` |
| `<<ArtifactProperty> String` | `String : String` |
| `<<ArtifactProperty> Real` | `Real : Real` |

**Object Level**

| Not Considered in this Thesis - All Instantiations in O-Telos | **Token Level** |
| ExampleConnection in Connection, Token with | `ExampleConnection in Connection, Token with` |
| pin | `pin : ExamplePin1 end` |
| ExamplePin1 in ConnectorPin, Token with | `ExamplePin1 in ConnectorPin, Token with` |
| `v` | `v : Pin1Voltage` |
| Pin1Voltage in Voltage, Token with | `Pin1Voltage in Voltage, Token with` |
| voltage | `voltage : 100` |

Table 3.1 - ‘Comparison of MATrA Modelling Conventions : UML and O-Telos’
3.3.4 *`tool2matra’* Mapping Function

The difficulty faced by aerospace practitioners in establishing traceability linkages between data stored across disparate CASE tools was indicated at the outset (subsection 1.3); this point is emphasised by figure 3.6. One solution is to transfer data from the tools to a Workspace of notation dependent structures (introduced in 3.3.2) capable of receiving this data. By expressing these structures in a uniform format (i.e., in a common language), links (expressed in the same language as the structures) can be inserted that capture dependencies among data in CASE tools, within the Workspace.

The potential volume of data generated by aerospace projects means that all mappings across the CASE tool/MATrA interface should (as far as possible) be achieved with limited human intervention. In this research we treat the mapping process as a ‘black-box’, such that all transfers are considered in terms of an undefined function, *tool2matra*, that maps data from the internal models of CASE tools onto notation dependent structures as figure 3.7 illustrates.
The function takes as its input parameters, a populated CASE tool data structure (pCDS) with a corresponding unpopulated notation dependent structure (uNDS) and the Product Data Synthesis (PDS), and returns a populated notation dependent structure (pNDS). The interface to this function may be defined as follows:

\[
pNDS \text{ } tool2Matra \text{ (pCDS, uNDS, PDS)}
\]

The resulting Workspace provides an integrated environment allowing traceability linkages to be established between otherwise disjoint data (as shown in figure 3.7); by ‘dropping’ from CASE tool level into the Workspace, engineers can move around the complete data set wherever traceability links exist - but especially between notations. For instance, from a use case in a Use Case Model tool, to an Activity in RTN-SL, to a SPARK Ada package specification - a combination unlikely to be supported at tool level.

It should be stressed that where a ‘clean’ mapping of elements to the Workspace is not achieved, tool2Matra will create an error-log. The reasons for failure are likely to be attributable to bad data detected by PDS cross-checks and due to:- i) misnomer errors on the part of developers when referring to legitimate elements of the target system (caused for example by typos); ii) developers referring to elements of the target system that do not and should not exist; or iii) developers referring to legitimate system elements that have not as yet been added to the Product Data Synthesis. We consider the PDS in more depth in the following subsection.

### 3.3.5 Product Data Synthesis (PDS)

As subsection 3.3.4 has demonstrated, the Product Data Synthesis plays an important role in MATrA by preventing bad data from entering the Workspace during tool2matra mapping. However, it is also required to maintain consistency once data are inside the Workspace, notably following updates to the PDS itself. This is ensured by rules that maintain the integrity of links associating elements of the PDS with corresponding Workspace elements. The links are introduced in subsection 3.3.6.4.4, while constraints on their instances are specified in Chapters Four and Five.

Product Data Synthesis is a notation independent structure populated by engineers with appropriate design authority (see also 7.4.6). It represents fundamental elements of the emerging system and associations between these elements. By fundamental elements we mean generic types for modelling architecture and behaviour, while the associations enable meaningful combination of these elements.

It can be see from figure 3.8 that core PDS constituents are referred to as build elements (BuildElement) and build associations (BuildAssociation), both of which are abstract. Each build association subtype has single source and target build elements. Correct usage (i.e., legitimate combination) of the various element and association subtypes is maintained through appropriate constraints (see 3.3.5.2).

---

3 Representing Build Associations as classes allows rationale to be expressed over the claims made by these associations using the structure in Appendix B (Part 2).
3.3.5.1 Build Elements

BuildElement is specialised by classes representing modules (Module), interfaces (Interface), functions, (Function), transactions (Transaction), conditions (Condition), input/output (InputOutput), properties

Figure 3.8 - 'Product Data Synthesis Elements'
MA TrA: Foundations and Fundamentals

(Property) and specifications (Specification). Module (with attribute module_name of type String) describes a system or 'component' of a system - be it hardware, software or human; modules or functions may be connected via an Interface (with attribute interface_name of type String). The functional architecture of a module is described using the Function element (with attribute function_name of type String); a Transaction (with attribute transaction_name of type String) is a combination of functions that perform some task or unit of work. Flows (energy, material or signals) within and between modules are described using the InputOutput element (with attribute flow_name of type String). Meanwhile, Condition (with attribute condition_label of type String) expresses some “state of affairs” and is a generalisation of state and event⁴.

All of the above can be described by the Property element (with attribute property_name of type String), which in turn has a Specification (with attribute value_specification of type String).

3.3.5.2 Build Associations

In the interests of readability, we present build associations permitted between the above build elements in the form of a table (table 3.2).

<table>
<thead>
<tr>
<th>BuildAssociation</th>
<th>Source</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasProperty</td>
<td>Module</td>
<td>Property</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td>Property</td>
</tr>
<tr>
<td></td>
<td>Interface</td>
<td>Property</td>
</tr>
<tr>
<td></td>
<td>Transaction</td>
<td>Property</td>
</tr>
<tr>
<td></td>
<td>Condition</td>
<td>Property</td>
</tr>
<tr>
<td></td>
<td>InputOutput</td>
<td>Property</td>
</tr>
<tr>
<td>Encapsulates</td>
<td>Module</td>
<td>Function</td>
</tr>
<tr>
<td>BehavesAccordingTo</td>
<td>Module</td>
<td>Transaction</td>
</tr>
<tr>
<td>ProducesExternalIO</td>
<td>Function</td>
<td>InputOutput</td>
</tr>
<tr>
<td></td>
<td>Module</td>
<td>InputOutput</td>
</tr>
<tr>
<td>ConsumesExternalIO</td>
<td>Function</td>
<td>InputOutput</td>
</tr>
<tr>
<td></td>
<td>Module</td>
<td>InputOutput</td>
</tr>
<tr>
<td>ConsumesInternalIO</td>
<td>Function</td>
<td>InputOutput</td>
</tr>
<tr>
<td></td>
<td>Module</td>
<td>InputOutput</td>
</tr>
<tr>
<td>ProducesInternalIO</td>
<td>Function</td>
<td>InputOutput</td>
</tr>
<tr>
<td>CarriesCondition</td>
<td>InputOutput</td>
<td>Condition</td>
</tr>
<tr>
<td>SentTo</td>
<td>InputOutput</td>
<td>Interface</td>
</tr>
<tr>
<td>ReceivedFrom</td>
<td>InputOutput</td>
<td>Interface</td>
</tr>
<tr>
<td>LeadsTo</td>
<td>Condition</td>
<td>Condition</td>
</tr>
<tr>
<td>OccurringIn</td>
<td>Condition</td>
<td>Condition</td>
</tr>
<tr>
<td>UsesFunction</td>
<td>Transaction</td>
<td>Function</td>
</tr>
<tr>
<td>InterfacesTo</td>
<td>Module</td>
<td>Module</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td>Function</td>
</tr>
</tbody>
</table>

⁴ The notion of Condition as an abstraction of state and event is taken from Wilson & McDermid (1995)
Table 3.2 - ‘Product Data Synthesis Build Associations’

Note: we are not claiming this set of build elements and associations is exhaustive, but rather sufficient to demonstrate verification of Workspace elements for the notations featured in Chapters Four and Five.

Most build associations in table 3.2 are self explanatory and based on existing systems engineering literature. HasProperty for example corresponds to the ‘has-attribute’ association in Klein’s work (figure 3.3), but with additional source types to accommodate extensions for behavioural entities; the same is true of HasInterface (whose source types reflect the ability to capture functional as well as physical architectures). Similarly, HasSpecification corresponds to ‘has-value’, while the notion of module decomposition using ‘has-submodule’ (HasSubmodule) is extended to fault-tolerant architectures (HasFTSubmodule\(^5\)), functions (HasSubfunction) and conditions (HasSubcondition).

The remaining associations capture behaviour and are mainly derived from work by Oliver (1994). In particular Encapsulates relates modules to functions, BehavesAccordingTo, modules to transactions and UsesFunction transactions to functions. HasCondition meanwhile captures states and events of module, function, interface and transaction entities. Events provide stimuli that are carried in input-output flows (represented by CarriesCondition\(^6\)) and which trigger state changes, denoted using OccurringIn and LeadsTo associations.

Flows within and between systems are recorded through ProducesInternalI/O/ConsumesInternalI/O and ProducesExternalI/O/ConsumesExternalI/O\(^7\) respectively; these may be exchanged via interfaces using SentTo and ReceivedFrom. Where two interfaces join to one another, the Connection class is used; if the connection itself is an entity and needs to be modelled as such, then it too is classed as an interface. If

---

\(^5\) This BuildAssociation is based on work by Pearson et al. (1998).
\(^6\) This is also compatible with Pyle et al. (1993).
\(^7\) The fact that modules can also consume and produce InputOutput enables external entities to be treated as black boxes.
no interface is explicitly stated between two interacting modules or two functions, an InterfacesTo association is employed.

Constraints restricting the source and target of these associations follow a similar pattern. This can be parameterised in the Object Constraint Language as follows:

\[
\text{BuildAssociation Subtype t} \invariant \\
\\text{self.allInstances->forall(t | (t.source.oclType = \{Type S\} or t.source.oclType ...) and (t.target.oclType = \{Type T\} or t.target.oclType ...))}
\]

For instance, the following invariant instantiates these parameters by constraining source and target of Encapsulates associations to Module and Function types respectively:

\[
\text{Encapsulates invariant} \\
\\text{self.allInstances->forall(e | e.source.oclType = Module and e.target.oclType = Function)}
\]

Note: additional rules could be stated over build associations to prevent, for instance, circular dependencies among build elements (e.g., modules containing themselves as sub-modules).

### 3.3.6 MATrA Framework Model

The Framework Model (figure 3.9) assembles core MATrA elements, many of which are abstract, allowing common behaviour to be managed more easily. Means to create relations between notation dependent structures and to link these to the Product Data Synthesis are also introduced. Elements of the model are described in the subsections that follow.

![MATrA Framework Model](image)

**Figure 3.9 - ‘MATrA Framework Model Elements’**

O-Telos implementation of the base classes in figure 3.9 (along with their associations) appears below:

---

*Note the () generalisations are non-standard OCL.*
AerospaceTraceabilityEntity in SimpleClass with
constraint
abstract_ATE: $ \forall t/Token, s/SimpleClass\n(t \text{ in } s) \implies \neg (t \text{ in } \text{AerospaceTraceabilityEntity})$
end

AerospaceBuildEntity in SimpleClass isA AerospaceTraceabilityEntity with
constraint
abstract_ABE: $ \forall t/Token, s/SimpleClass\n(t \text{ in } s) \implies \neg (t \text{ in } \text{AerospaceBuildEntity})$
end

BuildElement in SimpleClass isA AerospaceBuildEntity with
constraint
abstract_BE: $ \forall t/Token, s/SimpleClass\n(t \text{ in } s) \implies \neg (t \text{ in } \text{BuildElement})$
end

BuildAssociation in SimpleClass isA AerospaceBuildEntity with
attribute
source: BuildElement;
target: BuildElement
constraint
abstract_BA: $ \forall t/Token, s/SimpleClass\n(t \text{ in } s) \implies \neg (t \text{ in } \text{BuildAssociation})$
end

AerospaceEngineeringEntity in SimpleClass isA AerospaceTraceabilityEntity with
constraint
abstract_AEE: $ \forall t/Token, s/SimpleClass\n(t \text{ in } s) \implies \neg (t \text{ in } \text{AerospaceEngineeringEntity})$
end

AerospaceEngineeringObject in Artifact, SimpleClass isA AerospaceEngineeringEntity
with
constraint
abstract_AEO: $ \forall t/Token, s/SimpleClass\n(t \text{ in } s) \implies \neg (t \text{ in } \text{AerospaceEngineeringObject})$
end

AerospaceEngineeringAssociation in SimpleClass isA AerospaceEngineeringEntity with
attribute
from_entity:

AerospaceEngineeringEntity; to_entity:
AerospaceEngineeringEntity
constraint
abstract_AEA: $ \forall t/Token, s/SimpleClass\n(t \text{ in } s) \implies \neg (t \text{ in } \text{AerospaceEngineeringAssociation})$
end

AerospaceManagementEntity in SimpleClass isA AerospaceTraceabilityEntity with
constraint
abstract_AME: $ \forall t/Token, s/SimpleClass\n(t \text{ in } s) \implies \neg (t \text{ in } \text{AerospaceManagementEntity})$
end

AerospaceEngineeringProject in SimpleClass isA AerospaceManagementEntity with
attribute
project_title: String;
has_pds: ProductDataSynthesis;
has_workspace: TraceabilityWorkspace;
has_link_entity: AerospaceLinkEntity
end

ProductDataSynthesis in SimpleClass isA AerospaceManagementEntity with
attribute
build_entity: AerospaceBuildEntity
end

TraceabilityWorkspace in SimpleClass isA AerospaceManagementEntity with
attribute
engineering_entity: AerospaceEngineeringEntity
end

AerospaceLinkEntity in SimpleClass isA AerospaceManagementEntity
constraint
abstract_ALE: $ \forall t/Token, s/SimpleClass\n(t \text{ in } s) \implies \neg (t \text{ in } \text{AerospaceLinkEntity})$
end

BEmodelAEO in SimpleClass isA AerospaceLinkEntity
BEelementAEO in SimpleClass isA AerospaceLinkEntity

3.3.5.1 **Aerospace Traceability Entity**
The root of the Framework is termed aerospace traceability entity (AerospaceTraceabilityEntity) or ATE, an abstract class that generalises every constituent MATrA element. Its presence is motivated by the desire to allow argumentation to be expressed over any of these constituents. This is achieved by placing ATE at the hub of our Argumentation Structure (Appendix B) which optimises the DRCS rationale components from figure 3.4. AerospaceTraceabilityEntity is specialised by aerospace build entity (AerospaceBuildEntity), aerospace engineering entity (AerospaceEngineeringEntity) and aerospace management entity (AerospaceManagementEntity). These are described below.
3.3.6.2 Aerospace Build Entity
AerospaceBuildEntity (ABE) is an abstract class that generalises BuildElement and BuildAssociation from 3.3.5. We use it in modelling the Product Data Synthesis (see subsection 3.3.6.4.2).

3.3.6.3 Aerospace Engineering Entity
AerospaceEngineeringEntity (AEE) is again abstract and is used to model the traceability Workspace (subsection 3.3.6.4.3). It has two subtypes, aerospace engineering object (AerospaceEngineeringObject) and aerospace engineering association (AerospaceEngineeringAssociation) which are described below.

3.3.6.3.1 Aerospace Engineering Object
AerospaceEngineeringObject or AEO (also abstract) instantiates the Artifact meta-class (from 3.3.3). Its subtypes subsume all notation dependent structures and their constituent elements. This is evident from the worked example (3.3.7) and the structures featured in Chapters Four and Five.

3.3.6.3.2 Aerospace Engineering Association
AerospaceEngineeringAssociation (figure 3.10) or AEA is again an abstract class whose subtypes realise traceability between two AEEs (with rolenames from_entity and to_entity), or more specifically, subtypes of the two specialisations of AEE which as indicated in 3.3.6.3 are AerospaceEngineeringObject and AerospaceEngineeringAssociation itself. AEAs can link two AEOs - at either element or model level - or alternatively, an AEO and another AEA (though not two AEAs9); the need to connect two associations was highlighted by our featured example in 2.2.2.2.9. As with build associations, correct usage is maintained through constraints (not shown) expressed over AerospaceEngineeringAssociation subtypes.

Figure 3.10 - 'Aerospace Engineering Association (and example subtypes)'

---

9 This is preserved by a constraint - not shown.
While this thesis concentrates on developing meta-models for notation dependent structures, i.e. the from_entity and to_entity (AEO type) ends of AerospaceEngineeringAssociation subtypes, the case studies in Chapter Six do include instances of the examples shown in figure 3.10. We therefore state the O-Telos base classes for these associations to help reader understanding:

**Base Classes for AEAs Used in Case Study in Subsection 6.2**

PathThrough in SimpleClass isA AerospaceEngineeringAssociation end
IsARoleOf in SimpleClass isA AerospaceEngineeringAssociation end
IllustratedBy in SimpleClass isA AerospaceEngineeringAssociation end
HasCommonInstance in SimpleClass isA AerospaceEngineeringAssociation end

**Base Classes for AEAs Used in Case Study in Subsection 6.3**

ImplementedBy in SimpleClass isA AerospaceEngineeringAssociation end
AssessedBy in SimpleClass isA AerospaceEngineeringAssociation end
Effects in SimpleClass isA AerospaceEngineeringAssociation end
ContributesTo in SimpleClass isA AerospaceEngineeringAssociation end

3.3.6.4 Aerospace Management Entity

AerospaceManagementEntity (AME) is an abstract class, subsuming specialisations for key MATra elements, namely aerospace engineering project (AerospaceEngineeringProject), Product Data Synthesis (ProductDataSynthesis) and traceability Workspace (TraceabilityWorkspace). In addition, AME includes a further specialisation providing means of linking Workspace and PDS elements through the notion of aerospace link entities (AerospaceLinkEntity). Each of these concepts is described below.

3.3.6.4.1 Aerospace Engineering Project

AerospaceEngineeringProject (AEP) is simply a 'container' for all artifacts relating to a particular project and to that extent, is analogous to a traceability cube (as introduced in subsection 1.4.5). AEP is modelled as an aggregation of a ProductDataSynthesis and a TraceabilityWorkspace, together with linkages between the two described by means of AerospaceLinkEntity subtypes, as indicated in figure 3.11.

![Figure 3.11 - 'Aerospace Engineering Project']

3.3.6.4.2 Product Data Synthesis

ProductDataSynthesis is defined as an aggregation of AerospaceBuildEntity types, as described in subsection 3.3.6.2 (see figure 3.12).

![Figure 3.12 - 'Product Data Synthesis']
3.3.6.4.3 Traceability Workspace

Similarly, TraceabilityWorkspace is defined as an aggregation of AerospaceEngineeringEntity types, as described in subsection 3.3.6.3 (see figure 3.13).

![Traceability Workspace Diagram](image)

Figure 3.13 - 'Traceability Workspace'

3.3.6.4.4 Aerospace Link Entity

AerospaceLinkEntity (ALE) is an abstract class whose subtypes provide means by which to navigate between the PDS and Workspace. From the Framework Model (figure 3.9), it can be seen that the subtypes in question are BEmodelAEO and BEelementAEO. These represent associations between build elements and meta-models, and between build elements and meta-model elements respectively. Conceptually, this distinction is best illustrated by an example.

In figure 3.14 (A), BuildElement '(BE)X' is linked via a BEmodelAEO association to a Workspace meta-model for which it is the subject\(^{10}\). In figure 3.14(B), BuildElement '(BE)Y' is linked over a similar association to another Workspace meta-model in which (BE)X is named as a model element - namely '(ME)X'; (BE)X and (ME)X are therefore related via a BEelementAEO association. Moreover, it can be seen that in the PDS, build element (BE)Y is linked to (BE)X over an (unspecified) BuildAssociation, such as HasSubmodule or HasFunction.

\(^{10}\) It will be seen from Chapters Four and Five that most of the meta-models featured have a subject-module (or similarly named) attribute identifying them with a PDS Module.
Again we choose to simplify the model through a common definition (figure 3.15), with constraints (below) restricting instantiation of the aerospace_engineering_object end to type TraceabilityStructure (or a subtype thereof) for BEmodelAEO associations, and to type StructureElement for BEelementAEO associations.

![Figure 3.15 - 'Aerospace Link Entity']

**BEmodelAEO invariant**

```plaintext
self.allInstances->forall(b \ b.aerospace_engineering_object.oclType.oclIsKindOf(TraceabilityStructure))
```

**BEelementAEO invariant**

```plaintext
self.allInstances->forall(b \ b.aerospace_engineering_object.oclType.oclType = StructureElement)
```

### 3.3.7 Circuit Diagrams: A Meta-model Worked Example

As previously indicated, this thesis concentrates on the notation dependent structure (i.e., meta-modelling) aspects of MATrA introduced in 3.3.2. Therefore to aid reader orientation, we present a worked example based on Circuit Diagrams which follows the approach adopted for our featured notations in Chapters Four and Five. Specifically, we discuss motivation for their inclusion, along with key concepts, a meta-model expressed using the Class Diagram view of UML and appropriate constraints stated in OCL. To provide a flavour of tool support, we embed and instantiate base classes for the model in the ConceptBase implementation of O-Telos.

#### 3.3.7.1 Motivation

Electronics (i.e., avionics) have been used in aircraft command and control systems for over half a century. Their physical designs are expressed as Circuit Diagrams and whilst not considered here in depth, it is nevertheless important to show the feasibility of representing them within MATrA. In doing so, we draw reader attention to Modelica - an object-oriented language for the modelling and simulation of physical systems that has influenced this aspect of our work (Mattsson & Elmquist, 1998). Readers should also be aware that this section provides background to the case study in Chapter Six (subsection 6.3) which includes a Failure Modes and Effects Analysis of Circuit Diagram components for part of an aircraft braking system.

#### 3.3.7.2 Concepts

Complex electrical circuits may contain a wide range of components. However, our interest is restricted to a small subset necessary to demonstrate the basic wiring schematic in 3.3.7.4, specifically voltage...
source, resistor, inductor and capacitor elements, together with a ground point. These are connected using a standard pin type interface; for the purpose of this example, we assume each of the above types has two pins (as shown in figure 3.16), p and n, denoting positive and negative - the exception being ground-point, which has a single positive pin. The connection of two pins along a wire forms a node.

![Figure 3.16 - 'Generic Two Pin Electrical Component']

Two basic properties, voltage (v) and current (i) are necessary to define interactions among components connected via wires. These in turn allow calculation of further properties (e.g., the potential difference or 'voltage drop' between the value of v on entering a component at p and leaving at n), along with well-formedness constraints expressing first principles (e.g., Kirchhoff's Current Law requires that the current flowing into a node equals the current flowing out). Therefore besides modelling the basic components, we will also state a number of key invariants.

### 3.3.7.3 Circuit Diagram Structure (Meta-model)

This section introduces a UML model representing Circuit Diagrams (3.3.7.3.1), together with OCL constraints over elements of the model (3.3.7.3.2) and O-Telos implementation of its base classes (3.3.7.3.3).

#### 3.3.7.3.1 Specification of Circuit Diagram Meta-model in UML

As figure 3.17 illustrates, Circuit Diagram (CircuitDiagram) instantiates the DevelopmentStructure meta-class and is defined as an aggregation of classes representing the circuit elements described above.

We begin our description of these elements by mentioning the basic variables needed to model interaction via a wire - current (Current) and voltage (Voltage) - definitions of which instantiate the ArtifactProperty meta-class. These properties are used in defining connector pin (ConnectorPin), an instantiation of StructureElement which provides our standard interface class (verified against the PDS Interface type for tool2matra transfer) with attributes 'i' and V of type Current and Voltage respectively (see figure 3.17).

In turn, ConnectorPin is used towards definition of standard electronic components. For instance, the notion of elements with two pins is captured by the abstract TwoPin superclass (figure 3.18) whose rolenames p and n differentiate positive and negative pins. TwoPin is again defined as an instance of StructureElement with attributes 'pd' - representing potential difference - and 'i' - denoting current into pin

---

11 We do not explain the detail of these circuit elements here, nor the rules constraining their behaviour; instead, readers are referred to Cogdell (1999).
12 UML definitions of these classes are omitted from our example, although their O-Telos equivalents are included. Readers can also refer back to Table 3.1 for an indication of how ArtifactProperty types are represented.
13 In this worked example, we omit the relationship between circuit diagram elements and PDS.
Specialisations of TwoPin may now be defined for specific component types (which map onto Module elements of the PDS) - see figure 3.18. For example, a simple resistor (Resistor) adds an attribute for resistance (r) of type resistance (Resistance). In turn this can be used to express Ohm's law which states that \( v = i \times r \) (again, see 3.3.7.2.3). Further classes representing AC power source (ACsource), capacitors (Capacitor) and inductors (Inductor) are similarly defined, together with the single pin ground point (Ground) element (all of which instantiate the StructureElement meta-class).

Finally, it is necessary to consider connection of pairs of circuit elements via pins to form nodes. This is
achieved using a Connection class (which also corresponds to Interface in the PDS). Again, invariants are defined in the following subsection to ensure the resulting circuits are well-formed in terms of connecting components with (for instance) compatible voltages.

3.3.7.3.2 OCL Constraints over Circuit Diagram Meta-model

As indicated above, a number of restrictions over elements of the CircuitDiagram meta-model apply. These are expressed as follows:

Constraints over TwoPin type:

- Constraint ensuring correct potential difference across a component (i.e., pd equals positive pin voltage minus negative pin voltage).

TwoPin invariant
self.allInstances->forall(t | t.p.d.voltage = t.p.v.voltage - t.n.v.voltage)

- Constraint ensuring that current flowing into a component at pin p flows out again at pin n (i.e., there is no current loss).

TwoPin invariant
self.allInstances->forall(t | t.p.i.current = t.n.i.current)

- Constraint over Resistor ensuring conformance to Ohm's Law (i.e., Potential difference = Current x Resistance):

Resistor invariant
self.allInstances->forall(r | r.p.d.voltage = r.r.resistance * r.i.current)

Constraints over Connection type:

- Constraint to ensure connection of pins with compatible voltages.

Connection invariant
self.allInstances->forall(c | self.pin->forall(p1, p2 | not (c.pin->includes(p1) and c.pin->includes(p2) and p1.v.voltage <> p2.v.voltage)))

- Constraint expressing Kirchhoff's Current Law (i.e., current into a node equals the current out).

Connection invariant
self.allInstances->forall(c | self.pin->forall(p1, p2 | not (c.pin->includes(p1) and c.pin->includes(p2) and p1.i.current <> p2.i.current)))

Note, as the complexity of the circuits increases, further and more complex rules and constraints may be defined - for example to identify sneak patterns for use in sneak circuit analysis14.

14 Circuit sneaks are caused when current which is expected to flow along a particular path of wires unexpectedly flows in another direction causing systems to malfunction or sub-systems to activate unexpectedly.
3.3.7.3.3 O-Telos Implementation of Circuit Diagram Base Classes

The following O-Telos code implements base class elements for the Circuit Diagram meta-model.

**Definition of Electrical Measures**

Voltage in ArtifactProperty, SimpleClass with described_by
- voltage : Real;
- unit : String = "V"
end

Resistance in ArtifactProperty, SimpleClass with described_by
- resistance : Real;
- unit : String = "Ohm"
end

Current in ArtifactProperty, SimpleClass with described_by
- current : Real;
- unit : String = "A"
end

Capacitance in ArtifactProperty, SimpleClass with described_by
- capacitance : Real;
- unit : String = "F"
end

Inductance in ArtifactProperty, SimpleClass with described_by
- inductance : Real;
- unit : String = "L"
end

Frequency in ArtifactProperty, SimpleClass with described_by
- frequency : Real;
- unit : String = "Hz"
end

**Definition of Circuit Primitives**

ConnectorPin in StructureElement, SimpleClass isA AerospaceEngineeringObject with
- v : Voltage;
- i : Current
end

Connection in StructureElement, SimpleClass isA AerospaceEngineeringObject with
- pin : ConnectorPin
end

TwoPin in StructureElement, SimpleClass isA AerospaceEngineeringObject with
- has_property
  - pd : Voltage;
  - i : Current
- has_part
  - p : ConnectorPin;
  - n : ConnectorPin
constraint
  abstract_TwoPin: $forall t/Token,
  s/SimpleClass
  (t in s) == not (t in TwoPin)$
end

ACsource in StructureElement, SimpleClass
isA TwoPin with has_property
- f : Frequency;
- va : Voltage
end

Resistor in StructureElement, SimpleClass
isA TwoPin with has_property
- r : Resistance
end

Capacitor in StructureElement, SimpleClass
isA TwoPin with has_property
- c : Capacitance
end

Inductor in StructureElement, SimpleClass
isA TwoPin with has_property
- l : Inductance
end

Ground in StructureElement, SimpleClass
isA AerospaceEngineeringObject with
- p : ConnectorPin
end

**Definition of Circuit Diagram**

CircuitDiagram in DevelopmentStructure, SimpleClass isA AerospaceEngineeringObject with
- has_property
  - circuit_name : String
- has_element
  - ac_source : ACsource;
  - resistor : Resistor;
  - capacitor : Capacitor;
  - inductor : Inductor;
  - ground : Ground;
  - connection : Connection;
  - connector_pin : ConnectorPin
end

3.3.7.4 Circuit Diagram Example

We now present an example that instantiates the above model using a simple electrical circuit shown in figure 3.19. The circuit, named "Simple Circuit", comprises six connected components - a voltage source (AC), two resistors (R1 and R2), an inductor (L), a capacitor (C) and a ground point (G).

Note: readers are reminded that in operational circumstances, the O-Telos objects that follow will be created automatically by the tool2matra function and 'concealed' from users by an appropriate interface.
Figure 3.19 - ‘Simple Electrical Circuit Diagram (source: Mattsson & Elmqvist, 1998)’

The (partial) O-Telos representation of this Circuit Diagram and its featured components can be stated as follows:-

**Instantiation of Electrical Measures**

- **ACVoltage in Voltage, Token with voltage vlt : 220**
  - end
- **R1Resistance in Resistance, Token with resistance rst : 10**
  - end
- **R2Resistance in Resistance, Token with resistance rst : 100**
  - end
- **Ccapacitance in Capacitance, Token with capacitance cap : 0.01**
  - end
- **Linductance in Inductance, Token with inductance ind : 0.1**
  - end

**Instantiation of Circuit Primitives**

- **ACp in ConnectorPin, Token end**
- **ACn in ConnectorPin, Token end**
- **R1p in ConnectorPin, Token end**
- **R1n in ConnectorPin, Token end**
- **R2p in ConnectorPin, Token end**
- **R2n in ConnectorPin, Token end**
- **Cp in ConnectorPin, Token end**
- **Cn in ConnectorPin, Token end**
- **Lp in ConnectorPin, Token end**
- **Ln in ConnectorPin, Token end**
- **Gp in ConnectorPin, Token end**
- **Cn1 in Connection, Token with pin p1 : ACp; p2 : R1p**
  - end
- **Cn2 in Connection, Token with pin p1 : R1n; p2 : Cp**
  - end
- **Cn3 in Connection, Token with pin p1 : Cn; p2 : ACn**
  - end
- **Cn4 in Connection, Token with pin p1 : R1p; p2 : R2p**
  - end
- **Cn5 in Connection, Token with pin p1 : R2n; p2 : Lp**
  - end
- **Cn6 in Connection, Token with pin p1 : Ln; p2 : Cn**
  - end
- **Cn7 in Connection, Token with pin p1 : ACn; p2 : Gp**
  - end
- **AC in ACsource, Token with va voltAmplitude : ACvoltage p acp : ACp n acn : ACn**
  - end
- **R1 in Resistor, Token with r rResistance : R1resistance p rp : R1p n rn : R1n**
  - end
- **R2 in Resistor, Token with r rResistance : R2resistance p rp : R2p n rn : R2n**
  - end
- **C in Capacitor, Token with c cCapacitance : Ccapacitance p cp : Cp n cn : Cn**
  - end
3.3.7.5 Circuit Diagram Example Summary

The above example provides an overview of our approach to representing (in this case graphical) notations used in avionics development. Specifically, it sets out a UML meta-model for Circuit Diagrams, together with some basic OCL constraints and O-Telos implementation of base classes. Given a common representation format such as this, associations (AEAs) can be established to support traceability between various different notations, including those featured in Chapters Four and Five; we demonstrate this aspect in Chapter Six.

3.4 Chapter Summary

Chapter Three described how existing work in the public domain has influenced the development of MATrA. We then provided an overview of key MATrA concepts, including means to represent notations used by aerospace practitioners in a format amenable to traceability (demonstrated by example). This was highlighted as being the main focus of our work and will now be explored in depth through Chapters Four and Five.
Chapter 4  Structuring Development Artifacts

4.1 Introduction
This chapter introduces meta-models (traceability structures) representing well-defined and flexible development notations used by avionics engineers that provide input to, and are supported by, the MATrA Workspace. In each case, we present factors motivating the inclusion of a particular notation and a précis of its main concepts. We also introduce a meta-model expressed in UML capturing the main syntactic elements, as well as population and well-formedness constraints stated in OCL and O-Telos implementation of the UML base classes. Where a meta-model is not featured as part of a case study within Chapter Six, then smaller worked examples are provided to demonstrate key aspects. We further relate the models to the traceability dimensions (from Chapter One) as appropriate.

4.2 MATrA Natural Language Structure

4.2.1 Introduction
This section proposes a utility structure allowing the capture and traceability of artifacts expressed using natural language. Its main function in this thesis is to support the design of other structures, the idea being to replace a standard text String type with an MNLS wherever fine-grained traceability is required.

4.2.2 Motivation
Despite a varied range of graphical abstractions and the continued 'push' from academia towards use of formal methods, artifacts expressed in natural language are still commonplace in aerospace systems engineering (especially in the civil market which is not bound by standards such as DefStan 00-55 [MoD, 1997] which calls for a formal mathematical specification¹). To render these documents traceable (and allow relationships with other representations), MATrA includes the notion of a Natural Language Structure (MNLS). Though an obvious application is the representation of requirements statements (an issue to which we return in our future work section in 7.4.5), the basic MNLS described here is intended as a utility structure. This utility stems from two key features: i) artifacts can be represented at varying granularities; and ii) the granularity levels are user-determined.

4.2.3 Tracing Textual Artifacts: A Natural Language Structure

4.2.3.1 Concepts
Our approach to the representation of natural language artifacts has been influenced by HYDRA, a model for structuring informal artifacts developed as part of project NATURE (Pohl & Haumer 1995) and based on the hypertext principle of nodes (Conklin, 1987)². Where MNLS differs from HYDRA is that its nodes are 'typed' and can therefore be verified against Product Data Synthesis elements. This in

¹ Subsection 26.1 of Defence Standard 00-55 states that "Formal methods will be the primary means used for the specification and design processes of Safety Related Software, but other methods (for example structured design methods) should be used as appropriate to complement the particular formal method chosen".
² HYDRA works over textual descriptions that are marked up and mapped to hypertext nodes which may then be interlinked.
Structuring Development Artifacts

turn means rules may be defined for analysis, as well as raising the possibility of developing specialised models where the content is regulated, as is the case with a structure for specifying Scenarios to be introduced in subsection 4.3.

As with both the Dexter model (Halasz & Schwartz, 1994) and HYDRA, MNLS nodes can be either composite or atomic. Composite Nodes are used exclusively for structuring and as such contain no user-information; rather they are composed of other nodes (composite and/or atomic). In contrast, atomic nodes are data carrying in the sense that they have some form of information content depending on their type. Those that reference a PDS entry (i.e. are PDS-checked) such as a module, property, function, or condition, etc. contain the name of a corresponding Build Element (which may be “qualified’ to express ‘scope’). Those that don’t are designated either as ‘non-significant’ (in the traceability sense) plain text, or else as null nodes. Strictly, the latter are non-information carrying, but rather a format preserving mechanism to which we return presently.

It is assumed that users enter text into a single atomic node (of type plain-text) via an appropriate interface operating over the structure. It is further assumed that this interface includes operations allowing text fragments to be selected (or ‘marked-up’) from this single block and then ‘split’ to create separate nodes. That is, with each split operation, the original node is transformed into three new atomic nodes (grouped by a composite) containing: i) text preceding the selection; ii) the subject selection itself; and iii) text following the selection. Preceding and following fragments may then be split further to an infinite number of levels, thereby yielding a tree-like structure.

Subject nodes (other than plain text) reference a PDS element and are therefore typed accordingly. Conversely preceding and following fragments (unless split further) are either plain text, or else null; the latter is used in situations where a subject node has no preceding or following content. For example in the statement ‘Fuel System transfer operations are independent from all other ....’, Fuel System could be selected as our subject node (a module) even though nothing comes before it; in which case, preceding fragment would be designated null, with the following fragment ‘transfer operations ....’ as plain text.

One further point; in order to minimise redundancy, we have sought to model the MNLS so that PDS-checked subject nodes exist only once. This assumes that the interface software is able to check whether its underlying object-base already contains an atomic node of the proposed type with the proposed label. For instance, if we were to establish a further structure containing a Fuel System subject (or alternatively to repeat it within the same structure), then the same node created for our previous statement involving this element would simply be reused; i.e., the Fuel System module node would now be part of two structures.

Following definition of the MNLS, we present a brief worked example that provides a ‘walk-through’ of these concepts at both the physical and logical level.

---

3 Tools such as DOORS and RTM (sections 2.3.2.1 and 2.3.2.2) allow text to be split in a similar way, normally to separate paragraphs into individual requirements. MNLS seeks to provide a formal basis for artifacts structured using such an approach.
## 4.2.3.2 Meta-model Definitions

We now introduce the MATrA Natural Language Structure stated in UML (4.2.3.2.1), together with OCL constraints over elements of the model (4.2.3.2.2) and O-Telos implementation of its base classes (4.2.3.2.3).

### 4.2.3.2.1 MATrA Natural Language Structure Meta-model

Figures 4.1 and 4.2 show elements and associations making up the MNLS meta-model. Its core class, MatraNaturalLanguageStructure instantiates the TraceabilityStructure (rather than DevelopmentStructure) meta-class. This reflects the fact an MNLS may be incorporated into development, assessment or product management artifacts (cf. its use to represent pre and post-conditions of the Use Case Model in subsection 4.3 and also the detection column for FMEA tables in 5.3).

![MATrA Natural Language Structure: Elements](image)

A MatraNaturalLanguageStructure is defined as an aggregation of zero-or-more composite nodes (MatraNLSComposite), plain text nodes (PlainTextNode), null nodes (NullNode), module nodes (ModuleNode), function nodes (FunctionNode), transaction nodes (TransactionNode), property nodes (PropertyNode), condition nodes (ConditionNode), and interface nodes (InterfaceNode).
(PropertyNode), specification nodes (SpecificationNode), condition nodes (ConditionNode), interface nodes (InterfaceNode) and input-output nodes (InputOutputNode).

Figure 4.2 - 'MATrA Natural Language Structure : Associations'

Figure 4.2 indicates how these elements (all of which instantiate the StructureElement meta-class) are related. Essentially, NullNode, PlainTextNode, ModuleNode, FunctionNode, TransactionNode, PropertyNode, SpecificationNode, ConditionNode, InterfaceNode and InputOutputNode are subtypes of the abstract MatraNLSAtomicNode class. All barring NullNode and PlainTextNode have an appropriate property of type String that is cross checked against the Product Data Synthesis (see rule 4.2.3.2.2(1)); PlainTextNode also has a property (mnls_text) which provides a placeholder for non-significant String. To simplify the
modelling, we choose to express the reflexive qualified_by association on MatraNLSAtomicNode and to suppress its use on NullNode using an appropriate constraint (see 4.2.3.2.2(2a))\(^4\). We do not however do the same for PlainTextNode as this may conceivably have its own qualification (of the same type) to clarify non-significant terms.

MatraNLSAtomicNode is a subtype of the abstract MatraNLSNode, as is MatraNLSComposite; the latter is also defined as an aggregation of MatraNLSAtomicNode type elements over association rolenames preceding_fragment and following_fragment, as well as of subject_node with type MatraNLSAtomicNode (a constraint suppresses use of NullNode as the subject_node - see 4.2.3.2.2(3)). This allows us to model the concepts involved in splitting nodes outlined in 4.2.3.1. Notice that multiplicity on the aggregation side of the subject_node association is one-to-many (again suppressed for the NullNode and PlainTextNode pairing - 4.2.3.2.2(4)), thereby capturing the redundancy minimisation intent also discussed in that section.

### 4.2.3.2.2 OCL Constraints

We express a small number of possible restrictions over the meta-model from 4.2.3.2.1. These are as follows:

1. Constraint to ensure that a corresponding BuildElement exists in the PDS for ModuleNode types (similar constraints can be stated for other types of node):

   **ModuleNode invariant**
   \[
   \text{self.allInstances->forall(m I self.bEelementAEO.build_element->exists(be I m.module_name = be.module_name))}
   \]

2. The following constraints apply to qualification of Atomic Nodes:

   a) Constraint to prevent qualification of a NullNode.

   **NullNode invariant**
   \[
   \text{self.allInstances->forall(n I n.qualified_by->size = 0)}
   \]

   b) Constraint to force qualification (i.e., scoping) of Property Nodes which may otherwise be ambiguous. Other MatraNLSAtomicNode subtypes can have similar constraints (which may be desirable properties, rather than invariants) though these are not shown.

   **PropertyNode invariant**
   \[
   \text{self.allInstances->forall(p I p.qualified_by->size = 1)}
   \]

   c) Constraint to prevent nonsensical qualification; e.g., a module qualified_by a property (again this is just one possible example of such a constraint - at the very least a comparable constraint inhibiting NullNode qualifiers is required).

   **ModuleNode invariant**
   \[
   \text{self.allInstances->forall(m I not m.qualified_by.oclType = PropertyNode)}
   \]

---

\(^4\) Whilst this means that NullNode is strictly not a specialisation of MatraNLSAtomicNode, treating it as such removes the need for further type-preserving abstract classes.
d) Constraint to ensure qualifiers are correct; e.g., where a module qualifies a property, then the PDS must contain a BuildElement of type Module defined with that particular Property.

PropertyNode invariant
self.allInstances->forall(p l
self.qualified_by->select(octType = ModuleNode)->not exists(m l
p.qualified_by->includes(m) and
m. BuildElementAO.build_element.hasProperty.target.property_name
->not includes(p.property_name)))

3. Constraint that a MatraNLSComposite subject_node is not of type NullNode, and that preceding_fragment and following_fragment are either of type NullNode or PlainTextNode, or else are a MatraNLSComposite.

MatraNLSComposite invariant
self.allInstances->forall(c l
c.subject_node.oclType <> NullNode and
(c.preceding_fragment.oclType = PlainTextNode or c.preceding_fragment.oclType = NullNode or
c.preceding_fragment.oclType = MatraNLSComposite) and
(c.following_fragment.oclType = PlainTextNode or c.following_fragment.oclType = NullNode or
c.following_fragment.oclType = MatraNLSComposite))

4. Constraint further restricting multiplicity to one (rather than 1..* as shown on the Class Diagram) on the aggregate side of MatraNLSComposite for NullNode and PlainTextNode types.

MatraNLSAtomicNode invariant
self.allInstances->forall(c l not ((c.oclType = NullNode or
c.oclType = PlainTextNode) and c.matraNLSComposite->size > 1))

5. Constraint to enforce uniqueness of ModuleNode elements; similar constraints (not shown) apply to other PDS-checked node types.

ModuleNode invariant
self.allInstances->forall(m1, m2 l not (m1 <> m2 and m1.module_name = m2.module_name))

4.2.3.2.3 O-Telos Implementation of MNLS Base Classes

The following O-Telos code implements base classes for the meta-model in subsection 4.2.3.2.1:-

Definition of MNLS Constituents

MatraNLSNode in StructureElement,
SimpleClass isA AerospaceEngineeringObject with
constraint
abstract_Nde: $ forall t/Token
s/SimpleClass
(t in s) ==> not (t in MatraNLSNode) $ end

MatraNLSAtomicNode in StructureElement,
SimpleClass isA MatraNLSNode with
has_part
qualified_by : MatraNLSAtomicNode
constraint
abstract_Atn: $ forall t/Token
s/SimpleClass
(t in s) ==> not (t in MatraNLSAtomicNode) $ end

MatraNLSComposite in StructureElement,
SimpleClass isA MatraNLSNode with
has_part
preceding_fragment : MatraNLSNode;
subject_node : MatraNLSAtomicNode;
following_fragment : MatraNLSComposite
end

PlainTextNode in StructureElement,
SimpleClass isA MatraNLSAtomicNode with
has_property
mnls_text : String
end

NullNode in StructureElement, SimpleClass isA MatraNLSAtomicNode end

ModuleNode in StructureElement,
4.2.3.3 Worked Example

Extensive application of the MATrA Natural Language Structure is demonstrated by the case studies in sections 6.2 and 6.3. However, to provide an overview of the structure at both a logical (i.e. user) level and a physical level (i.e., the underlying mechanics), we ‘walk-through’ an example based on a simple requirement statement. Note that in this thesis, MNLS is considered in terms of its integration with other structures, such as Use Case and Failure Modes & Effects Analysis. Whilst the following demonstrates a potential application to requirements specification, our research indicates the need for a dedicated requirements structure with additional features and verification rules. This is acknowledged as an area for future work and discussed in subsection 7.4.5.

Meantime, consider the following statement. It contains several fragments that can be usefully extracted (in the sense of being transformed into separate formal objects) for traceability purposes:

"Total-Fuel-Capacity shall be >= 195,000 litres of useable fuel"

In this case, we choose to demonstrate non-sequential splitting of nodes such that the engineer first selects the ‘>= 195,000 litres’ text-fragment, which will become a SpecificationNode. This action destroys the initial atomic node creating in its place a composite C, comprising three atomic nodes - $C_pA_p$, $C_fA_f$ and $C_{p+1}A_{p+1}$. These contain the text before the marked section ("The Total-Fuel-Capacity shall- be "), the marked section itself (>=195,000 litres) and the text following the marked section (‘ of

---

5 In the sense that process does not follow a strict left-to-right pattern.
6 The subscripts $p$ and $f$ denote preceding, subject and following nodes.
7 Note, this assumes that Specification may be written to and contrasts with the other BuildElement types which simply provide an integrity check for incoming data (i.e to the Traceability Workspace) transferred from CASE tools.
useable fuel.'). A Commentary (of type PlainTextNode - CjAfQ) is then attached to CjAf qualifying exactly what is meant by the term 'useable fuel'. This is summarised in figure 4.3.

![Figure 4.3 - 'MNLS Sample Population : Node Splitting Stage 1'](image)

Next, the engineer selects the property 'Total-Fuel-Capacity', which as previously indicated is now part of CjAp. Once again, this node is destroyed and superseded by a composite C2 made up of atomic nodes C2A, C2A, and C2Af where C2A (containing the Total-Fuel-Capacity fragment) is demarcated as being of type PropertyNode.

Note that C2A becomes a null node which, as stated previously, occurs wherever the marked selection is located at the beginning (as is the case here) or end of a text fragment, whilst C2Af contains the PlainTextNode 'shall-be'. The Total-Fuel-Capacity PropertyNode is also qualified (see C2AfQ) by specifying its host module, in this case 'Fuel System'. This is summarised in figure 4.4.

![Figure 4.4 - 'MNLS Sample Population : Node Splitting Stage 2'](image)

O-Telos code for the completed MNLS - i.e., following 'Stage 2' (figure 4.4 above) is as follows:-

```plaintext
ExampleMNLS in MatraNaturalLanguageStructure, Token with
mnls_composite

mnlsComposite1 : C1;

mnlsComposite2 : C2

mnls_plain_text
```

102
4.2.4 Summary

Our investigation has shown that natural language continues to be a highly important format for representing artifacts used by engineers within the aerospace industry. Accordingly means are required to allow traceability of elements embedded within the text of such artifacts.

This subsection has modified and built-on an existing meta-model (based on the hypertext principle of nodes) produced by partners from project NATURE, allowing its integration within the MATrA framework. Our main modification to the original model was to introduce node typing such that node types correspond to (and may therefore be verified against) build element types from the Product Data Synthesis.

A worked example featuring the MATrA Natural Language Structure was considered at both the logical and physical levels. Further demonstration will appear in the main case studies (Chapter Six), with a full evaluation provided in Chapter Seven.
4.3 User Centred Requirements Structure

4.3.1 Introduction
In this subsection, we introduce an integrated structure comprising meta-models that capture syntactic elements of three complementary requirements elicitation, analysis and documentation notations: Use Case Models, Scenarios and Message Sequence Charts; a formal semantics sufficient to preserve well-formedness and usage restrictions is also provided. We refer to this integrated structure as the User Centred Requirements Structure (UCRS).

4.3.2 Motivation
It is now universally accepted that frailties in the requirements phase of systems engineering projects can prove a major factor in their failure. A corollary of this (as indicated in Chapter One) has been the transition of Requirements Engineering, from a special interest group within the software and systems engineering fraternity, to a major discipline in its own right. Accordingly, a range of formal, semi-formal and informal approaches have sought to address issues involved in achieving the desired output from RE (i.e., a consistent and complete specification capable of forming the basis of the next phase of system development). While developers have in the past succumbed to (often) unsubstantiated claims and biases made in support of some of these approaches, as RE matures, practitioners are more able to distinguish what ‘works’ - either in terms of aiding specification directly, or else in supporting communication among agents involved in the process - from what palpably does not.

Use Case Models, Scenarios and Message Sequence Charts are all complementary and established notations used in the requirements analysis phase of projects. They allow developers to model from a user perspective the high-level functionality of, and interactions with, a target system. In the aerospace industry for example they have been used for a variety of civil and military applications, from exchanges between the ground crew and Fuel System, to pilot interaction with Mission Planning and Navigation Systems (BAe 1999).

We now provide a brief overview of Use Case Models, Scenarios and Message Sequence Charts before introducing a novel meta-model (the User Centred Requirements Structure) that integrates these approaches in a format amenable to traceability.

4.3.2.1 Use Case Overview
Use cases have been acknowledged by most methodologists in the field as an effective means of support for user centred requirements analysis, i.e. the capture of requirements from a user perspective. Use Case diagrams show abstract interactions between a target system and its environment, as well as capturing the scope of functionality. And because they express such information in terms of the problem domain vocabulary, they are readily understandable to participants with a non-technical background.

Use cases have previously been a feature of development methods by Jacobson et al. (1993), Rumbaugh et al. (1991) and Booch (1994), although their profile has undoubtedly been raised by being a constituent of the Unified Modelling Language.
Use Case diagrams are based around a simple, though deceptively rich notation the main elements of which are actors and use cases. Actors represent sets of external entities who share some common characteristics in terms of how they interact with the target system. These may be human, or else some form of quasi-autonomous system built from hardware/software components. Actors are denoted on Use Case diagrams using 'stick-men' symbols.

A single use case describes a subset of system functionality which delivers some unit of work to an actor or actors in response to an initial stimulus by one particular actor. Use cases are denoted on diagrams using a simple name bearing ellipse, thereby reinforcing the point that this form of analysis regards the system as a ‘black-box’ (i.e., functionality is exposed, whilst design specification - if such details are known at this stage - is hidden). Interactions between actor and use case symbols are modelled using arcs. A use case can also be annotated with pre and post conditions.

Jacobson et al. (1993) provide two means of combining use cases via the uses (also known as includes) and extends associations. The former embeds new behaviour into a complete base case, while the latter embeds a fragmentary sub-sequence (analogous to a sub-routine) as a necessary part of a larger case enabling the same behaviour to occur in several otherwise disjoint use cases. In essence, these associations can be said to support the principles of aggregation (uses/includes) and inheritance (extends).

Figure 4.5 depicts a basic use case example for a missile subsystem known as the AGA (Attitude Guidance Autopilot). The diagram is reproduced from Grigg & Henderson (2000) in which use case diagrams provide a front-end to real-time transaction (RTT) models (Haveman et al., 1997). RTTs are an abstraction mechanism enabling the recording and analysis of functional and timing properties in the early stages of development for distributed and reactive real-time systems. Figure 4.5 identifies two use cases/RTTs, the Autopilot and the Attitude Guidance (notice that the former includes functionality of the latter), together with three actor entities present within the host missile system, namely Sensor(s), an Information Processing System (IPS) and a Fin Controller.

Figure 4.5 - 'Use Case Diagram of the AGA Sub-system'

Although figure 4.5 shows which actors interact with which use cases, it tells us nothing about specific

---

8 Note that Rumbaugh (1994) questions whether the extends and uses associations are fundamentally different, suggesting that both may be treated as "special cases" of some unifying association which he terms adds between a base case and an additional case in which the multiplicity is (normally) either one for the uses case or optional for the extends case. This is reflected in our treatment of paths through use cases, as described by the Interaction View in 4.3.3.4.1.
usage - i.e., the sequence of individual interactions making up each potential end-to-end transaction between the target system and its environment. Scenarios - natural language descriptions documenting different paths through a use case - provide just such a facility; we consider these in the next subsection.

4.3.2.2 Scenarios Overview

The terms use case and scenario (in the context of projections of future system usage) are often used synonymously. However, according to our interpretation, each scenario represents a separate order-dependent sequence of message events conveying a 'path' through a use case, such that the use case being described is the set of all such scenarios. Put another way, each scenario can be regarded as an instance of a use case in much the same way that UML Object Models instantiate a Class Diagram. Like use cases, scenarios have also featured widely in object-oriented techniques including (Jacobson et al., 1993) and Graham (1996), as well as the Unified Modelling Language.

The overriding advantage of scenarios (at least those where the actors concerned are human) is that users can 'walk' analysts through typical usage situations. Such walkthroughs can take many forms, ranging from simple verbal descriptions, to those involving role play. In doing so scenarios can help validate existing requirements (Sutcliffe et al., 1998), uncover less obvious requirements (including those for exception handling) and surface hidden assumptions (Potts et al., 1994). Further, in an extensive review of their potential application and benefits, Weidenhaupt et al. (1998) also identifies scenarios as a means of reducing complexity (by enforcing a usage-oriented decomposition of requirements from an early stage), as well as promoting agreement among stakeholders (by grounding discussions and focusing negotiations during trade-off analysis). The fact they largely abstract away from design and implementation issues and are normally stated in natural language also means there are no barriers to participation in scenario based requirements analysis.

The text of each message event within a scenario typically includes a message sender and message receiver, a verb-phrase indicating a sender's action, together with an indication of the message content; sender and receiver correspond to the actor and target system entities from the Use Case Model. Sometimes, the system acts as both sender and receiver, which normally indicates (at least for the purpose of this discussion) that some internal action (e.g., a verification, calculation or update operation) is performed on data received in a previous exchange.

The following scenario (table 4.1) takes a normal path (i.e., no exceptional events) through the Autopilot use case from figure 4.5; the unit of work it describes is the provision of guidance data by the AGA to the Fin Controller subsystem. The first, third fourth and sixth events describe conventional information exchanges between the target system (AGA) and external actor entities (Sensors, Information Processing System and Fin controller), whereas events two and five are examples of internal system actions (both involving different forms of calculation).

---

9 Note, different interpretations of the term scenario from other domains have been proposed; e.g., as a means for describing socio-technical systems (Kyng, 1995).

10 Scenarios have also received formal treatment (cf. Haia et al. 1994; Rolland & Anchor, 1998), as well as being represented using tabular and storyboard techniques (cf. Rubin & Goldberg, 1992; Karat & Bennett, 1991)
Provide Guidance Data to Fin Controller - Normal Path

1. The Sensor sends position data to the AGA subsystem;
2. The AGA subsystem calculates the missile attitude;
3. The AGA subsystem sends missile attitude to the Information Processing System;
4. The Information Processing System sends aim point data to AGA subsystem;
5. The AGA subsystem calculates guidance data;
6. The AGA subsystem sends guidance data to the Fin Controller.

Table 4.1 - ‘Example Scenario for the Autopilot Use Case’

The text of these simple one-line instructions can be given greater clarity when the scenario is viewed in diagrammatic form using a Message Sequence Chart as described in the next subsection.

4.3.2.3 Message Sequence Chart Overview

Message Sequence Charts (MSCs) are a widely used graphical notation for visualising the temporal nature of interactions between system components or, as is our interest, a target system and its host environment. Like scenarios, they do not seek to express complete behaviour, but rather a single execution trace (i.e., path). Historically, Message Sequence Charts have mainly been applied to the area of telecommunication systems (ITU-T, 1993), although the DCSC is to investigate the possibility of their use with RTTs since both are concerned with specifying end-to-end threads of functionality. Variations on the MSC are a feature of UML and other object-based approaches (e.g., Selic et al.).

Figure 4.6 - ‘Message Sequence Chart for the Provide Guidance Data to Fin Controller Scenario (Normal Path)’

Figure 4.6 presents a visualisation of the scenario in table 4.1 which features the core graphical syntax of all Message Sequence Charts. Vertical axes represent instances, i.e., participants of the use case (either actor instances or the target system) as indicated through named rectangles at their head. Conversely, directed horizontal lines indicate messages where each message starts at an originator instance and ends at a target instance. Some messages are internal to an instance; i.e., they have the same originator and target instance (e.g., “calculation of missile attitude” and “calculation of guidance...
User-Centred Requirements Structure

data" from figure 4.6). Given our intended usage of MSCs in modelling interactions between a system and its environment, these are exclusively actions that are intrinsic to the target system. We note however that software-oriented applications based on object-modelling also use Message Sequence Charts to show the precise interaction of objects within the target system itself.

The instance axis orders events in time downwards. Translated in the context of figure 4.6, this means that “missile attitude” is sent before “aim point data”, and so on. However the time axis depicts only sequence in the sense that the scale is non-linear and therefore no semantic importance (e.g., interval) should be implied by the spacing between messages.

The axis/instance and message elements are sufficient to construct most Message Sequence Charts. Nevertheless, extensions to the core syntax make it possible to express additional information such as arrival patterns (e.g., periodic or aperiodic), synchronisation (balking, waiting, asynchronous) and broadcast messages (which originate from a sender instance at the same time point and are received by \( n > 1 \) instances at potentially different times). These are all features of the UML variant of MSCs known as Sequence Diagrams which can also contain state marks placed on the vertical axes in order to provide a bridge to the corresponding Statechart diagrams. Also of note are extensions proposed by Regnell et al. (1996) who employ MSCs in conjunction with Episodes (proposed by Potts et al. [1994] as a means of grouping the fine-grained events occurring within a use case into coherent parts) such that each MSC represents a single episode and a set of MSCs depicts a scenario.

4.3.3 Tracing User Centred Requirements in MATrA: UCRS - An Integrated Use Case & Interaction Structure

The notations discussed in subsections 4.3.2.1 through 4.3.2.3 provide a sound basis for the elicitation, analysis and documentation of user centred functional requirements in a format amenable to all stakeholders with an interest in this process. We now describe the User Centred Requirements Structure, a novel meta-model that integrates Use Case Models, Scenarios and Message Sequence Charts and in doing so, provides strong traceability between their constituent elements.

4.3.3.1 Concepts

As indicated in Chapter One (subsection 1.3), notations used by avionics engineers fall into two broad categories: those with a well-defined syntax and semantics and those that are less rigorously defined but which offer flexibility as a result. Use Case Models, Scenarios and to a lesser extent Message Sequence Charts are of the latter such that UCRS integrates and extends these notations to serve domain specific needs. The structure has been shaped by discussions with practitioners and examination of actual specifications featuring these notations.

Use Case diagrams in MATrA support all of the concepts outlined in subsection 4.3.2.1, i.e., actors, interactions and use cases, as well as means for combining the latter through includes and extends associations and for their description through pre and post conditions. We do however include the additional notion of a service (Regnell et al., 1996); i.e., a demarcated set of functionality encapsulated
within a single package (not in the UML sense)\(^\text{11}\).

Discussions with practitioners suggest that user-centred requirements for even relatively simple systems (in terms of functionality) can yield a large number of use cases. In MATrA we manage their number through the notion of views. Essentially, a view is a composition of one or more Use Case Models; reuse of actor and use case entities across these models is facilitated by all elements of a model also being elements of the view to which that model belongs. Therefore, the set of elements available to a model is the union of those elements introduced by itself and all models within the same view.

As regards descriptions of use cases, MATrA supports both textual and Message Sequence Chart formats. Perhaps due to their traditional prose representation, textual scenarios lack even a quasi standard. There are however some works proposing alternative representations as well as suggestions on what well-formed events should contain, ranging from structured natural language techniques (e.g., Leite et al. 1997) to more mathematical approaches (e.g., Rolland & Achour, 1998). On balance, we favour a natural language representation (for reasons of usability), underpinned with a lightweight formal semantics. This is realised in part through fine-grained modelling, allowing us to identify and validate event primitives against the PDS and also to impose a degree of well-formedness.

The latter is further aided by differentiating between event types, for which we take work by Rolland & Anchor (ibid.) as our steer. They identify two forms of atomic event: i) message communication between instances (i.e., actors in the Use Case View) and the target system, which may be further qualified as a service request, service provision, information request or information provision; and ii) internal system actions, which constitute operations intrinsic to the target system.

In addition, a target system can host Timer sub-modules which they control in terms of set and reset events. Timers can also time-out, although the host plays a passive role in events of this form. For more information on the use of timers and timing events within use case descriptions, readers are referred to work by Regnell et al. (1996) which also inspired the UCRS mechanisms for grouping of sequential events and the expression of upper and lower bounds - both of which are necessary to allow iteration within scenarios.

Message Sequence Charts of the UCRS contain the same core elements as textual scenarios. Indeed, other than the fact that MSC communication events differentiate between types of message based on arrival pattern (periodic or aperiodic), synchronisation (simplex, synchronous, balking, etc.) and delay\(^\text{12}\), they differ only in terms of presentation\(^\text{13}\). Hence, there are clear optimisation, consistency and traceability benefits to be gained by providing a common underlying representation. This is the approach adopted within MATrA through our Interaction Model.

\(^{11}\) Graphically, services (represented by named rectangles) are the only deviation from traditional use case diagrams. Pre and post conditions simply annotate the conventional use case ellipse symbol.

\(^{12}\) All of these properties have a corresponding graphical convention.

\(^{13}\) This was demonstrated through our example in 4.3.2.2 and 4.3.2.3 (albeit using a subset of the concepts outlined above)
User-Centred Requirements Structure

Notwithstanding the commonality between textual scenarios and MSCs, the labelling of apparently equivalent communication and action events often differs between the two formats. This is probably explained by a need for the former to adhere to the rules of English grammar, whereas the latter merely annotate symbols. To resolve the issue, we propose that each message and action is able to use an appropriately worded parameter which is displayed in place of the message/action name (note the latter are still required as they provide the basis for PDS verification).

4.3.3.1.1 UCRS Notation

At a logical level, some refinements to the way Scenarios and Message Sequence Charts are presented is necessary. For instance, with the textual representation, each scenario event is preceded by its type: C[SR], C[SP], C[IR], or C[IP] for Communication events (Service Request/Provision and Information Request/Provision respectively), or alternatively, 'A' for Action or 'T' for Timing events. With the MSC equivalent, we place communication event types below their corresponding message arrow.

Our policy of validating event content against the PDS requires us to identify sender, receiver, message, action and other relevant primitives within text strings. Accordingly, these are enclosed within square brackets preceded by a typed subscript; e.g., [sr, sender], [rc, receiver], [msg, message], [act, action], [timer, timer]; for message elements where a parameter is being used, the actual name (corresponding to its PDS entry) appears in subscript following the parameter label - i.e., [msg parameter, PDS name]. Timer events over the (host) instance are also demarcated using [timer-set, Instance], [reset, Instance] and [host-on-timeout, Instance]. With the graphical representation, this information is of course conveyed using symbols (note a parameter appears above the message arrow and PDS name below); besides those introduced in 4.2.2.3, we add the following to depict timer (set, reset and time-out) events.

\[\text{\textbullet} \text{Set. The timer is activated.}\]
\[\text{\textbullet} \text{Reset. The timer is deactivated.}\]
\[\text{\textbullet} \text{Time-out. The timer expires.}\]

Figure 4.7 - 'MSC Timer Event Types (from Regnell et al., 1996)'

Event groups in textual scenarios are enclosed within a rectangle and preceded by upper and lower bounds. For MSCs, the grouping formalism is displayed using a shaded rectangle. Readers are referred to subsection 6.2 and to Appendix C, Part Two for examples of the notation.

4.3.3.2 User Centred Requirements Structure (Definition)

As previously indicated, User Centred Requirements Structure provides an integrated approach to the representation and traceability of Use Case diagrams, Scenarios and Message Sequence Charts. The structure is essentially a 'container' that draws together elements from Use Case and Interaction Views, where the latter is simply an abstraction of concepts modelled using Scenarios and Message Sequence Charts.
In modelling UCRS, we make a number of assumptions. Specifically, that the structure has bespoke tool support and that instances ported to the traceability Workspace include a Use Case View. We assume this contains at least one Use Case Model featuring a minimum definition of one use case, one actor and one interaction. If present, we further assume the Interaction View contains at least one Interaction Model and that this model is minimally defined by a single scenario with at least one communication event in which a message is exchanged between two instances (sender and receiver, where the target system is receiver). This may populate either a textual or MSC viewpoint, with provision made in the form of rules (see subsection 4.3.3.4.2) to derive one from the other, though user intervention is required to construct legitimate prose event statements around core elements from a sequence chart.

A detailed explanation of the Use Case and Interaction Views appears in subsections 4.3.3.3 and 4.3.3.4 respectively. However, as a precursor to this we describe a meta-model (4.3.3.2.1), OCL constraints (4.3.3.2.2) and O-Telos implementation of base-classes (4.3.3.2.3) for the User Centred Requirements Structure itself.

**4.3.3.2.1 User Centred Requirements Structure Meta-model**

The User Centred Requirements Structure (figure 4.8) is as an aggregation of a Use Case View (UseCaseView) and Interaction View (InteractionView). Its core class (UserCentredRequirementsStructure) - an instance of the DevelopmentStructure metaclass - can be compared with a banner comment or header in programming language terminology. As with most such classes in the MATrA Workspace, it contains a small number of attributes for configuration management - in this case date (ucrs_date) and subject module (subject_module) which is verified against the Product Data Synthesis (see subsection 4.3.3.2.2). UserCentredRequirementsStructure also provides means to record comments through aggregation to a MatraNaturalLanguageStructure, thereby allowing primitives embedded within comment strings to be linked to other models or model elements.

![Development Structure Diagram](image)

**Figure 4.8 - 'User Centred Requirements Structure (UCRS)'**

**4.3.3.2.2 OCL Constraints over the User Centred Requirements Structure**

The following constraint is applied to UCRS in order to verify its subject_module property exists in the Product Data Synthesis (navigation is via the BEmodelAEO AerospaceLinkEntity):

```ocl
t{UserCentredRequirementsStructure} invariant
    self.allInstances->forall(u |self.bEmodelAEO.build_element->exists(be |u.subject_module = be.module_name))
```
4.3.3.2.3 O-Telos Base Classes for User Centred Requirements Meta-model

O-Telos implementation of the base classes featured in figure 4.8 (4.3.3.2.1) is as follows:-

Definition of User Centred Requirements Structure

UserCentredRequirementsStructure in DevelopmentStructure, SimpleClass isA AerospaceEngineeringObject with has_property
  subject_module : String;
  ucrs_date : Date
has_structure
  ucrs_comments : MatraNaturalLanguageStructure;
  use_case_view : UseCaseView;
  interaction_view : InteractionView
end

In the following sections, we introduce the constituents of User Centred Requirements Structure:-
- Use Case View (4.3.3.3) and
- Interaction View (4.3.3.4)

4.3.3.3 Use Case View

Use Case View captures the elements necessary to construct Use Case Models. Again, we represent these in terms of a meta-model (4.3.3.3.1), OCL constraints (4.3.3.3.2) and O-Telos implementation of base-classes (4.3.3.3.3).

4.3.3.3.1 Use Case View Meta-model

Elements of Use Case View (UseCaseView), with its date (ucv_date) attribute and MNLS comment facility are depicted in figure 4.9. Its core class instantiates DevelopmentStructure and is defined as an aggregation of the use case diagram elements described in subsection 4.3.2.1 - namely use case (UseCase) and actor (Actor) entities, together with interaction (Interaction), includes (Includes) and extends (Extends) associations - all of which instantiate the StructureElement meta-class. Means to record pre and post conditions using MATrA Natural Language Structures are also a feature (enabling PDS verification of embedded ConditionNode primitives as appropriate).

Use cases and actors have (unique) name attributes (use_case_name and actor_name respectively) to enable meaningful identification, and for verifying instances of these classes against the Product Data Synthesis. UseCase also contains a boolean attribute (sub_case) to indicate whether instances must combine with others in order to deliver a 'unit of work'. In addition, the Extends class bears a condition rubric (extends_condition) which states the circumstances under which one use case may legitimately extend another. It is also worth noting that the cardinality of association between Actor and UseCaseView is one-to-many on both sides. The underlying rationale is that an actor can conceivably be an external entity to several aircraft systems and hence potentially contribute to many different views.

Remaining UseCaseView constituents are Service (described in terms of one or more use cases through an aggregation association over the service_use_case rolename) and Use Case Model (UseCaseModel). The former is an instance of StructureElement, whilst the latter instantiates DevelopmentStructure and is again analogous to a program banner.
Structuring Development Artifacts

Figure 4.9 - 'UCRS - Use Case View : Elements'

Figure 4.10 - 'UCRS - Use Case Model : Elements'
A UseCaseModel (figure 4.10) is an aggregation of the diagramming elements described when introducing UseCaseView; recall that in order to foster reuse, all such elements (UseCase, Actor, Includes, Extends, etc.) simultaneously belong both to a model and the view to which that model belongs (associations which can be derived using deductive rules - see ii.8 in 4.3.3.3.2 for instance). Accordingly, all elements carry a one-to-many association on the UseCaseModel side.

Associations between UseCaseModel elements are illustrated in figure 4.11. Of particular interest are the transitive_includes and transitive_extended_by associations on the UseCase class. As the names suggest, their purpose is to capture the transitive closure of Includes and Extends. It is also worth explaining the association ends on these two classes; 'base' ends (includes_base and extends_base) signify the including and extending use cases such that where 'a' includes 'b' and 'a' extends 'b', 'a' is the base in both, and 'b' is the element being either included or extended. The other significant aspect of this particular model is the introduction of an (abstract) Interactor class; since both actors and use cases send and receive messages, it is convenient to generalise this behaviour.

4.3.3.3.2 OCL Constraints over Use Case View Meta-model
We have defined a number of constraints and rules over elements of UseCaseView which are grouped on the basis of consistency and well-formedness. A selection of these are included below (readers are referred to Appendix A, Part One for the remainder).

1. Consistency
1. Constraint to ensure that use cases are declared as Transactions of the subject_module in the Product Data Synthesis.

UseCase Invariant
self.allInstances->forall(u | self.useCaseView.userCentredRequirementsStructure.bEmodelAEO.build_element->exists (be | self.bEelementAEO.build_element->exists (t | u.useCaseView.userCentredRequirementsStructure.subject_module = be.module_name and u.bEelementAEO.build_element->includes(t) and

Figure 4.11 - 'UCRS - Use Case Model : Associations'
u.use_case_name = t.transaction_name and
be.behavesAccordingTo.target->includes())

2. Constraint to ensure the target system is not among Actor elements (i.e., all actors are different from
the subject_module).

Actor invariant
self.allInstances->forall(a |
a.actor_name <> a.useCaseView.userCentredRequirementsStructure.subject_module)

3. Constraint to ensure actors exist in the PDS as modules interfaced with by the subject_module.

Actor invariant
self.allInstances->forall(a |
sel.useCaseView.userCentredRequirementsStructure.bEelementAEO.build_element->exists(be |
sel.bEelementAEO.build_element->exists(m |
sel.useCaseView.userCentredRequirementsStructure.subject_module = be.module_name and
sel.bEelementAEO.build_element->includes(m) and
a.actor_name = m.module_name and
(UsesInterfacesTo.target->includes(m) or m.usesInterfacesTo.target->includes(b)) )))

ii. Well-Formedness

1. Constraint to ensure each interaction involves an Actor and a UseCase.

Interaction invariant
self.allInstances->forall(i |
(i.interactor_1.oclType = Actor and i.interactor_2.oclType = UseCase) or
(i.interactor_1.oclType = UseCase and i.interactor_2.oclType = Actor))

2. Constraint to prevent 'dangling' use cases (i.e., those not attached to an interaction) - a similar
constraint is defined for actors (see Appendix, A Part 1).

UseCase invariant
self.allInstances->forall(u |
sel.useCaseView.ucv_interaction->exists(i |
i.interactor_1= u or i.interactor_2 = u))

3. Constraint to prevent a UseCase from including itself (a similar constraint is defined for «extends» -
see Appendix A, Part 1).

Includes invariant
self.allInstances->forall(i |
i.includes_base <> i.includes_include)

4. Constraint to ensure that at most one «includes» association exists between two use cases (see also
Appendix A, Part 1 for a similar constraint over «extends»).

Includes invariant
self.allInstances->forall(i1, i2 |
not (i1 <> i2 and i1.includes_base = i2.includes_base and i1.includes_include = i2.includes_include))
5. Constraint ensuring two use cases cannot include each other; i.e. not 'use case a' «includes» 'b' and 'use case b' «includes» 'a' (again, see Appendix A, Part 1 for similar constraint over «extends»).

**Includes Invariant**

\[
\text{self.allInstances->forall}(i1, i2 \mid
\text{not } (i1.\text{includes}_\text{include} = i2.\text{includes}_\text{base} \text{ and } i1.\text{includes}_\text{base} = i2.\text{includes}_\text{include}))
\]

6. Constraint to prevent cycles within «includes» associations (see Appendix A, Part 1 for a similar constraint over «extends»).

First, we must define a rule to determine the transitive closure for «includes» associations.

**UseCase**

\[
\text{self.allInstances->forall}(u1, u2 \mid
\text{self.includes->exists}(i1 \mid
i1.\text{includes}_\text{base} = u1 \text{ and } i1.\text{includes}_\text{include} = u2) \text{ or }
\text{self.allInstances->exists}(u3 \mid
\text{self.includes->exists}(i2 \mid
i2.\text{includes}_\text{base} = u1 \text{ and } i2.\text{includes}_\text{include} = u3 \text{ and } u3.\text{transitive_includes->includes}(u2)))
\implies
u1.\text{transitive_includes->includes}(u2)
\]

This rule allows us to specify the following constraint.

**UseCase invariant**

\[
\text{self.allInstances->forall}(u \mid \text{not } (u.\text{transitive_includes->includes}(u)))
\]

7. Constraint to ensure use cases for two services are not identical.

**Service Invariant**

\[
\text{self.allInstances->forall}(s1, s2 \mid
\text{not } (s1 <> s2 \text{ and } s1.\text{service_use_case->symmetricDifference}(s2.\text{service_use_case}->isEmpty)))
\]

8. Rule to ensure that all use cases associated with a particular model are also associated with the UseCaseView to which that model belongs (similar rules may be found in Appendix A, Part 1 for actors, interactions, etc.)

**UseCaseView**

\[
\text{self.allInstances->forall}(v \mid
\text{self.use_case_model.ucm_use_case->forall}(u \mid
v.\text{use_case_model.ucm_use_case->includes}(u)))
\implies
v.ucv_use_case->includes(u)
\]

9. Rule to ensure a UseCase appearing in more than one model reuses the original interaction details (see Appendix A, Part 1 for similar rules over other use case elements).

**UseCaseModel**

\[
\text{self.allInstances->forall}(m \mid
\text{self.useCaseView.ucv_interaction->forall}(i \mid
\text{self.ucm_use_case->exists}(u \mid
m.ucm_use_case->includes(u) \text{ and })
\]

Structuring Development Artifacts

m. useCaseView.ucv_interaction->includes(i) and ((i.interactor_1 = u or i.interactor_2 = u))) implies m.ucm_interaction->includes(i) and m.ucm_actor->includes(i. interactor 1->select(oclType = Actor)->union(i. interactor 2->select(oclType = Actor)))

4.3.3.3 O-Telos Base Classes for Use Case View Meta-model

We implement base classes for UseCaseView and UseCaseModel (shown in figures 4.9, 4.10 and 4.11) as follows:

**Definition of Use Case Model Constituents**

- **Interactor** in StructureElement, SimpleClass isA AerospaceEngineeringObject with constraint abstract_Int: $forall t/Token s/SimpleClass (t in s) == not (t in Interactor) $
- **Actor** in StructureElement, SimpleClass isA Interactor with has-property actor_name : String
- **UseCase** in StructureElement, SimpleClass isA Interactor with has-property
  - use_case_name : String;
  - sub_case : Bool
  has-structure
  - pre_condition : MatraNaturalLanguageStructure;
  - post_condition : MatraNaturalLanguageStructure
  has-transitive_part
  - transitive_includes : UseCase;
  - transitive_extended_by : UseCase
- **Interaction** in StructureElement, SimpleClass isA AerospaceEngineeringObject with has_part
  - interactor_1 : Interactor;
  - interactor_2 : Interactor
- **Includes** in StructureElement, SimpleClass isA AerospaceEngineeringObject with has_part
  - includes_base : UseCase;
  - includes_include : UseCase
- **Extends** in StructureElement, SimpleClass isA AerospaceEngineeringObject with has_property
  - extends_condition : String
  has_part
  - extends_base : UseCase;
  - extends_extend : UseCase

**Definition of Use Case View**

UseCaseView in DevelopmentStructure, SimpleClass isA AerospaceEngineeringObject with
- has_property
  - uc_data : Date
- has-structure
  - uc_comments : MatraNaturalLanguageStructure;
  - uc_pre_condition : MatraNaturalLanguageStructure;
  - uc_post_condition : MatraNaturalLanguageStructure
- has_element
  - uc_use_case : UseCase;
  - uc_actor : Actor;
  - uc_interaction : Interaction;
  - uc_includes : Includes;
  - uc_extends : Extends
- has_uses
  - uc_service : Service

**Definition of (additional) Use Case View Constituent**

Service in StructureElement, SimpleClass isA AerospaceEngineeringObject with has_property
- service_name : String
- has_part
  - service_use_case : UseCase

4.3.3.4 Interaction View

The Interaction View captures elements necessary to describe paths of events through use cases (from
the Use Case View) represented either in textual or Message Sequence Chart form. Once again, these elements are represented by a meta-model (4.3.3.4.1), OCL constraints (4.3.3.4.2) and O-Telos implementation of the base-classes (4.3.3.4.3).

4.3.3.4.1 Interaction View Meta-model

Interaction View (InteractionView) - an instantiation of the DevelopmentStructure meta-class, with date attribute (inv_date) and MNLS commenting facility - is essentially an aggregation of Interaction Models (InteractionModel).

Each InteractionModel (likewise an instance of DevelopmentStructure composed of StructureElement types) describes a use case from the Use Case View (as indicated by the describes_use_case attribute which is verified using a constraint - see i.1 in subsection 4.3.3.4.2). These descriptions take the form of scenarios (Scenario) whose varied involvement within an Interaction Model is evident from the rolenames inm_scenario, inm_included_scenario and inm_extension_scenario. Respectively, these denote paths through the use case being described, and paths through other use cases that are included in, or which extend the use case being described (via «includes» and «extends» associations). We also provide means of representing the different event types introduced in 4.3.3.1 - i.e., communication (CommunicationEvent), internal action (InternalActionEvent) and timing (TimingEvent) - see figure 4.12.

The remaining InteractionModel elements reflect our intent to provide a common underlying representation for textual Scenarios and Message Sequence Charts. These include constituent elements of events such as instance (Instance), message (Message), action (Action) and timer (Timer), as well message description (MessageDescription) and action description (ActionDescription) parameters alluded to in 4.3.3.1. Note that for optimisation reasons, all such elements have a multiplicity of one-to-many on the InteractionModel side.

In addition, we include constructs for describing dual perspectives on the different event types based around these elements. That is textual and MSC representations of communication events (TsnCommunication and MscCommunication), internal action events (TsnAction and MscAction) and timing events (TsnTiming and MscTiming), as well as means to form dual scenario viewpoints (TsnScenarioViewpoint and MscScenarioViewpoint) from these representations. We now consider elements and associations of the InteractionModel in more detail.

From figure 4.13, it can be seen that Scenario contains the String attribute scenario_title and Boolean attribute is_exception. The latter merely identifies use case paths which, for whatever reason (be it user intervention or a system malfunction) fail to deliver their intended ‘unit of work’. Reflexive aggregation associations on Scenario further describe relationships formed through «includes» and «extends», either directly (includes_scenario and extended_by_scenario) or transitively (transitive_includes_scenario and transitive_extended_by_scenario).

Scenarios are aggregated to events over scenario_event, included_event and extension_event rolenames.
These respectively denote events defining a scenario, and events that define other scenarios related to it by «includes» or «extends» associations (the latter pairing being optional). Note also the one-to-many aggregate side multiplicity indicating that each event may belong to several scenarios.

As previously indicated, each scenario is composed from CommunicationEvent, InternalActionEvent and TimingEvent types. It is convenient to generalise these as specialisations of an abstract Event class in order that associations such as follows_from and transitive_follows_from (identifying preceding events and the transitive closure of preceding events, both of which are used to maintain sequence) may be defined once and inherited by its subtypes.

Due to their (potential) many-to-many relationship with scenarios, each event may have several sequence numbers (SequenceNumber). If we take as an example, an event that closes a scenario - by for
Figure 4.13 - 'UCRS Interaction Model: Associations'
example displaying a message to a (receiver) instance; in the normal path, this event may have a sequence number of say ‘10’. However, with exceptions, for example where an instance chooses to abort an operation, several intermediate events that precede it are potentially omitted, thereby lowering the sequence number of that event within those particular scenarios. An event may exhibit further sequence numbers arising from «includes» or «extends» participation in a scenario. To accommodate these eventualities, Event employs three rolenames in its associations with SequenceNumber, namely sequence_no, included_seq_no and extension_seq_no.

Note that the composition convention is used to express Event-to-SequenceNumber aggregation. But for our efforts to minimise event redundancy, the latter would be simply an attribute of the former. However to realise multiple sequence numbers and allow their association with Scenarios, SequenceNumber is promoted to a class (an instantiation of ArtifactProperty). We should explain that sequence numbers are related to scenarios in order to establish correct usage; while an event may have several sequence numbers, each sequence number may belong to only one scenario. Correct coupling of sequence numbers to scenarios is maintained using appropriate constraints (for which readers are referred to Appendix A, Part Two).

Scenarios may also contain sequential combinations of (two or more) events known as event groups (EventGroup). Aggregation is defined over event_group, included_event_group and extension_event_group rolenames. Again these respectively denote event groups that define a scenario and those that define other scenarios related to it by «includes» or «extends» associations.

In order to permit iteration, lower and upper bounds (LowerBound and UpperBound) may be stated for both events and event groups. Note both are modelled as classes (instantiations of ArtifactProperty) to permit shared use (again through composition) of a single definition. Having introduced these basic building blocks, we are in a position to describe in more detail the elements and associations of scenarios and events.

Each scenario has a textual viewpoint (TsnViewpoint) and an MSC viewpoint (MscViewpoint) constructed from TsnCommunication, TsnAction and TsnTiming perspectives and MscCommunication, MscAction and MscTiming perspectives respectively (note the multiplicity allowing each perspective to be part of one or more viewpoints). We now describe the composition of each event type, thereby relating event perspectives to the constituent elements they share. However, before doing so, we introduce means to model the textual perspective on each which is essentially a specialisation of the MNLS from section 4.2 known as the Scenario Event Natural Language Structure (SENLS).

It can be seen from figure 4.14 that SENLS introduces specialisations of MatraNLSAtomicNode for key primitives of the Communication, Internal Action and Timing event types. Specifically, Message, Action, TimerDuration, Timer and Instance. Several rolenames reflect the diverse usage of Instance as a message sender or receiver (tsn_sender_node and tsn_receiver_node), a sender and receiver in the context of
internal actions (tsn_sdr_rcc_node) and finally in timer set, reset and time-out events (tsn_timer_set_node, tsn_timer_reset_node and tsn_host_on_timeout_node); constraints (see exclusive-or conditions in figure 4.14 and also the restrictions in Appendix A, Part 2) ensure correct combination of these primitives.

Note that each primitive (other than TimerDuration which is event specific) potentially belongs to many SENLSs and is verified against the Product Data Synthesis (e.g., instances correspond to Modules, messages to InputOutput and actions to Functions - readers are referred to 4.3.3.4.2 for these rule definitions, in particular i(3), i(4) and i(5)). We now consider definition of CommunicationEvent types.

CommunicationEvent (figure 4.15) includes the property interaction_type (of type InteractionType, an instantiation of ArtifactProperty and specialisation of String restricted to IR, IP, SR and SP denoting the interactions discussed in 4.3.3.1). Textual and Message Sequence Chart perspectives are represented by TsnCommunication and MscCommunication respectively; recall that whichever is populated, the other can be derived through an implementation of the rules in 4.3.3.4.2, i.e. (and Appendix A, Part 2). However readers are reminded of the need for user intervention in deriving prose representations of sequence chart events in order to produce meaningful and legitimate statements.
TsnCommunication is described by a Scenario Event Natural Language Structure, comprising sender and receiver Instance (over tsn_sender_node and tsn_receiver_node rolenames) and Message nodes. In the context of textual representations, each message may be assigned a message description (MessageDescription) parameter (as discussed in 4.3.3.1) using the tsn_msg_parameter rolename.

**Diagram:**

![Diagram of 'Interaction Model - Communication Event: Elements and Associations'](image)

Figure 4.15 - ‘Interaction Model - Communication Event: Elements and Associations’

MscCommunication has the attributes link name (link_name), synchronisation, frequency and delayed. Link name is of type String and allows nomination of a connection between the sender and receiver instances (which is checked against the PDS). However, given that UCRS is intended for use primarily during the requirements phase of a project, such information may be as yet undetermined; hence the default link name is ‘unspecified’. Meanwhile, the frequency and synchronisation attributes (whose eponymous types are both instantiations of ArtifactProperty and specialisations of String) capture arrival patterns (with default ‘aperiodic’) and synchronisation patterns (default ‘simplex’) respectively. Finally, the Boolean delayed attribute (default false) denotes a non-negligible pause in message transmission (relative to the overall dynamics of the target system).

MscCommunication is aggregated to Instance (over msc_sender_instance and msc_receiver_instance) and

---

14 Note, for added flexibility we could have allowed the textual perspective of Message to have several different parameters - potentially one for each SENLS it appears in. However, that would risk obscuring the original message and intended usage.

15 All case study examples featuring these properties (see section 6.2) are confined to default values.
Message elements - the same SENLS elements that describe TsnCommunication. As with textual
descriptions, sequence diagram messages may also feature a MessageDescription parameter assigned
using the msc_msg_parameter rolename.

As figure 4.16 indicates, InternalActionEvent is modelled in similar fashion to CommunicationEvent with
textual and Message Sequence Chart perspectives represented by TsnAction and MscAction respectively.
TsnAction is also described by a Scenario Event Natural Language Structure, comprising a
sender/receiver Instance (over rolename tsn_sdr_rcr_node) and Action node. In the context of textual
representations, actions may be assigned an action description (ActionDescription) parameter using the
tsn_act_parameter rolename.

MscAction is aggregated to the Instance (over msc_sdr_rcr_node) and Action elements of SENLS that
describe TsnAction. Again, sequence diagram actions may be given an ActionDescription parameter
through the msc_act_parameter rolename.

It can be seen from figure 4.17 that TimingEvent also follows the pattern of CommunicationEvent and
InternalActionEvent such that textual perspectives are represented by TsnTiming and sequence chart
perspectives by MscTiming. TsnTiming is likewise described as an SENLS, this time composed of an
Instance element over `tsn_timer_set_node`, `tsn_timer_reset_node` and `tsn_host_on_timeout_node` rolenames, denoting an initiator of timer set and reset events acting on some `Timer` element, with passive involvement in time-out events.

![Interaction Model - Timing Event: Elements and Associations](image)

**Figure 4.17 - Interaction Model - Timing Event: Elements and Associations**

It should be noted that `TimerDuration`, an aggregation component of `Timer` serves a different purpose to the message and action descriptions in `CommunicationEvent` and `InternalActionEvent`. Rather than providing an optional and alternative means of presentation for the aggregate class (i.e., `Message` or `Action`), `TimerDuration` is a necessary element of timer set events (and only timer set events) which states the timing interval. Note also that each `Timer` potentially appears in several timer set events and as such has several durations. However, each duration is event specific and so belongs to the textual and sequence chart perspectives of only one timer set event.

As with our previous event models, `MscTiming` is aggregated to the `Instance` (over `msc_timer_set_node`, `msc_timer_reset_node` and `msc_host_on_timeout_node` rolenames) and `Timer` elements of `SENLS` that describe `TsnTiming`, as it is to `TimerDuration`.

### 4.3.3.4.2 OCL Constraints over Interaction View Meta-model

Again, a number of constraints and rules over elements of the Interaction View have been defined. As with Use Case View, these are grouped on the basis of consistency and well-formedness. A selection of
User-Centred Requirements Structure

these are included below (while readers referred to Appendix A, Part Two for the remainder).

1. **Consistency**

1. Constraint to ensure a UseCase described in an InteractionModel exists in the UseCaseView.

**Interaction Model Invariant**

\[ \text{self.allInstances->forall(i | self.interactionView.userCentredRequirementsStructure.useCaseView.ucv_use_case->exists(u | i.describes_use_case = u.use_case_name))} \]

2. Constraint over Communication Events (textual perspective); ensures there exists an Interaction in UseCaseView with the use case nominated in InteractionModel.describes_use_case as one of its interactors, such that sender and receiver in the event specification equate (not necessarily respectively) to the other end of this interaction and the subject_module attribute of UserCentredRequirementsStructure.

**Communication Event Invariant**

\[ \text{self.allInstances->forall(c | self.interactionModel.interactionView.userCentredRequirementsStructure.useCaseView.ucv_use_case->exists(u | self.interactionModel.interactionView.userCentredRequirementsStructure.useCaseView.ucv_interaction->exists(i | c.interactionModel.describes_use_case = u.use_case_name and (i.interactor_2 = u and ((c.tsn_communication_event.tsn_sender_node.instance_name = i.interactor_1.actor_name and c.tsn_communication_event.tsn_receiver_node.instance_name = c.interactionModel.interactionView.userCentredRequirementsStructure.subject_module) or (c.tsn_communication_event.tsn_receiver_node.instance_name = i.interactor_1.actor_name and c.tsn_communication_event.tsn_sender_node.instance_name = c.interactionModel.interactionView.userCentredRequirementsStructure.subject_module))) or (i.interactor_1 = u and ((c.tsn_communication_event.tsn_sender_node.instance_name = i.interactor_2.actor_name and c.tsn_communication_event.tsn_receiver_node.instance_name = c.interactionModel.interactionView.userCentredRequirementsStructure.subject_module) or (c.tsn_communication_event.tsn_receiver_node.instance_name = i.interactor_2.actor_name and c.tsn_communication_event.tsn_sender_node.instance_name = c.interactionModel.interactionView.userCentredRequirementsStructure.subject_module)))))} \]

3. Constraint (over textual perspective) to ensure PDS consistency of Communication Events. That is, the corresponding PDS sender module produces an external flow (corresponding to the event message) which is consumed by a function of the corresponding PDS receiver module; both sender and receiver modules interface with one another. Depending on how advanced the development is, this may be through some as yet unspecified means (using InterfacesTo), or via a mutual interface (using Interface), or else via their own respective interfaces adjoined by a Connection.

**Communication Event Invariant**

\[ \text{self.allInstances->forall(c | self.tsn_communication_event.communication_description.tsn_sender_node.build_Element->exists(be1 | self.tsn_communication_event.communication_description.tsn_receiver_node.build_Element->exists(be2 | self.tsn_communication_event.communication_description.tsn_message_node.))} \]
Structuring Development Artifacts

4. Constraint (again expressed over the textual perspective) to ensure that actions are declared as functions in the PDS. Furthermore, each function describes the behaviour of a transaction corresponding to the use case nominated in InteractionModel.describes_use_case.

**InternalActionEvent invariant**

self.allInstances->forall(a |
  self.tsn_action_event.action_description.tsn_sdr_rcr_node.bEelementAEO.build_element->exists(be | 
  self.tsn_action_event.action_description.tsn_action_node.bEelementAEO.build_element->exists(f | 
  self.tsn_action_event.action_description.tsn_sdr_rcr_node.bEelementAEO.build_element->exists(t | 
  a.tsn_action_event.action_description.tsn_sdr_rcr_node.instance_name = be.module_name and 
  a.tsn_action_event.action_description.tsn_action_node.action_name = f.function_name and 
  a.interactionModel.describes_use_case = t.transaction_name and 
  be.behavesAccordingTo.target->includes(t) and 
  f.usesFunction.target->includes(f)))))

5. Constraint (also expressed over textual perspective) to ensure timers are defined as sub-modules of the target system in the PDS.

**TimingEvent invariant**

self.allInstances->forall(t |
  self.tsn_timing_event.timing_description.instance.bEelementAEO.build_element->exists(be1 | 
  self.tsn_timing_event.timing_description.tsn_timer_instance_node.bEelementAEO.build_element->exists(be2 | 
  t.tsn_timing_event.timing_description.instance.instance_name = be1.module_name and 
  t.tsn_timing_event.timing_description.tsn_timer_instance_node.timer_name = be2.module_name and 
  be1.hasSubmodule.target->includes(be2))))

6. Rules to derive Message Sequence Chart event perspectives from textual representations.

**CommunicationEvent**

self.allInstances->forall(c |
  self.msc_communication_event->exists(m | 
  c.tsn_communication_event->notEmpty and 
  c.msc_communication_event->includes(m)))) implies 
  m.msc_sender_instance->includes
User-Centred Requirements Structure

(c.tsn_communication_event.communication_description.tsn_sender_node) and
m.msc_receiver_instance->includes
(c.tsn_communication_event.communication_description.tsn_receiver_node) and
m.msc_message->includes(c.tsn_communication_event.communication_description.tsn_message_node)

InternalActionEvent
self.allInstances->forall(a I
self.msc_action_event->exists(m l
a.tsn_action_event->notEmpty
and
a.msc_action_event->includes(m)))
implies
m.msc_sdr_rcr_instance->includes
(a.tsn_action_event.action_description.tsn_sdr_rcr_node) and
m.msc_system_action->includes
(a.tsn_action_event.action_description.tsn_action_node)

TimingEvent
self.allInstances->forall(t l
self.msc_timing_event->exists(m l
t.tsn_timing_event->notEmpty
and
t.msc_timing_event->includes(m)))
implies
m.msc_host_on_timeout_instance->union(m.msc_timer_set_instance->union(m.msc_timer_reset_instance))
->includes
(t.tsn_timing_event.timing_description.tsn_host_on_timeout_node->union
(t.tsn_timing_event.timing_description.tsn_timer_set_node->union
(t.tsn_timing_event.timing_description.tsn_timer_reset_node))) and
m.msc_timer_instance->includes(t.tsn_timing_event.timing_description.tsn_timer_instance_node) and
m.msc_timer_duration->includes(t.tsn_timing_event.timing_description.tsn_timer_duration)

ii. Well-Formedness
1. Constraint enforcing uniqueness of Instance primitives throughout an InteractionView (Appendix A,
Part 2 includes similar constraints over Message, Action and Timer).

InteractionView Invariant
self.allInstances->forall(v l
self.interaction_model.inm_instance->forall(i1, i2 l
not(
v.interaction_model.inm_instance->includes(i1) and v.interaction_model.inm_instance->includes(i2) and i1 <> i2
and
i1.instance_name = i2.instance_name)))

2. Constraint ensuring that each InteractionModel within an InteractionView describes a different UseCase.

InteractionView Invariant
self.allInstances->forall(v l
self.interaction_model->forall(m1, m2 l
not (v.interaction_model->includes(m1) and v.interaction_model->includes(m2) and
m1 <> m2 and m1.describes_use_case = m2.describes_use_case)))

3. Constraint ensuring that 'included scenarios' are paths through different use cases (Appendix A,
Part 2 includes a similar constraint for 'extended-by scenarios').
Structuring Development Artifacts

**Interaction Model Invariant**

\[
\text{self.allInstances->forall}(m_1, m_2 \mid \\
\text{self.inm_scenario->forall}(s_1, s_2 \mid \\
\text{not}(m_1.\text{inm_scenario->includes}(s_1) \text{ and } m_2.\text{inm_scenario->includes}(s_2) \text{ and } \\
s_1.\text{includes_scenario->includes}(s_2) \text{ and } m_1 = m_2))
\]

4. Constraint to ensure that for two Interaction Models \((m_1, m_2)\), where the UseCaseView contains an «includes» association in which the use case described in \(m_1\) is the base of this association and the use case described in \(m_2\) is the included part, then at least one Scenario forming \(m_2\) should appear among the included scenarios of any Scenario in \(m_1\). Readers are referred to Appendix A, Part 2 for a similar constraint over «extends» associations.

**Interaction Model Invariant**

\[
\text{self.allInstances->forall}(m_1, m_2 \mid \\
\text{self.interactionView.userCentredRequirementsStructure.useCaseView.ucv_includes->} \\
\text{not exists}(i \mid \\
m_1.\text{describes_use_case} = i.\text{includes_base.use_case_name} \text{ and } \\
m_2.\text{describes_use_case} = i.\text{includes_include.use_case_name} \text{ and } \\
(m_2.\text{inm_scenario->intersection}(m_1.\text{inm_scenario.includes_scenario})->\text{isEmpty})))
\]

5. ‘Style rule’ that the first message in a Scenario is a Communication Event. This can be enforced by restricting sequence numbers.

**Event**

\[
\text{self.allInstances->forall}(e \mid \\
\text{self.sequence_no->forall}(s \mid \\
\text{self.included_seq_no->forall}(i \mid \\
\text{self.extension_seq_no->forall}(x \mid \\
\text{not}((e.\text{sequence_no->includes}(s) \text{ and } s.\text{sequence_no} = 1) \text{ or } \\
(e.\text{included_seq_no->includes}(i) \text{ and } i.\text{sequence_no} = 1) \text{ or } \\
(e.\text{extension_seq_no->includes}(x) \text{ and } x.\text{sequence_no} = 1)) \text{ and } e.\text{oclType <> CommunicationEvent}))))
\]

6. ‘Style rule’ that the first message in a Scenario must be from an (actor) Instance to the target system and must not be an included or extension event; note the rule applies only to scenarios of use cases not designated as a sub-case.

**Scenario**

\[
\text{self.allInstances->forall}(s \mid \\
\text{self.scenario_event->union}(\text{self.included_event->union}(\text{self.extension_event})) \\
\rightarrowforall(c \mid \\
\text{self.interactionModel.interactionView.userCentredRequirementsStructure.useCaseView.ucv_use_case} \\
\rightarrowforall(u \mid \\
\text{self.scenario_event.sequence_no->union}(\text{self.included_event.included_seq_no->union} \\
(\text{self.extension_event.extension_seq_no}))->\text{intersection} \\
\text{self.scn_seq_no->union}(\text{self.scn_included_seq_no->union}(\text{self.scn_extension_seq_no}))->\forall(n \mid \\
\text{not} (s.\text{interactionModel.interactionView.userCentredRequirementsStructure.useCaseView.ucv_use_case} \\
\rightarrow\text{includes}(u) \text{ and } \\
u.\text{use_case_name} = s.\text{interactionModel.describes_use_case} \text{ and } \\
u.\text{sub_case} = \text{False} \text{ and } \\
((\text{c.included_event->includes}(c) \text{ and } c.\text{included_seq_no->intersection}(s.\text{scn_included_seq_no})->\text{includes}(n) \text{ and } \\
\text{forall}(n \mid \\
\text{not}(n.\text{include}(s)))))))
\]

129
4.3.3.4.3 O-Telos Base Classes for Interaction View Meta-model

We implement base classes for the Interaction View (shown in figures 4.12 through 4.17) as follows:

**Definition of Interaction Model**

**Constituents**

Scenario in StructureElement, SimpleClass

isA AerospaceEngineeringObject with

has_property

scenario_title : String;

is_exception : Bool;

scn_seq_no : SequenceNumber;

scn_included_seq_no : SequenceNumber;

scn_extension_seq_no : SequenceNumber

has_part

scenario_event : Event;

included_event : Event;

extension_event : Event;

includes_scenario : Scenario;

extended_by_scenario : Scenario;

event_group : EventGroup;

included_event_group : EventGroup;

extension_event_group : EventGroup;

tsn_viewpoint : TsnScenarioViewpoint;

msc_viewpoint : MscScenarioViewpoint

has_transitive_part

transitive_includes_scenario : Scenario;

transitive_extended_by_scenario : Scenario

end

TsnScenarioViewpoint in StructureElement, SimpleClass

isA AerospaceEngineeringObject with has_part

tsv_tsn_comm : TsnCommunication;

tsv_tsn_act : TsnAction;

tsv_tsn_tim : TsnTiming

end

MscScenarioViewpoint in StructureElement, SimpleClass

isA AerospaceEngineeringObject with has_part

msv_msc_comm : MscCommunication;

msv_msc_act : MscAction;

msv_msc_tim : MscTiming

end

Event in StructureElement, SimpleClass

isA AerospaceEngineeringObject with

has_property

sequence_no : SequenceNumber;

included_seq_no : SequenceNumber;

extension_seq_no : SequenceNumber;

lwr_bd : LowerBound;

upr_bd : UpperBound

has_part

follows_from : Event

transitive_follows_from : Event

constraint

abstractEvt: $forall t/Token

s/SimpleClass

(t in s) == not (t in Event)

end

CommunicationEvent in StructureElement, SimpleClass

isA Event with

has_property

interaction_type : InteractionType

has_part

tsn_communication_event : TsnCommunication;

msc_communication_event : MscCommunication

end

TsnCommunication in StructureElement, SimpleClass

isA AerospaceEngineeringObject with

has_structure

communication_description : ScenarioEventNaturalLanguageStructure

end

MscCommunication in StructureElement, SimpleClass

isA AerospaceEngineeringObject with

has_property

link_name : String;

synchronisation : Synchronisation;

frequency : Frequency;

delayed : Bool

has_part

msc_sender_instance : Instance;

msc_receiver_instance : Instance;

msc_message : Message

end

MessageDescription in StructureElement, SimpleClass

isA AerospaceEngineeringObject with

has_property

msg_parameter : String

end

InternalActionEvent in StructureElement, SimpleClass

isA Event with

has_part

tsn_action_event : TsnAction;

msc_action_event : MscAction

end

TsnAction in StructureElement, SimpleClass

isA AerospaceEngineeringObject with

has_structure

action_description : ScenarioEventNaturalLanguageStructure

end

MscAction in StructureElement, SimpleClass

isA AerospaceEngineeringObject with

has_part

msc_sdr_rrc_instance : Instance;

msc_system_action : Action

end

ActionDescription in StructureElement, SimpleClass

isA AerospaceEngineeringObject with

has_property

action_parameter : String
Structuring Development Artifacts

TimingEvent in StructureElement, SimpleClass isA Event with
  has_part
    tsn_timing_event : TsnTiming;
    msc_timing_event : MscTiming
end

TsnTiming in StructureElement, SimpleClass isA AerospaceEngineeringObject with
  has_structure
    timing_description : ScenarioEventNaturalLanguageStructure
end

MscTiming in StructureElement, SimpleClass isA AerospaceEngineeringObject with
  has-part
    msc_timer_set_instance : Instance;
    msc_timer_reset_instance : Instance;
    msc_host_on_timeout_instance : Instance;
    msc_timer_instance : Timer;
    msc_timer_duration : TimerDuration
end

EventGroup in StructureElement, SimpleClass isA AerospaceEngineeringObject with
  has_property
    grp_lb : LowerBound;
    grp_ub : UpperBound
end

SequenceNumber in ArtifactProperty, SimpleClass with
  described_by
    sequence_no : Integer
end

LowerBound in ArtifactProperty, SimpleClass with
  described_by
    lower_bound : String
end

UpperBound in ArtifactProperty, SimpleClass with
  described_by
    upper_bound : String
end

InteractionType in ArtifactProperty, SimpleClass isA String with
  constraint
    enum_Int: forall i/InteractionType (i = "SR") or (i = "SP") or (i = "IR") or (i = "IP")
end

Synchronisation in ArtifactProperty, SimpleClass isA String with
  constraint
    enum_Syn: forall s/Synchronisation (s = "sim") or (s = "syn") or (s = "bal") or (s = "tim") or (s = "asy")
end

Frequency in ArtifactProperty, SimpleClass isA String with
  constraint
    enum_Frq: forall f/Frequency(f = "periodic") or (f = "aperiodic")
end

Definition of Interaction Model
InteractionModel in DevelopmentStructure,
  SimpleClass isA AerospaceEngineeringObject with
    has_property
      model_name : String;
      describes_use_case : String;
      imn_date : Date
    has_structure
      imn_comments : MatraNaturalLanguageStructure
      has_element
        imm_scenario : Scenario;
        imn_scenario : Scenario;
        imn_extension_scenario : Scenario;
        imn_event : Event;
        imm_communication_event : CommunicationEvent;
        imn_action_event : InternalActionEvent;
        imm_timing_event : TimingEvent;
        imn_tsn_viewpoint : TsnScenarioViewpoint;
        imn_msc_viewpoint : MscScenarioViewpoint;
        imm_msc_communication : MscCommunication;
        imm_msc_action : MscAction;
        imm_msc_timing : MscTiming;
        imn_msc_communication : MscCommunication;
        imn_msc_action : MscAction;
        imn_msc_timing : MscTiming;
        imn_msc_communication : MscCommunication;
end

Definition of Interaction View
InteractionView in DevelopmentStructure, SimpleClass isA AerospaceEngineeringObject with
  has_property
    inv_date : Date
  has_structure
    inv_comments : MatraNaturalLanguageStructure;
    interaction_nodel : InteractionModel
end

Definition of SENLS Constituents
Message in StructureElement, SimpleClass isA MatraNLSAtomicNode with
  has_property
    message_name : String
end

Action in StructureElement, SimpleClass isA MatraNLSAtomicNode with
  has_property
    action_name : String
end

--- Message Description specified as part of Interaction Model
Action in StructureElement, SimpleClass isA MatraNLSAtomicNode with
  has_part
    tsn_act parameter : ActionDescription;
    msc_act parameter : ActionDescription
end
User-Centred Requirements Structure

--- Action Description specified as part of Interaction Model

TimerDuration in StructureElement, SimpleClass isA MatraNLSAtomicNode with has_property
duration : String
end

Timer in StructureElement, SimpleClass isA MatraNLSAtomicNode with has_property
timer_name : String
has_part
  Timer_duration : TimerDuration
end

Instance in StructureElement, SimpleClass isA MatraNLSAtomicNode with has_property
instance_name : String
end

Definition of SENLS

ScenarioEventNaturalLanguageStructure in TraceabilityStructure, SimpleClass isA MatraNaturalLanguageStructure with has_element
tsn_message_node : Message;
tsn_action_node : Action;
tsn_timer_duration : TimerDuration;
tsn_timer_instance_node : Timer;
tsn_sender_node : Instance;
tsn_receiver_node : Instance;
tsn_sdr_rcr_node : Instance;
tsn_host_on_timeout_node : Instance;
tsn_timer_set_node : Instance;
tsn_timer_reset_node : Instance
end

4.3.4 Relationship to the Traceability Dimensions

While the focus of this thesis is on developing meta-models for well-defined and flexible notations, we
do nevertheless introduce a few examples of potential links between these models in order to
demonstrate the AerospaceEngineeringAssociation concept from Chapter Three.

The fact that UCRS integrates Use Case Models, Scenarios and Message Sequence Charts suggests a
need for traceability within the structure itself. Hence in Chapter Six we illustrate some potential (intra-
micro) horizontal associations between these notations using associations proposed in subsection
3.3.6.3.2. For instance, we can assert that a particular scenario constitutes a ‘path-through’ a use case
(in the sense of elaborating the use case description), and that the textual view of a scenario is
‘illustrated-by’ its message sequence chart equivalent.

Discussions with aerospace practitioners indicates that Use Case Modelling is often used in conjunction
with Real-Time Networks, a design methodology considered in the next subsection. This in turn implies
that (intra-micro) vertical traceability should exist between elements of the two notations, a point to
which we return in subsection 4.4.4.

Note the need for a thorough investigation into inter/intra, macro/micro and horizontal/vertical
associations that can exist between all artifact types used in avionics systems engineering is identified as
a future work item in 7.4.3.

4.3.5 Summary

The modelling of interactions between external entities and a target system is an important facet of
requirements engineering in the aerospace industry. Use Case Models, together with Scenarios and their
graphical representation as Message Sequence Charts afford established means of accomplishing this.
Accordingly, they were chosen as representative requirements notations for the MATra traceability
framework.

From discussions with practitioners and examination of actual commercial specifications, we developed
a novel structure to allow integration of these notations. Rules and well-formedness constraints provide
a light-weight semantics for the structure, allowing verification and traceability. In particular, existing literature on Scenarios allowed us to develop a semi-formal foundation (featuring a specialised MNLS from section 4.2) to what is the least formal of the three notations, but which retains the usability of prose representations.

Chapter Six (section 6.2) demonstrates application of the User Centred Requirements Structure using a commercial specification for a sub-system of the Hawk lead-in fighter/trainer aircraft; a full evaluation of UCRS appears in Chapter Seven.
4.4 Real-Time Network Specification Language Structure (Graphical)

4.4.1 Introduction

In this section we introduce a structure capturing the graphical syntax and 'light-weight' formal semantics (i.e., with sufficient rigour for traceability purposes) for a Real-Time Network Specification Language (RTN-SL) currently being developed by the MBDA-UK business unit of BAE SYSTEMS.

4.4.2 Motivation

In Chapter One, we alluded to scale and complexity as common characteristics of avionics systems. A proven method for dealing with any complex system is to partition it into smaller independently operating subsystems which only interact with one another and with the system environment through explicitly defined communication connections. For computer based systems, this approach is known as the Real-Time Network (RTN) architecture in which concurrent processing components exchange information and synchronise through shared data in the connections\(^{16}\).

MASCOT (Modular Approach to Software, Construction, Operation and Test) is a design methodology based on the Real-Time Network concept (Simpson, 1986). It comprises a design language and graphical notation, together with a process for design derivation based on structural decomposition. This involves identification of computational components of a system (its subsystems and processes) and the interactions between these components (i.e., data-flow paths), as well as protocols that characterise these interactions. MASCOT-3 is advocated for the design of large concurrent or distributed, real-time embedded software systems and has been used extensively throughout the defence industry.

DORIS (Simpson, 1994), the Data-Oriented Requirements Implementation Scheme is a variant of MASCOT-3 developed by MBDA-UK. The main difference is that DORIS distinguishes three levels of design abstraction, such that application network designs and execution network designs are specified in addition to the functional design. The former define the logical architecture of a system, whilst the latter extend the application network to include all additional components required to support distribution across a particular processor network. This enables flexible re-mapping of a design to the hardware platform as the hardware platform evolves (Paynter et al. 2000). DORIS also supports a wider-range of synchronous and asynchronous communication protocols appropriate to both shared-memory and message passing implementations (Simpson, 2000a-c).

In this thesis, we focus on support for tracing designs represented using RTN-SL, a specification and design language (intended for integration into MASCOT-3 and DORIS) currently being developed by MBDA-UK for defining the behaviour of Real-Time Networks (Paynter, 2000). RTN-SL features both a formal textual and graphical syntax\(^{17}\); in concentrating on the latter\(^{18}\) we note that the textual syntax

---

\(^{16}\) For a comparison of methods for real-time systems development, readers are referred to Hull et al. (1991).

\(^{17}\) The graphical syntax captures a subset of the textual syntax.

\(^{18}\) This is due to the fact that whilst the graphical syntax is stable, the formal textual syntax is continuing to evolve.
has been influenced by Spark-Ada (Barnes, 1996) and that the same approach used in subsection 4.5 to
develop a traceability meta-model for this language can be similarly applied to produce a comparable
structure for the RTN-SL textual syntax (we briefly return to this issue in 4.4.3). The following
subsection provides an overview of the RTN-SL and its graphical notation in particular.

4.4.2.1 RTN-SL Overview

RTN-SL is used to define flat Real-Time Networks, including the real-time and functional behaviour of
their activities. A flat RTN comprises a set of activities (single-threaded processes) connected into a
network by ports that interface with communication paths. Between each activity is an
intercommunication data area (IDA) which defines the interaction protocol used on that path. Activities
may not be directly connected together, but instead must communicate via an IDA (Paynter, 2000).

RTN-SL supports five protocols, namely pool, channel, signal, stimulus and dataless channel; the three
basic protocols are the pool (similar to a shared variable), channel (a bounded buffer) and signal (a
one-place over-writing buffer). These impose different synchronisation constraints on the reader and
writer of the protocol depending on whether they allow data to be destroyed when it is read or written
(see table 4.2 for a summary). The other two RTN-SL protocols, namely stimulus and dataless channel
are variants of signal and channel respectively; they differ in that both allow communication of void
(null) data.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Reader Constraints</th>
<th>Writer Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pool</td>
<td>Non-destructive</td>
<td>Destructive</td>
</tr>
<tr>
<td></td>
<td>reading of data</td>
<td>reading of data</td>
</tr>
<tr>
<td></td>
<td>Reader cannot be</td>
<td>Reader can be</td>
</tr>
<tr>
<td></td>
<td>delayed</td>
<td>delayed</td>
</tr>
<tr>
<td>Channel</td>
<td>Non-destructive</td>
<td>Destructive</td>
</tr>
<tr>
<td></td>
<td>writing of data</td>
<td>writing of data</td>
</tr>
<tr>
<td></td>
<td>Writer cannot be</td>
<td>Writer cannot be</td>
</tr>
<tr>
<td></td>
<td>delayed</td>
<td>delayed</td>
</tr>
<tr>
<td>Signal (Stimulus inset)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel (Dataless Channel inset)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 4.2 - 'IDA Communication Protocols'**

Abstract data types (ADTs) may be defined to support definition of a flat Real-Time Network. These
are necessary when a data type must be visible in more than one activity and/or IDA, where one activity
communicates data to another which is not a built-in type (Paynter, 2000).

Conceptually, the RTN-SL includes a sub-language known as Activity Description Language (ADL) for
defining the behaviour of activities. In turn, the ADL includes a timed state-machine notation sub-
language termed the Activity State-Machine (ASM)\textsuperscript{19} which is used to define the structure and timing constraints of an activity's algorithm.

ASM distinguishes between static and dynamic states; static states model potential blocking-points of an algorithm where an activity is either attempting to communicate using a synchronous protocol, or is engaged in a timed delay. Transitions from a static state are labelled either with the event indicating that the communication may continue or finish, or with the lower and upper bonds of the time delay (Paynter, 2000).

Dynamic states model an activity's computation, each one encapsulating some non-reactive functionality. They have a best-case execution time (BCET) bound, a worst-case execution time (WCET) bound and worst-case response time (WCRT) bound; an optional worst-case response time on read may also be recorded. An activity is normally required to exit a state within these times\textsuperscript{20}. Transitions from a dynamic state are labelled with conditions over the local activity's state which are evaluated when a dynamic state terminates (Paynter, 2000). A composite dynamic state comprises several dynamic states whose operations may execute in any order.

4.4.2.2 RTN-SL Graphical Syntax

To aid comprehension, RTN-SL has a graphical syntax which we term RTN-SLg and which builds upon the graphical syntax of MASCOT (Simpson, 1986). Activities are represented as circles; IDAs by the appropriate symbol from table 4.2 (which is enclosed within a rectangle where the IDA is defined and hence named); ADTs by diamonds; communication paths by solid arcs with arrows indicating the direction of data-flow; and ADT imports by dotted lines with arrows pointing towards the importing unit. Ports are represented by solid circles on the perimeter of activities and the state machine drawn within the activity circle.

Static ASM states are represented by an ellipse and dynamic states as a rectangle. Where an input or output is associated with a dynamic state, then the appropriate port name is shown in the top left hand and right hand corners of the rectangle respectively. Labels in the bottom left and right hand corners indicate BCET, WCET and WCRT; the optional WCRT on read is shown in a box next to the input port name. The dynamic state graphical syntax is summarised in figure 4.18.

<table>
<thead>
<tr>
<th>Input Port Name</th>
<th>Optional WCRT on Read</th>
<th>Output Port Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic State Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCET (Lower Bound)</td>
<td>WCET</td>
<td>WCRT (Upper Bound)</td>
</tr>
</tbody>
</table>

Figure 4.18 - ‘ASM Dynamic State Graphical Syntax’

\textsuperscript{19} Strictly, the ADL (including ASM) and RTN-SL are a single language.

\textsuperscript{20} A discussion on the conditions necessary for this to occur is beyond the scope of this thesis.
Composite dynamic states are depicted by partitioning a dynamic state using dotted lines. Each constituent state may have its own input and output port, although one set of time bounds applies to the whole state. An example composite comprising three dynamic sub-states is shown in figure 4.19.

![ASM Graphical Syntax for Composite Dynamic States](image)

Figure 4.19 - ‘ASM Graphical Syntax for Composite Dynamic States’

Transitions between states are depicted by directed arcs pointing from the source to the target state. Conditions, events and time-outs are shown as textual labels adjacent to the appropriate transition. The initial state is shown by being the target of a transition with no source.

![Example RTN-SLg Specification](image)

Figure 4.20 - ‘Example RTN-SLg Specification’ (source Paynter, 2000)

Figure 4.20 illustrates the main constituents of an RTN-SLg specification as described above. Activity A2 includes three static states, A (also the initial state), C and E, together with two dynamic states, B and D. When the activity is in state A and a void data value is present in the stimulus connected to port 2 (P2), then event $S_{P2}$ occurs and the activity moves to state B which reads from the pool on port 3 (P3) within a finish time of $U_1$. The computation (not shown) associated with state B is executed and takes between $L_1$ and $U_2$ time units, although interruptions from the scheduling of other activities may prolong this to $U_3$; conditions $C_1$ and $C_2$ are then evaluated. A transition with a true condition is taken and either state C or D is entered. Assuming state C, the activity is de-scheduled for between $t_1$ and $t_2$.

---

21 Computations are represented solely by the RTN-SL textual syntax.
time units whereupon state D is entered. The algorithm associated with state D (again not shown) is executed between its time bounds (as for state B) and a value written to the channel linked to port 4 (P4). On evaluation of condition C3 to true, the activity enters and waits in state E until the Ws event occurs\(^\text{22}\) whereupon it returns to the initial state (A). Activity A1 executes concurrently with A2, communicating with it via IDA ida1. ADT adt1 is used in the definition of A1, A2 and ida1; ADT adt2 is used in the definition of adt3, which in turn is used in the definition of A2.

### 4.4.3 Tracing Real-Time Network Specifications in MATrA: An RTN-SLg Model

In this subsection, we introduce a meta-model to support traceability of designs specified using the RTN-SLg notation, thereby allowing integration of Real-Time Networks into the MATrA framework.

The fact that RTN-SLg has a corresponding textual language leads to two options for constructing the meta-model:

1. Employ an identical process to that used for the Use Case and Message Sequence Chart notations in subsection 4.3; i.e., examine components of the graphical notation, along with any well-formedness constraints restricting how these components are connected together; the resultant model is then verified against the textual language subset. Note that if like its textual counterpart the RTN-SLg syntax was also formally defined, perhaps using node-labelled graphs (cf. Paynter 1995) or a similarly rich language, then production of the meta-model would be more straightforward and the result more rigorous.

2. Examine the textual language subset that corresponds to the graphical notation; this would involve constructing schemas for each syntactic language element (BNF category) of the subset. It also assumes a function exists based on a mapping between the graphical and textual syntax that translates an RTN-SLg specification to an RTN-SL textual specification; tool2matra would then operate over the latter.

Application of both these options should result in two isomorphic models that while potentially different in their labelling of classes and associations, share the same structure. In this subsection we pursue option one, with option two applied after developing an approach for textual languages in subsection 4.5.

#### 4.4.3.1 Concepts

In developing a meta-model to support traceability of RTN-SLg specifications, we have sought to capture the complete graphical syntax described in subsection 4.4.2.2. The model therefore contains representations of ADTs, IDAs, Ports and Activities, together with ASMs comprising static, dynamic and composite dynamic states.

There are numerous well-formedness constraints needed to ensure an RTN-SLg specification is

\(^{22}\) This will be instantaneous if there is a space in the channel.
Structuring Development Artifacts


4.4.3.2 RTN-SLg Meta-model Definitions

In this subsection, we introduce the RTN-SLg meta-model (4.4.3.2.1) which is again specified using the Class Diagram view of UML. We also introduce constraints over the model expressed in OCL (4.4.3.2.2), together with an implementation of its base classes in the O-Telos language (4.4.3.2.3).

4.4.3.2.1 RTN-SLg Meta-model

Figures 4.21 (elements) and 4.22 (associations) depict the UML meta-model for RTN-SLg. Its core class (RTN-SLg), with subject module (subject_module) and other configuration attributes (plus MNLS commenting facility) is an instantiation of the DevelopmentStructure meta-class. RTN-SLg is further defined as an aggregation of StructureElement meta-classes representing both RTN and ASM primitives.

Firstly, we consider those elements necessary to represent real-time networks, namely activity (Activity), abstract data type (Adt), IDA (Ida) and port (Port) entities, together with definitions (ConnectionDef) of network paths linking IDAs and activity ports.

Ida and Activity are both subtypes of the abstract component class (Component) reflecting the fact that RTN aspects of our model are grounded on work towards the formal characterisation of MASCOT in Paynter (1995). This influence is further evident in the (derived) reflexive associations on Component to record predecessor (predecessor) and successor (successor), together with further associations capturing their respective transitive closure elements (all_predecessors and all_successors). Our use of the oc1Type operation also mirrors that of the function Node_Type in Paynter (ibid.).

Activity, Port, Ida and Adt carry (unique) name attributes for the purpose of meaningful identification and cross-verification with Product Data Synthesis elements; the same label (name) is used across all of these classes as per the RTN-SL textual syntax. Ida also features a protocol kind (kind) attribute and an aggregation defining imported Adts (with_adt). Adt itself includes reflexive associations to identify imported ‘with’ abstract data types (also labelled with_adt), along with the transitive closure of imported ‘with’ (with_all). A ConnectionDef is formed from an aggregation of a Port and an Ida, the latter over an exclusive-or association denoting the direction of data flow from (from_ida) xor to (to_ida) an Ida.

The Activity class meanwhile is an aggregation of ports, “with-ed” Adts (again, with_adt) and a state machine (StateMachine) which in turn has a number of state definitions (identified through the rolename state_def), with subtypes static state (StaticState), dynamic state (DynamicState) and composite dynamic

23 MASCOT designs are defined as directed-graphs: MASCOT_DESIGNS:: NODES: COMPONENTS; ARCS: COMPONENTS x COMPONENTS.
24 Paynter (ibid.) uses the functions Predecessors : COMPONENTS -\rightarrow COMPONENTS-set and Successors: COMPONENTS -\rightarrow COMPONENTS-set, which operate over MASCOT DESIGNS and return the set of all predecessors and set of all successors respectively; appropriate rules in subsection 4.4.3.2.2 (i) derive corresponding elements for our meta-model.
25 Node_Type : COMPONENTS -\rightarrow (IDA, ACTIVITY} is a (total) function which returns the type of a node (i.e., IDA or Activity); this is similar to the oc1Type operation (from subsection 2.2.2.2.9) which evaluates to the type of an object.
26 See OCL constraints in parts iii and iv(l) of subsection 4.4.3.2.2.
Figure 4.21 - 'RTN-SLg Structure : Elements'
Figure 4.22 - 'RTN-SLg Structure : Associations'
(CompositeDynamic). The state class (State) itself is abstract, with a composition aggregation to initial status (InitialStatus) containing an attribute initial (initial) of type Bool. Prima facie, InitialStatus can be encapsulated as an attribute of State; however, we promote it to a class to enable its suppression on dynamic states that form a CompositeDynamic\(^{27}\).

A StaticState has a unique name (name) attribute with a static transition definition (rolename static\_transition_def) defined by the class StaticTransition which goes to (goes\_to) a target State. Situations in which the transition is enabled may be either events (Event) of type (event_type) read, write or stim to a particular Port, exclusive-or, a timing definition (TimingTransitionDef) with upper and lower bounds (denoted by the attribute lb\_ub).

A DynamicState also has a unique name (name) attribute, together with a dynamic transition definition (rolename dynamic\_transition_def) defined by the class DynamicTransition which again goes to (goes\_to) a target State on the specified condition holding (denoted by the on attribute). DynamicState is also an aggregation of timing (Timing) and local operation definitions (LocalOpDef). The former is described in terms of timing attributes for best-case execution time (bcet) bound, worst-case execution time (wcet) bound and worst-case response time (wret) bound, whilst the latter comprises definitions of the mapping of input (ReadsFromClause) and output (WritesToClause) parameters from and to the appropriate ports. The WCRTonRd class (with attribute finish) aggregated to ReadsFromClause (using the rolename by) supports optional definition of a worst case read time bound.

Finally, a CompositeDynamic state is described in terms of (two or more) dynamic states, together with dynamic transitions and a timing definition; again a constraint\(^{28}\) ensures suppression of these characteristics for individual dynamic states such that their definitions apply to the whole CompositeDynamic.

### 4.4.3.2.2 OCL Constraints

In this subsection, we express the following types of constraint over the RTN-SLg meta-model:

- Well-formedness of Real-Time Network elements;
- Well-formedness of Activity State Machine elements;
- Verification of RTN-SLg elements against the Product Data Synthesis;
- Other structural and consistency constraints attributable to modelling decisions.

#### I. Well-formed Real-Time Networks

The constraints below apply to Real-Time Network elements of the RTN-SLg meta-model; i.e., activities, ports and IDAs. They are based on rules for a well-formed MASCOT subset (Paynter, 1995) which (as indicated in 4.4.3.2.1) views RTNs as labelled directed graphs and describes relationships among activities and IDAs (i.e., nodes) in terms of their predecessors and successors.

\(^{27}\) See OCL constraint iv(2) in subsection 4.4.3.2.2.

\(^{28}\) See OCL constraint iv(3) in subsection 4.4.3.2.2.
Structuring Development Artifacts

- Immediate predecessor and successor are populated using the following deductive axioms:-

  \[
  \text{Activity} \\
  \text{self.allInstances->forall}(a \mid \\
  \text{self.port.connectionDef->forall}(c \mid \\
  \text{c.port.activity->includes}(a) \text{ and } \text{c.from_ida->size=}1)) \\
  \text{implies} \\
  \text{a.predecessor->includes}(c.\text{from_ida}) \\
  \]

  \[
  \text{Activity} \\
  \text{self.allInstances->forall}(a \mid \\
  \text{self.port.connectionDef->forall}(c \mid \\
  \text{c.port.activity->includes}(a) \text{ and } \text{c.to_ida->size=}1)) \\
  \text{implies} \\
  \text{a.successor->includes}(c.\text{to_ida}) \\
  \]

  \[
  \text{Ida} \\
  \text{self.allInstances->forall}(i \mid \\
  \text{self.connectionDef->forall}(c \mid \\
  \text{c.to_ida->includes}(i))) \\
  \text{implies} \\
  \text{i.predecessor->includes}(c.\text{port.activity}) \\
  \]

  \[
  \text{Ida} \\
  \text{self.allInstances->forall}(i \mid \\
  \text{self.connectionDef->forall}(c \mid \\
  \text{c.from_ida->includes}(i))) \\
  \text{implies} \\
  \text{i.successor->includes}(c.\text{port.activity}) \\
  \]

- Similarly, the following rules - this time expressed over the abstract Component class (and by implication of using the allInstances operation, its Activity and Ida subtypes) - determine all predecessors and all successors. These will be used in specifying several of the invariants that follow:-

  \[
  \text{Component} \\
  \text{self.allInstances->forall}(c_1, c_2 \mid \\
  \text{c_1.predecessor->includes}(c_2) \text{ or } \\
  \text{self.allInstances->exists}(c' \mid \\
  \text{c_1.predecessor->includes}(c') \text{ and } \text{c'.all_predecessors->includes}(c_2)) \\
  \text{implies} \\
  \text{c_1.all_predecessors->includes}(c_2) \\
  \]

  \[
  \text{Component} \\
  \text{self.allInstances->forall}(c_1, c_2 \mid \\
  \text{c_1.successor->includes}(c_2) \text{ or } \\
  \text{self.allInstances->exists}(c' \mid \\
  \text{c_1.successor->includes}(c') \text{ and } \text{c'.all_successors->includes}(c_2)) \\
  \text{implies} \\
  \text{c_1.all_successors->includes}(c_2) \\
  \]

Well-formedness Constraints for Real-Time Networks

We are now in a position to define a subset of well-formedness constraints applicable to valid Real-Time Networks:-
1. Each Component is connected to another Component. This constraint is captured by the RTN-SLg class diagram and its multiplicity semantics.

2. Every connection path must connect to a Port at one end and to an Ida at the other, a standard MASCOT restriction on direct activity to activity communication. This constraint is also captured by the multiplicity semantics of the RTN-SLg class diagram.

3. All design components (i.e., activities and IDAs) with no 'predecessors' are of type Ida; these are termed input IDAs and reflect their use in modelling protocols of the hardware/software interface.

   Component Invariant
   self.allinstances->forall(c | not(c.all_predecessors->size = 0 and c.oclType <> Ida))

4. Similarly, all design components with no 'successors' are of type Ida (termed output IDAs).

   Component Invariant
   self.allinstances->forall(c | not(c.all_successors->size = 0 and c.oclType <> Ida))

5. Every design component is connected by a path to an input Ida; i.e., no (cyclic) part of a design is not connected to an input device and therefore independent of all inputs.

   Component Invariant
   self.allinstances->forall(c | self.allinstances->exists(c' | c.all_predecessors->includes(c') and c'.allpredecessors->size = 0 and c'.oclType = Ida))

6. Similarly, every design component is connected by a path to an output Ida; i.e., no (cyclic) part of a design is not connected to an output device and therefore excluded from contributing to system output (with the result that any computations are wasted).

   Component Invariant
   self.allinstances->forall(c | self.allinstances->exists(c' | c.all_successors->includes(c') and c'.all_successors->size = 0 and c'.oclType = Ida))

7. An Ida may only connect to more than one writing activity if it does not have a pool protocol and it is not an output interface Ida.

   ConnectionDef invariant
   self.allinstances->forall(c1, c2 | self.to_ida->forall(i | not(c1 <> c2 and c1.port.activity <> c2.port.activity and c1.to_ida->includes(i) and c2.to_ida->includes(i) and (i.all_successors->size = 0 or i.kind = "pool"))))

8. An Ida may only connect to more than one reading Activity if it has a pool protocol.
Structuring Development Artifacts

**ConnectionDef invariant**

\[
\text{self.allInstances->forAll}(c_1, c_2 \mid \text{self.from_ida->forAll}(i \mid \text{not}(c_1 < c_2 \land \text{c1.port.activity <> c2.port.activity}) \\
\text{c1.from_ida->includes(i) and c2.from_ida->includes(i) and i.kind <> "pool"}))
\]

**Well-formed Activity State Machines**

The following well-formedness constraints apply to activity state machine elements of the RTN-SLg meta-model and are based on the principal restrictions set out in (Paynter, 2000):

1. A DynamicState may only read from ports that are connected to data-flows into an Activity (a) - i.e., from an Ida - and write to ports connected to data flows from an Activity (b) - i.e., to an Ida.
   
   a) **DynamicState invariant**
   
   \[
   \text{self.allInstances->forAll}(d \mid \text{self.local_op_def.reads_from.port->not exists (p \mid d.local_op_def.reads_from.port->includes(p) and p.connectionDef.to_ida->size <> 0)})
   \]

   b) **DynamicState invariant**
   
   \[
   \text{self.allInstances->forAll}(d \mid \text{self.local_op_def.writes_to.port->not exists (p \mid d.local_op_def.writes_to.port->includes(p) and p.connectionDef.from_ida->size <> 0)})
   \]

2. Three kinds of event labels may annotate transitions from static states, namely “S” (stim), “W” (write) and “R” (read) events. Any Port associated with an “S” event must be connected to a data-flow with a void protocol.

   **StaticTransition invariant**
   
   \[
   \text{self.allInstances->forAll}(t \mid \text{not}(t.event.event_type = "S" and (t.event.port.connectionDef.from_ida.kind = "pool" or t.event.port.connectionDef.from_ida.kind = "channel" or t.event.port.connectionDef.from_ida.kind = "signal")))
   \]

3. No Port with a void data flow may be read by any DynamicState as there can be no data to read.

   **DynamicState invariant**
   
   \[
   \text{self.allInstances->forAll}(d \mid \text{not}(d.local_op_def.reads_from.port.connectionDef.from_ida.kind = "stimulus" or d.local_op_def.reads_from.port.connectionDef.from_ida.kind = "dataless"))
   \]

4. A transition labelled with an “R” event must terminate at a DynamicState which reads from the same Port.

   **StaticTransition invariant**
   
   \[
   \text{self.allInstances->forAll}(t \mid \text{not}(t.event.event_type = "R" and t.event.port <> t.goes_to.local_op_def.reads_from.port))
   \]

5. If a DynamicState reads from a holding protocol (signal or channel), each transition into it must originate from a StaticState, thereby modelling the wait for data.
Real-Time Network Specification Language Structure (Graphical)

**DynamicState Invariant**

```
self.allInstances->forall(d1, d2 | 
not (d1.dynamic_transition_def.goes_to->includes(d2) and 
d2.local_op_def.reads_from.port.connectionDef.from_ida.kind = "channel" or 
d2.local_op_def.reads_from.port.connectionDef.from_ida.kind = "signal"))
```

6. No DynamicState which reads from a holding protocol may be the initial state.

**DynamicState Invariant**

```
self.allInstances->forall(d | 
not((d.local_op_def.reads_from.port.connectionDef.from_ida.kind = "signal" or 
d.local_op_def.reads_from.port.connectionDef.from_ida.kind = "channel") and 
d.initial_status.initial = True))
```

7. Event labels on transitions from static states are different.

**StaticTransition Invariant**

```
self.allInstances->forall(t1, t2 | 
not(t1 <> t2 and t1.event_type = t2.event_type and 
t1.event.port = t2.event.port))
```

8. A DynamicState which writes to a Port with a channel protocol must be followed by a StaticState which is itself the source of a transition with a "W" (write event) associated with the same Port. This captures the possibility of de-scheduling should the channel be full.

**DynamicState Invariant**

```
self.allInstances->select(self.local_op_def.writes_to.port.connectionDef.to_ida.kind = "channel")->forall(d | 
self.dynamic_transition_def.goes_to->exists(s | 
d.dynamic_transition_def.goes_to->includes(s) and 
soclType = StaticState and 
s.static_transition_def.event->size = 1 and 
s.static_transition_def.event.event_type = "W" and 
s.static_transition_def.event.port = d.local_op_def.writes_to.port))
```

9. The aggregated states in a CompositeDynamicState may neither write to a Port associated with a channel protocol, nor read from one associated with anything other than a pool. This is due to the need to model synchronisation points by explicit static states.

**CompositeDynamic Invariant**

```
self.allInstances->forall(c | 
self.dynamic_state->forall(d | 
not(c.dynamic_state->includes(d) and 
(d.local_op_def.reads_from.port.connectionDef.from_ida.kind <> "pool" or 
d.local_op_def.writes_to.port.connectionDef.to_ida.kind = "channel")))
```

10. A StaticState may have at most one exit transition. This is handled through a multiplicity constraint, but can also be stated in OCL as follows.

**StaticState Invariant**

```
self.allInstances->forall(s | s.static_transition_def->size <= 1)
```
11. ADTs must form an acyclic graph.
   - First we define a rule to populate the reflexive with_all association on the Adt class:

     ```
     Adt
     self.allInstances->forall(a1, a2)
     a1.with_adt->includes(a2) or
     self.allInstances->exists(a')
     a1.with_adt->includes(a') and
     a'.with_all->includes(a2))
     implies
     a1.with_all->includes(a2)
     ```

   - An invariant can now be defined to prevent cycles.

     ```
     Adt invariant
     self.allInstances->forall(a | not(a.with_all->includes(a)))
     ```

iii. Verification of RTN-SLg Elements against the Product Data Synthesis

To maintain consistency within the MATrA traceability Workspace, we define a number of constraints to verify RTN-SLg elements against the Product Data Synthesis, including:

1. The RTN-SLg model subject module exists in the PDS.

   ```
   RTN-SLg invariant
   self.allInstances->forall(r | self.bEmodeIAEO.build_element->exists(be | r.subject_module = be.module_name))
   ```

2. Activities are designated in the PDS as functions of the subject_module.

   ```
   Activity invariant
   self.allInstances->forall(a | self.RTN-SLg.bEmodeIAEO.build_element->exists(be | self.bEelementIAEO.build_element->exists(f | a.RTN-SLg.subject_module = be.module_name and
   a.name = f.function_name and
   be.encapsulates.target->includes(f)))))
   ```

3. Ports are designated in the PDS as interfaces to functions.

   ```
   Port invariant
   self.allInstances->forall(p | self.activity->exists(f | self.bEelementIAEO.build_element->exists(f1 | self.bEelementIAEO.build_element->exists(f | a.port->includes(p) and
   a.name = f.function_name and
   p.name = i.interface_name and
   f.hasInterface.target->includes(f1))))))
   ```

4. IDAs are specified in the PDS as interfaces with a connection to the appropriate port.
Real-Time Network Specification Language Structure (Graphical)

ConnectionDef invariant
self.allInstances->forall(c 
  self.from_ida->union(self.to_ida).bElementAO.build_element->exists(i 
  self.port.bElementAO.build_element->exists(p 
  c.port.name = p.interface_name and 
  (c.from_ida.name = i.interface_name or 
  c.to_ida.name = i.interface_name) and 
  (p.connection.target.connection.target->includes(i) or i.connection.target.connection.target->includes(p)) )))

5. ADTs are specified in the PDS as sub-modules of the subject_module29.

Adt invariant
self.allInstances->forall(a 
  self.rTN-SLg.bElementAO.build_element->exists(be 
  self.bElementAO.build_element->exists(m 
  a.rTN-SLg.subject_module = be.module_name and 
  a.name = m.module_name and 
  be.hasSubmodule.target->includes(m) )))

6. States are specified in the PDS as conditions of the Function that corresponds to the Activity containing the ASM to which a State belongs.

State invariant
self.allInstances->select (oclType = Static or oclType = Dynamic)->forall(s 
  self.stateMachine.activity->exists(a 
  self.stateMachine.activity.bElementAO.build_element->exists(f 
  self.stateMachine.activity.bElementAO.build_element.hasCondition.target->exists(c 
  a.state_machine.state_def->includes(s) and 
  a.name = f.function_name and 
  s.name = c.condition_Iabel and 
  f.hasCondition.target->includes(c) )))

CompositeDynamic invariant
self.allInstances->forall(d 
  self.stateMachine.activity->exists(a 
  self.stateMachine.activity.bElementAO.build_element->exists(f 
  self.stateMachine.activity.bElementAO.build_element.hasCondition.target->exists(c 
  a.state_machine.state_def->includes(d) and 
  a.name = f.function_name and 
  f.hasCondition.target->includes(c) and 
  c.hasSubcondition.target.condition_label->includesAll(d.dynamic_state.name) )))

7. Transitions (including source and target states) are defined in the PDS as corresponding and correctly related conditions30.

StaticTransition invariant
self.allInstances->forall(t 
  self.staticState.stateMachine.activity->exists(a 
  self.staticState.stateMachine.activity.bElementAO.build_element->exists(f 
  self.staticState.stateMachine.activity.bElementAO.build_element.hasCondition.target->exists(c 

29 An Abstract Data Type in the RTN-SL is a module in which attribute types and functions may be defined (Paynter, 2000).
30 The RTN-SLg syntax names only constituent states of a CompositeDynamic (see figure 4.19); therefore it is not possible to verify a transition destination state of this type against the PDS.
self.staticState.stateMachine.activity.bEelementAEO.build_element.hasCondition.target.occuringIn.source.exists(e | a.name = f.function_name and f.hasCondition.target->includes(c) and e.occuringIn.target->includes(c) and t.staticState.name = c.condition_label and e.leadsTo.target.condition_label = (t.goes_to->reject(t.goes_to.oclType = CompositeDynamic)).name and ((t.event.event_type.concat(t.event.port.name) = e.condition_label) or (t.timing_transition_definition.lb_ub = e.condition_label)))))))

**DynamicTransition invariant**

self.allInstances->select(self.compositeDynamic->size = 0)->forall(t | self.dynamicState.stateMachine.activity->exists(a | self.dynamicState.stateMachine.activity.bEelementAEO.build_element->exists(f | self.dynamicState.stateMachine.activity.bEelementAEO.build_element.hasCondition.target->exists(c | self.dynamicState.stateMachine.activity.bEelementAEO.build_element.hasCondition.target.occuringIn.source.exists(e | a.name = f.function_name and f.hasCondition.target->includes(c) and e.occuringIn.target->includes(c) and t.dynamicState.name = c.condition_label and e.leadsTo.target.condition_label = (t.goes_to->reject(t.goes_to.oclType = CompositeDynamic)).name and t.on = e.condition_label))))))

Note: the invariant on DynamicTransition is over transitions from a DynamicState; an invariant over transitions from a CompositeDynamic may be similarly expressed, but without PDS verification of the source state.

iv. Other RTN-SLg Consistency Constraints

We conclude our round-up of constraints over the RTN-SLg meta-model by considering restrictions arising from modelling decisions (alluded to in subsection 4.4.3.2.1). These are as follows:-

1. All Activity, Port, Ida, Adt and State names are unique31.

   **Activity invariant**
   self.allInstances->forall(a1, a2 | not(a1.name = a2.name and a1 <> a2))

   **Port Invariant**
   self.allInstances->forall(p1, p2 | not(p1.name = p2.name and p1 <> p2))

   **Ida invariant**
   self.allInstances->forall(i1, i2 | not(i1.name = i2.name and i1 <> i2))

   **Adt invariant**
   self.allInstances->forall(a1, a2 | not(a1.name = a2.name and a1 <> a2))

31 Assumes scope is an individual RTN-SLg model.
Real-Time Network Specification Language Structure (Graphical)

**StaticState Invariant**
self.allInstances->forall(s1, s2 | not(s1.name = s2.name and s1 <> s2))

**DynamicState Invariant**
self.allInstances->forall(d1, d2 | not(d1.name = d2.name and d1 <> d2))

2. All states must have an InitialStatus except for dynamic states that form part of a CompositeDynamic.

**State Invariant**
self.allInstances->forall(s | not (s.initial_status->size = 0 and (s.oclType <> DynamicState or s.compositeDynamic->size <> 1)))

3. Dynamic states that are part of composite dynamics have no individual timing definition or state transitions.

**CompositeDynamic Invariant**
self.allInstances->forall(c | self.dynamic_state->forall(d | not(c.dynamic_state->includes(d) and (d.dynamic_transition_def-> size > 0 or d.timing_def-> size > 0))))

4. Only one State in a StateMachine has a true initial status.

**StateMachine Invariant**
self.allInstances->forall(m | self.state_def->forall(s1, s2 | not(m.state_def->includes(s1) and m.state_def->includes(s2) and s1.initial_status=True and s2.initial_status=True and s1 <> s2)))

5. Dynamic states that are not part of a CompositeDynamic have a timing definition.

**DynamicState Invariant**
self.allInstances->forall(d | not (d.CompositeDynamic->size = 0 and d.timing_def-> size = 0))

4.4.3.2.3 O-Telos Implementation of RTN-SLg Base Classes

The O-Telos code below implements RTN-SLg meta-model base class elements.

**Definition of Artifact Properties**

```plaintext
Protocol in ArtifactProperty, SimpleClass
isa String
with
constraint
enum-protocol: $forall p/Protocol (p = "pool") or (p = "signal") or (p = "channel") or (p = "stimulus") or (p = "dataless")$
end

EventType in ArtifactProperty, SimpleClass
isa String
with
constraint
enum_event_type: $forall e/EventType (e = "S") or (e = "R") or (e = "W")$
end

InitialStatus in ArtifactProperty, SimpleClass
with
described_by
  initial : Bool
end
```
Definition of RTN-SLg Primitives

Adt in StructureElement, SimpleClass isA AerospaceEngineeringObject with
  has_property
    name : String
  has_part
    with_adt : Adt
  has_transitive_part
    with_all : Adt
end

Component in StructureElement, SimpleClass isA AerospaceEngineeringObject with
  has_property
    name : String
  has_part
    predecessor : Component;
    successor : Component;
  has_transitive_part
    all_successors : Component;
    all_predecessors : Component
  constraint
    abstract_component:
      $ \forall \ t/Token \ s/SimpleClass \ (t \in s) \implies \neg (t \in Component) \ $
end

Ida in StructureElement, SimpleClass isA Component with has_property
  kind : Protocol
  with_adt : Adt
end

Port in StructureElement, SimpleClass isA AerospaceEngineeringObject with
  has_property
    name : String
end

ConnectionDef in StructureElement, SimpleClass isA AerospaceEngineeringObject with has_part
  port : Port;
  from_ida : Ida;
  to_ida : Ida
end

State in StructureElement, SimpleClass isA AerospaceEngineeringObject with
  has_property
    initial_status : InitialStatus
  constraint
    abstract_state:
      $ \forall \ t/Token \ s/SimpleClass \ (t \in s) \implies \neg (t \in State) \ $
end

StaticState in StructureElement, SimpleClass isA State with
  has_property
    name : String
  has_part
    static_transition_def :
      StaticTransition
end

DynamicState in StructureElement, SimpleClass isA State with
  has_property
    name : String
  has_part
    dynamic_transition_def :
      DynamicTransition;
      timing_def : Timing
end

StateMachine in StructureElement, SimpleClass isA AerospaceEngineeringObject with has_part
  state_def : State
end

Activity in StructureElement, SimpleClass isA Component with
  has_part
    with_adt : Adt;
    port : Port;
    state_machine : StateMachine
end

StaticTransition in StructureElement, SimpleClass isA AerospaceEngineeringObject with
  has_property
    on : String
  has_part
    goes_to : State;
    event : Event;
    timing_transition_definition :
      TimingTransitionDef
end

DynamicTransition in StructureElement, SimpleClass isA AerospaceEngineeringObject with
  has_property
    on : String
  has_part
    goes_to : State
end

Timing in StructureElement, SimpleClass isA AerospaceEngineeringObject with
  has_property
    bcet : String;
    wcet : String;
    wcrt : String
end

WritesToClause in StructureElement, SimpleClass isA AerospaceEngineeringObject with has_part
  port : Port
end

ReadsFromClause in StructureElement, SimpleClass isA AerospaceEngineeringObject with
  port : Port;
  by : WCRTonRd
end

WCRTonRd in StructureElement, SimpleClass isA AerospaceEngineeringObject with
  has_property
    finish : String
end

LocalOpDef in StructureElement, SimpleClass isA AerospaceEngineeringObject with
  has_part
    reads_from : ReadsFromClause;
    writes_to : WritesToClause
end

Event in StructureElement, SimpleClass isA
Real-Time Network Specification Language Structure (Graphical)

isA AerospaceEngineeringObject
with has_property
  event_type : EventType
has_part
  port : Port
end

TimingTransitionDef in StructureElement,
SimpleClass isA AerospaceEngineeringObject with
has_property
  lb_ub : String
end

Definition of RTN-SLg Specification
RTN_SLg in DevelopmentStructure,
SimpleClass isA AerospaceEngineeringObject with
has_property
  model_name : String;
subject_module : String;
rtn_date: Date
has_structure
  rtn_comments :
    MatraNaturalLanguageStructure
has_element

4.4.3.3 RTN-SLg Worked Examples
We now illustrate aspects of the RTN-SLg meta-model by instantiating the above classes with some worked examples. Specifically:

- Hypothetical RTN-SLg Example (from subsection 4.4.2.2);
- A hypothetical Composite Dynamic State;
- An RTN-SLg specification fragment for a Missile Tracking System.

I. Hypothetical RTN-SLg Example
In subsection 4.4.2.2 we introduced an example RTN-SLg specification (illustrated in figure 4.20) to demonstrate key aspects of the language’s graphical syntax. The following is therefore an O-Telos implementation of base classes for this example:

Sample Population of RTN-SLg (from figure 4.20)
Adt1Type in Adt, Token with
  adtName : "adt1"
end

Port1 in Port, Token with
  portName : "P1"
end

Activity1 in Activity, Token with
  activityName : "A1"
with_adt
  withAdt1 : Adt1Type
port
  port1 : Port1
end

Ida1 in Ida, Token with
  idaName : "Ida1"
kind
  idaKind : "pool"
with_adt
  withAdt1 : Adt1Type
end

Ida2 in Ida, Token with
  idaKind : "stimulus"
end

Ida3 in Ida, Token with
  idaKind : "channel"
end

Adt2Type in Adt, Token with
  adtName : "adt2"
with_adt
  withAdt1 : Adt1Type
end

Adt3Type in Adt, Token with
  adt_name : "adt3"
with_adt
  withAdt1 : Adt1Type
end

rtn_adt : Adt;
rtn_port : Port;
rtn_ida : Ida;
rtn_activity : Activity;
rtn_connection_def : ConnectionDef;
rtn_static_state : StaticState;
rtn_dynamic_state : DynamicState;
rtn_composite_dynamic_state : CompositeDynamic;
rtn_static_transition : StaticTransition;
rtn_dynamic_transition : DynamicTransition;
rtn_writes_to_clause : WritesToClause;
rtn_reads_from_clause : ReadsFromClause;
rtn_WCRTonRd : WCRTonRd;
rtn_local_op_def : LocalOpDef;
rtn_event : Event;
rtn_timing_transition_def : TimingTransitionDef;
rtn_state_machine : StateMachine
end
Port2 in Port, Token with
  name
    portName : "P2"
  end

Port3 in Port, Token with
  name
    portName : "P3"
  end

Port4 in Port, Token with
  name
    portName : "P4"
  end

StaticStateA in StaticState, Token with
  name
    staticStateName : "A"
    initialStatus : StaticStateAStatus
    static_transition_def
      staticTransitionDef : StaticTransitionAtoB
    end

StaticStateAStatus in InitialStatus, Token with
  initial
    initialState : True
  end

StaticTransitionAtoB in StaticTransition, Token with
  goes_to
    goTo : DynamicStateB
  event
    transEvent : AtoBEvent
  end

AtoBEvent in Event, Token with
  event_type
    eventType : "S"
  port
    eventPort : Port2
  end

DynamicStateB in DynamicState, Token with
  name
    dynamicStateName : "B"
    initial_status
      initialStatus : DynamicStateBStatus
      dynamic_transition_def
        dynamicTransitionDef1 : DynamicTransitionBtoC;
        dynamicTransitionDef2 : DynamicTransitionBtoD
      end
    local_op_def
      localOpDef : LocalOpDefStateB
    timing_def
      timingDef : TimingDefStateB
    end

DynamicStateBStatus in InitialStatus, Token with
  initial
    initialState : False
  end

DynamicTransitionBtoC in DynamicTransition, Token with
  transitionOn : "C1"
  goes_to
    goTo : StaticStateC
  end

DynamicTransitionBtoD in DynamicTransition, Token with
  transitionOn : "C2"
  goes_to
    goTo : DynamicStateD
  end

LocalOpDefStateB in LocalOpDef, Token with
  reads_from
    readsFrom : ReadsFromClauseStateB
  end

ReadsFromClauseStateB in ReadsFromClause, Token with
  port
    readPort : Port3
  by
    byDeadline : WCRTonRdStateB
  end

WCRTonRdStateB in WCRTonRd, Token with
  finish
    finishTime : "U1"
  end

TimingDefStateB in Timing, Token with
  bcet
    BCETime : "L1"
  wcet
    WCETime : "U2"
  wcrt
    WCRTime : "U3"
  end

StaticStateC in StaticState, Token with
  name
    staticStateName : "C"
    initial_status
      initialStatus : StaticStateCStatus
      static_transition_def
        staticTransitionDef : StaticTransitionCtoD
      end

StaticStateCStatus in InitialStatus, Token with
  initial
    initialState : False
  end

StaticTransitionCtoD in StaticTransition, Token with
  goes_to
    goTo : DynamicStateD
  timing_transition_definition
    timingTransitionDefinition : CtoDTimingTransition
  end

CtoDTimingTransition in TimingTransitionDef, Token with
  lb_ub
    lbUb : "[t1, t2]"
  end

DynamicStateD in DynamicState, Token with
  name
    dynamicStateName : "D"
    initial_status
      initialStatus : DynamicStateDStatus
      dynamic_transition_def
        dynamicTransitionDef : DynamicTransitionDtoE
      end
    local_op_def
      localOpDef : LocalOpDefStateD
    timing_def
      timingDef : TimingDefStateD
    end

DynamicStateDStatus in InitialStatus, Token with
  initial
    initialState : False
DynamicTransitionDtoE in DynamicTransition, Token with
  transitionOn : "C3"
  goes_to
    goesTo : StaticStateE
end

LocalOpDefStateD in LocalOpDef, Token with
  writes_to
    writesTo : WritesToClauseStateD
end

WritesToClauseStateD in WritesToClause, Token with
  port
    writePort : Port4
end

TimingDefStateD in Timing, Token with
  BCETime : "L2"
  WCETime : "U4"
  WCRTime : "U5"
end

StaticStateE in StaticState, Token with
  staticStateName : "E"
  initial_status
    initialStatus : StaticStateEStatus
  static_transition_def
    staticTransitionDef
      StaticTransitionEtoA
end

StaticStateEStatus in InitialStatus, Token with
  initial
    initialState : False
end

StaticTransitionEtoA in StaticTransition, Token with
  goes_to
    goesTo : StaticStateA
  event
    transEvent : EtoAEvent
end

EtoAEvent in Event, Token with
  event_type
    eventType : "W"
  port
    eventPort : Port4
end

Activity2StateMachine in StateMachine, Token with
  state_def
    stateDef1 : StaticStateA;
    stateDef2 : DynamicStateB;
    stateDef3 : StaticStateC;
    stateDef4 : DynamicStateD;
    stateDef5 : StaticStateE
end

Activity2 in Activity, Token with
  activityName : "A2"
  with_adt
    withAdt1 : Adt3Type
    port
      port1 : Port2;
      port2 : Port3;
      port3 : Port4

state_machine
  stateMachine : Activity2StateMachine
end

P1toIdal in ConnectionDef, Token with
  port
    outPort : Port1
    to_ida
      toIda : Idal
end

IdaltoP3 in ConnectionDef, Token with
  port
    inPort : Port3
    from_ida
      fromIda : Idal
end

Ida2toP2 in ConnectionDef, Token with
  port
    inPort : Port2
    from_ida
      fromIda : Idal
end

P4toIda3 in ConnectionDef, Token with
  port
    outPort : Port4
    to_ida
      toIda : Idal
end

Net1 in RTN_SLg, Token with
  model_name
    modelName : "RTN-SLg Example 1"
  rtn_adt
    rtnAdt1 : Adt1Type;
    rtnAdt2 : Adt2Type;
    rtnAdt3 : Adt3Type
  rtn_port
    rtnPort1 : Port1;
    rtnPort2 : Port2;
    rtnPort3 : Port3;
    rtnPort4 : Port4
  rtn_ida
    rtnIdal : Idal;
    rtnIda2 : Idal;
    rtnIda3 : Idal
  rtn_activity
    rtnActivity1 : Activity1;
    rtnActivity2 : Activity2
  rtn_connection_def
    rtnConnectionDef1 : P1toIdal;
    rtnConnectionDef2 : IdaltoP3;
    rtnConnectionDef3 : Ida2toP2;
    rtnConnectionDef4 : P4toIda3
  rtn_static_state
    rtnStaticState1 : StaticStateA;
    rtnStaticState2 : StaticStateC;
    rtnStaticState3 : StaticStateE
  rtn_dynamic_state
    rtnDynamicState1 : DynamicStateB;
    rtnDynamicState2 : DynamicStateD
  rtn_dynamic_transition
    rtnDynamicTransition1 : StaticTransitionAtoB;
    rtnDynamicTransition2 : StaticTransitionCtoD;
    rtnDynamicTransition3 : StaticTransitionEtoA
  rtn_dynamic_transition
    rtnDynamicTransition1 : DynamicTransitionBtoC;
    rtnDynamicTransition2 : DynamicTransitionDtoE
  rtn_timing_def
    rtnTimingDef1 : TimingDefStateB;
    rtnTimingDef2 : TimingDefStateD
  rtn_reads_from_clause
    rtnReadsFromClause1 : ReadsFromClauseStateB

ii. A Hypothetical Composite Dynamic State

We now present O-Telos implementation of an exemplar Composite Dynamic State using the graphical Activity State Machine fragment in figure 4.23:

![Diagram](image_url)

**Figure 4.23 - 'Hypothetical Composite Dynamic State'

Sample Population of RTN-SLq Composite Dynamic State

ExampleComposite in CompositeDynamic, Token with initial_status:
  ExampleCompositeStatus
  dynamic_statel : DynamicStatel;
  dynamic_state2 : DynamicState2
  dynamic_transition_def
  dynamicTransitionDef : CompositeDynamicTransition
timing_def
timingDef : CompositeDynamicTimingDef
end
ExampleCompositeStatus in InitialStatus, Token with initial
  initialStatus : False
end
DynamicStatel in DynamicState, Token with name
  dynamicStateName : "Statel"
  local_op_def
  localOpDef : LocalOpDefStatel
end
DynamicState2 in DynamicState, Token with name
  dynamicStateName : "State2"
  local_op_def
  localOpDef : LocalOpDefState2
end
LocalOpDefStatel in LocalOpDef, Token with
  reads_from
  readsFrom : ReadsFromClauseStatel
  writes_to
  writesTo : WritesToClauseStatel
end
LocalOpDefState2 in LocalOpDef, Token with
  reads_from
  readsFrom : ReadsFromClauseState2
  writes_to
  writesTo : WritesToClauseState2
end
CompositeDynamicTransition in DynamicTransition, Token with on
  transitionOn : "True"
go_to
  goesTo : AnotherState
end
CompositeDynamicTimingDef in Timing, Token with
  bcet
  BCETime : "L2"
  wcet
  WCETime : "U3"
  wcrt
  WCRTime : "U4"
end

155
iii. An RTN-SLg Specification (Fragment) for a Missile Tracking System

We conclude our demonstration of the RTN-SLg meta-model with a small example network taken from Paynter et al. (2000) and illustrated in figure 4.24. It is based on a design fragment for a typical Missile Tracking System and consists of two activities linked by a channel protocol; the arrival of a target Image on port P1 triggers activity A1 which performs some image processing to determine the target co-ordinates and sends them to port P2. This information is then passed to A2 (via the Image Store channel) which reads from port P3, calculates movement of the target and outputs this vector to the rest of the system via port P4.

**Figure 4.24 - ‘RTN-SLg Specification for A Missile Target Tracking System’**

O-Telos instantiation of the RTN-SLg meta-model capturing information contained in figure 4.24 is as follows:-

```
Sample Population of RTN-SLg Missile Tracking System Example

Image in Adt, Token with
  adtName : "Image"
end

Port1 in Port, Token with
  portName : "P1"
end

Port2 in Port, Token with
  portName : "P2"
end

StaticStateA in StaticState, Token with
  staticStateName : "A"
  initial_status : StaticStateAStatus
  static_transition_def : StaticTransitionDef
  staticTransitionAtOB
end

StaticStateAStatus in Initialstatus,
  initial : True
end

StaticTransitionAtOB in StaticTransition,
  Token with
    goes_to : DynamicStateB
    event : AtoBEvent
  end

AtoBEvent in Event, Token with
  eventType : "R"
  port : Port1
end

DynamicStateB in DynamicState, Token with
  dynamicStateName : "B: Image Processing"
  initial_status : DynamicStateBStatus
  dynamic_transition_def : DynamicTransitionDef
  DynamicTransitionBtoC
  local_op_def : LocalOpDefStateB
  timing_def : TimingDefStateB
end
```
end

DynamicStateB in InitialStatus, Token with
  \text{initialState} : \text{False}
end

DynamicTransitionB to C in DynamicTransition, Token with
  transitionOn : "True"
  \text{goesTo} : \text{StaticStateC}
end

LocalOpDefStateB in LocalOpDef, Token with
  \text{readsFrom} : \text{ReadsFromClauseStateB}
  \text{writesTo} : \text{WritesToClauseStateB}
end

ReadsFromClauseStateB in ReadsFromClause, Token with
  \text{readPort} : \text{Port1}
end

WritesToClauseStateB in WritesToClause, Token with
  \text{writePort} : \text{Port2}
end

TimingDefStateB in Timing, Token with
  BCETime : "L1"
  WCETime : "U1"
  WCRTime : "U2"
end

StaticStateC in StaticState, Token with
  staticStateName : "C"
  \text{initialStatus} : \text{StaticStateCStatus}
  static_transition_def
    \text{StaticTransitionCtoA}
end

StaticStateCStatus in InitialStatus, Token with
  \text{initialState} : \text{False}
end

StaticTransitionC to A in StaticTransition, Token with
  \text{goesTo} : \text{StaticStateA}
  \text{transEvent} : \text{CtoAEvent}
end

CtoAEvent in Event, Token with
  \text{eventType} : "W"
  \text{eventPort} : \text{Port2}
end

Activity1StateMachine in StateMachine, Token with
  state_def
    stateDef1 : \text{StaticStateA};
    stateDef2 : \text{DynamicStateB};
    stateDef3 : \text{StaticStateC}
end

Activity1 in Activity, Token with
  \text{activityName} : "A1: Identify Target"
  \text{withAdt}
    \text{withAdt1} : \text{Image}
  \text{port}
    \text{port1} : \text{Port1};
    \text{port2} : \text{Port2}
  \text{stateMachine}
    \text{stateMachine} : Activity1StateMachine
end

Ida1 in Ida, Token with
  \text{idaKind} : "signal"
end

Ida2 in Ida, Token with
  \text{idaName} : "Image Store"
  \text{idaKind} : "channel"
end

Ida3 in Ida, Token with
  \text{idaKind} : "pool"
end

Threat in Adt, Token with
  \text{adtName} : "Threat"
end

Port3 in Port, Token with
  \text{portName} : "P3"
end

Port4 in Port, Token with
  \text{portName} : "P4"
end

StaticStateD in StaticState, Token with
  staticStateName : "D"
  \text{initialStatus} : \text{StaticStateDStatus}
  static_transition_def
    \text{StaticTransitionDtoE}
end

StaticStateDStatus in InitialStatus, Token with
  \text{initialState} : \text{True}
end

StaticTransitionD to E in StaticTransition, Token with
  \text{goesTo} : \text{DynamicStateE}
  \text{transEvent} : \text{DtoEEvent}
end

DtoEEvent in Event, Token with
  \text{eventType} : "R"
  \text{eventPort} : \text{Port3}
end

DynamicStateE in DynamicState, Token with
  \text{dynamicStateName} : "E: Target Vector"
Real-Time Network Specification Language Structure (Graphical)

Calculation

initial status

  initialStatus : DynamicStateEStatus
dynamic_transition_def

dynamicTransitionDef1 :
DynamicTransitionEtoD
local_op_def
  localOpDef : LocalOpDefStateE
timing_def
  timingDef : TimingDefStateE
end

DynamicStateEStatus in InitialStatus,
Token with
initial
  initialState : False
end

DynamicTransitionEtoD in
DynamicTransition, Token with
on transitionOn : "True"
goes_to
goesTo : StaticStateD
end

LocalOpDefStateE in LocalOpDef, Token with
reads_from
  readsFrom : ReadsFromClauseStateE
writes_to
  writeTo : WritesToClauseStateE
end

ReadsFromClauseStateE in ReadsFromClause,
Token with
port
  readPort : Port3
end

WritesToClauseStateE in WritesToClause,
Token with
port
  writePort : Port4
end

TimingDefStateE in Timing, Token with
bcat
  BCETime : "L2"
wcet
  WCETime : "U1"
wcr
  WCRTime : "U4"
end

Activity2StateMachine in StateMachine,
Token with
state_def
  stateDef1 : StaticStateB;
  stateDef2 : DynamicStateE
end

Activity2 in Activity, Token with
name
  activityName : "A2: Calculate Target Vector"
with_adt
  withAdt1 : Image;
  withAdt2 : Threat
port
  port1 : Port3;
  port2 : Port4
state_machine
  stateMachine : Activity2StateMachine
end

IdaltoPl in ConnectionDef, Token with
port
  inPort : Port1
from_ida
  fromIda : Ida1
end

P2toIda2 in ConnectionDef, Token with
port
  outPort : Port2
to_ida
  toIda : Ida2
end

Ida2toP3 in ConnectionDef, Token with
port
  inPort : Port3
from_ida
  fromIda : Ida2
end

P4toIda3 in ConnectionDef, Token with
port
  outPort : Port4
to_ida
  toIda : Ida3
end

Net2 in RTN_SLg, Token with
model_name
  modelName : "RTN-SLg Example 2"
rtn_adt
  rtnAdt1 : Image;
  rtnAdt2 : Threat
rtn_port
  rtnPort1 : Port1;
  rtnPort2 : Port2;
  rtnPort3 : Port3;
  rtnPort4 : Port4
rtn_ida
  rtnIda1 : Ida1;
  rtnIda2 : Ida2;
  rtnIda3 : Ida3
rtn_activity
  rtnActivity1 : Activity1;
  rtnActivity2 : Activity2
rtn_connection_def
  rtnConnectionDef1 : Ida1toP1;
  rtnConnectionDef2 : P2toIda2;
  rtnConnectionDef3 : Ida2toP3;
  rtnConnectionDef4 : P4toIda3
rtn_static_state
  rtnStaticState1 : StaticStateA;
  rtnStaticState2 : StaticStateC;
  rtnStaticState3 : StaticStateD
rtn_dynamic_state
  rtnDynamicState1 : DynamicStateB;
  rtnDynamicState2 : DynamicStateE
rtn_static_transition
  rtnStaticTransition1 : StaticTransitionAtoB;
  rtnStaticTransition2 : StaticTransitionCtoA;
  rtnStaticTransition3 : StaticTransitionDtoE
rtn_dynamic_transition
  rtnDynamicTransition1 : DynamicTransitionEtoD;
  rtnDynamicTransition2 : DynamicTransitionEtoD
rtn_timing_def
  rtnTimingDef1 : TimingDefStateB;
  rtnTimingDef2 : TimingDefStateE
rtn_reads_from_clause
  rtnReadsFromClause1 :
  ReadsFromClauseStateB;
  rtnReadsFromClause2 :
  ReadsFromClauseStateB;
  rtnWrites_to_clause
  rtnWritesToClause1 :
  WritesToClauseStateB;
  rtnWritesToClause2 :
  WritesToClauseStateB
rtn_local_op_def
  rtnLocalOpDef1 : LocalOpDefStateB;
  rtnLocalOpDef2 : LocalOpDefStateE
4.4.4 Relationship to the Traceability Dimensions

We reiterate that this thesis is concerned mainly with providing a set of notation meta-models for an avionics traceability environment, rather than with defining actual linkages between them. However, it is worth noting that engineers might conceivably employ the Use Case approach from subsection 4.3 in conjunction with RTN-SLg and in doing so, create vertical trace relations between Use Case and Activity elements using the approach outlined in 3.3.6.3.2. Moreover, for hazard analysis purposes practitioners may wish to link network elements to safety techniques such as HAZOP (again giving vertical traceability), as shown for MASCOT designs in McDermid & Pumfrey (1994).

4.4.5 Summary

Real-Time Networks are a design method used extensively throughout the defence industry (including aerospace). On that basis, we chose RTN-SL as our representative design notation for the MATrA traceability framework.

A novel meta-model capturing the graphical syntax of RTN-SLg specifications was proposed, together with a set of restrictions that preserve their internal and PDS consistency. A partial O-Telos implementation of the model was then populated to demonstrate some worked examples.

Again, evaluation of the RTN-SLg structure is discussed fully in Chapter Seven.
4.5  Towards a SPARK Ada Programming Language Structure

4.5.1  Introduction
This section introduces a structure capturing a subset of the concrete (well-formedness) syntax for SPARK, a programming language supporting development of software for high integrity (including safety-critical) applications.

The structure was developed in parallel with a modelling philosophy - a set of principles (guidelines) by which practitioners may extend the structure to encompass the complete SPARK syntax. We go on to demonstrate how these guidelines may be similarly applied to the development of structures for other languages with a concrete syntax - in this case RTN-SL (textual).

4.5.2  Motivation
In safety-critical systems engineering, choice of programming language is of paramount importance, motivated by a need to increase the likelihood of software behaving as intended and hence reduce the risks arising from errors to an acceptable level. Carré et al. (1990) identify a number of factors determining suitability of a programming language for use in high integrity systems:-

- **Logical soundness**: the language should contain no ambiguities;
- **Simplicity of formal description**: it should be relatively simple to describe the language;
- **Expressive power**: the language should be sufficiently rich to describe 'real' systems;
- **Security**: it should be possible to determine statically if a program conforms to the language rules;
- **Verifiability**: program verification should be tractable for industrial-scale applications;
- **Bounded time and space requirements**: the resource requirements of a program should be determinable statically to avoid possible run-time errors due to exhaustion of finite resources.

Ada (Barnes, 1996) was written specifically for use in developing such applications and includes a number of features amenable to the production of dependable software. However, as Storey (1996) notes it does not represent an ideal, partly because some of the above factors are in conflict; for instance the corollary of expressive power is often complexity and the attendant problems of verification and security. Therefore, in recent years much effort has been devoted to development of an Ada subset - a so-called 'safe' subset comprising only those facilities necessary for writing high integrity programs. Safe subsets of any language (see Cullyer et al., 1991 for a comparison) are formed by excluding and restricting all features or constructs present in the full version that may prevent verification.

Arguably the most prominent Ada subset is SPARK (Barnes, 1997) which is also one of the few programming languages to boast complete formal semantics. The core of SPARK is an executable kernel derived from a subset of Ada 95\(^2\); subset in the sense that features such as goços, aliasing, default parameters (for procedures and functions), recursion, tasks, user-defined exceptions and exception handlers have all been omitted, while the type model is simplified through removal of pointers, type

\(^2\) A version of SPARK based on Ada 83 is also available.
aliasing, derived types and anonymous types.

In addition to the kernel, SPARK also includes some additional non-executable features termed annotations which are added by users in the form of Ada comments and which permit analysis and proof. Annotations are ignored by standard Ada compilers, but processed by the SPARK tool-set that supports the language. They divide into two categories: those providing flow analysis and those providing formal verification. Flow analysis is permitted at two levels: data flow analysis which just concerns the direction of data flow (e.g., checking variables are not read before being initialised), and information flow which also checks the coupling between variables (Barnes, 1997). Proof annotations meanwhile permit analysis of dynamic behaviour, including the results of functions, assertions such as loop invariants and pre and post conditions of sub-programs.

Ada remains the preferred language for high integrity systems in the defence and aerospace sectors (Wichmann, 1997), often as a means of implementing Real-Time Network design specifications (Paynter, 2000). However SPARK is increasingly finding favour in these domains (cf. King et al., 2000), not least due its support for formal verification which helps satisfy the rigorous demands of Defence Standards 00-55 (MoD, 1997) and 00-56 (MoD, 1996). That said, we are concerned purely with representation and so arguments pro and contra use of language subsets (and hence SPARK) are not an issue. Instead we apply a common set of principles from which meta-models representing core and specific features of both Ada and SPARK may be formed.

The inclusion of SPARK also enables us to demonstrate structuring of artifacts at the software implementation level, which allied to Circuit Diagrams (hardware implementation) from subsection 3.3.7, together with UCRS and RTN-SLg (subsections 4.3 and 4.4), ensures all life-cycle phases are represented within the MATrA framework shown by this thesis; safety and product management are addressed in Chapter Five.

4.5.2.1 SPARK Ada Overview

The SPARK kernel language comprises standard Ada features such as package, private types, typed constants, unconstrained array types, functions returning composite types and the library system. However, given our level of interest, namely to demonstrate a modelling philosophy on structuring SPARK programs for traceability purposes, a full discussion on these constructs is beyond our scope; interested readers are therefore referred to Barnes (1997).

In order to provide proof of concept for both the philosophy and the structure, we concentrate on representation of a single Ada construct - the package. A package provides means to group entities such as data types, procedures and other packages within a common framework. They also allow information hiding whereby details of the private part and body are not visible to external users of the package.

---

33 In MATrA, these may conceivably trace back (vertically) to pre and post-conditions of Use Case Models as discussed in subsection 4.3.

34 See Wichmann for an argument against using 'safe' language subsets.
Packages are defined in two parts, a specification which describes the interface to an external client (and which is the focus for this thesis) and the body which provides implementation details; the two are always textually distinct as the following demonstrates (words in bold, as in all examples that follow, denote reserved words that may not be used for any other purpose):

```
package P is -- specification
    ... -- visible part
private
    ... -- private part (optional)
end P;

package body P is -- body
    begin
    ... -- initialisation (optional)
end P;
```

Packages can be nested, with those at the top level - termed library packages - forming separate compilations; the body and specification of a library package may be compiled separately. Library packages can also include child packages. This is specified using a 'dot' notation prefixed with the parent name as follows:

```
package P.Child is
    ...
end P.Child;
```

The other form of child package is a private child, written as:

```
private package P.Child is
    ...
end P.Child;
```

As previously indicated, packages can contain types and procedures. Pre-defined types in SPARK are integer, float, boolean, character and string, although additional types can also be defined. For example in the following, BankAngle is an integer with range of values from -45 to +45, whilst WarningLight is an enumerated type with two literal values (On and Off):

```
type BankAngle is range -45 .. 45;
type WarningLight is (On, Off);
```

Procedure specification statements provide a procedure name, together with the types and modes of their parameters. Valid modes are in and out, indicating direction of information flow; thus in the following Present_Bank of Read_Bank_Indicator is an in parameter as it provides the in-going information to be read.

```
procedure Read_Bank_Indicator(Present_Bank: in BankAngle);
```

In addition to language features such as those outlined above, SPARK includes a number of
Structuring Development Artifacts

annotations. These are specified using the ‘--#’ convention, two hyphens being the standard Ada comment prefix. Recall that annotations are ignored by the Ada compiler and divide into two groups, those for information and data analysis and those for code verification. In this thesis, we concentrate on a subset of the former. Specifically:

- --# global - global definitions in procedures (which declare 'imported' or 'exported' global variables) or functions (which import such variables);
- --# derives - dependency relations in procedures (which specify the imported variables required to derive the value of each exported variable);
- --# inherit - the inherit clause in package declarations (which restricts penetration of the package to items specifically imported from other packages);
- --# own - the own variable clause in package specifications (which makes actions on the package state visible to analysis tools);
- --# initializes - the initialisation annotation in package specifications (which indicates initialisation by the package of its own state).

4.5.3 Tracing Software Implementations in MATrA: A SPARK Ada Model

4.5.3.1 Concepts

The previous subsection introduced our modelling scope, i.e. particular SPARK elements to be considered in this thesis - namely package specifications, together with the --# global, --# derives, --# inherit, --# own and --# initializes annotations.

As indicated above, SPARK has a complete formal semantics which in principle could be expressed in OCL as constraints over elements of our meta-model. However, the SPARK tool-set and in particular the Examiner tool which checks conformance to the rules of the kernel language means we can have considerable confidence in the integrity of code produced. It is therefore deemed superfluous to repeat validation during the tool2matra transfer process. Further, given that we stated PDS consistency checks (in OCL) for previous notations, we have not specified similar restrictions here. Note however that the following (informal) checks might apply:

- SPARK packages and child packages map to corresponding PDS modules and sub-modules;
- SPARK procedures map to PDS functions for the corresponding module;
- SPARK types map to PDS attributes for the corresponding module.

4.5.3.1.1 A Modelling Philosophy towards the Object-Based Representation of String Grammars for Specification & Code Level Languages

Representation of the nominated constructs takes as its starting point the concrete SPARK grammar (featured in Appendix One of [Barnes, 1997]) which is stated using Backus Naur Form (BNF) notation.

---

35 Nor would constraints be required given the availability of an inverse "matra2tool" function (a future work item discussed in subsection 7.4.7), which to ensure all edits are conducted using the SPARK tool-set, would not extend to this structure.

163
In BNF, categories (i.e., entities) to be represented are defined in terms of other categories using productions consisting of the name being defined, followed by the ::= symbol and a defining sequence (which may or may not include punctuation and reserved word 'tokens'); categories that cannot be further decomposed are known as terminals. Other symbols of note are [ ] square brackets enclosing optional items, { ) braces enclosing optional items that may be omitted or appear n > 0 times and the | vertical bar separating alternatives.

Our philosophy (developed in parallel with the SPARK meta-model) provides a series of principles or guidelines allowing object-based representation of language constructs stated as BNF categories, and to do so at a level of abstraction which is sufficient to identify traceability primitives. The guidelines' phrasing is deliberately general allowing their application to other languages with a concrete BNF syntax. We demonstrate this in subsection 4.5.4 by representing a fragment of the RTN-SL textual syntax. The guidelines are as follows:-

1. Each BNF category is represented as a class (both an instance of the StructureElement metaclass and a specialisation of AerospaceEngineeringObject).

2. Categories forming the defining sequence of a category are themselves represented as classes (related to the subject class through aggregation), except where rule 3 applies.

3. Where nothing is gained traceability-wise by mirroring definition of a category in terms of other categories (and hence the corresponding class in terms of other classes) - that is, no constituent categories can be represented by classes that map to PDS elements (nor trace to elements of other notations) - then it is treated as atomic and represented through specialisation of the standard String type (as well as an instance of StructureElement and a specialisation of AerospaceEngineeringObject).

4. All tokens (i.e., reserved words - such as from, global, inherit, etc. - together with punctuation marks - such as commas, semi-colons, parentheses and ampersands) are represented as specialisations of the standard String class (and also as instances of ArtifactProperty).

5. Categories of a defining sequence that exist exactly once are modelled with a multiplicity constraint of one (1) for aggregate component classes; attributes representing tokens have an implicit multiplicity of one.

6. Optional items from a defining sequence (expressed in BNF using square brackets) are modelled as follows:-

   a) Categories are represented by applying a zero-or-one multiplicity constraint (0..1) on the component side of associations between the class representing the category being described (aggregate) and each class representing an optional defining category (component).

   b) Tokens are promoted from attributes to classes (linked to their host class using composition) to allow for expression of optionality (see also rule 8 on alternatives).
7. Categories of a defining sequence that may be omitted, appear once, or be repeated several times (expressed in BNF using the braces formalism) are modelled using a zero-to-many multiplicity constraint (0..*) on the component side of associations between the class representing the category being described (aggregate) and each class representing an optional, singular or repeated defining category (component).

8. Categories decomposed into alternatives (expressed in BNF using the vertical bar separator) are modelled differently depending on what form their defining sequence takes.

(a) For a category 'c' whose alternatives are exclusively 1-tuples (e.g. c:: = a 1 b) and which is represented by a class 'C', we adopt the following convention:-

Both categories and tokens (the latter promoted from attributes as per rule 6b) are modelled as component classes of C with an aggregation multiplicity of 1 (or 0..1 if optional), and with a dashed line (bearing an 'exclusive-or' annotation) connecting their Class Diagram icons; semantics are that each occurrence of C must (if multiplicities are 1) or may (if 0..1) be composed of an occurrence of one entity from a set of alternatives, to the exclusion of all others.

(b) For a category 'c' whose alternatives include n > 1-tuples, i.e., a set of category or token elements (e.g., c:: = ab I ad) and which is represented by a class 'C', we adopt the following convention(s):-

where a subset of elements common to each alternative exists, e.g., a in ab I ad, then:-

i. if a is a category, we represent it as a component class of C with multiplicity 1 (if mandatory for all alternatives) or 0..1 (if optional for all);
ii. if a is a token, we represent it as an attribute of C (if mandatory for all alternatives) or as a component class of C with multiplicity 0..1 (if optional for all).

These are the main properties of alternatives for which a modelling philosophy can be stated. We make the observation that following subtraction of common elements, the remaining subsets for each alternative in example (b) are all disjoint 1-tuples (i.e. c:: = ab I ad) and therefore corresponding classes may be linked using the diagramming conventions described previously for (a).

However to preserve BNF semantics, most scenarios featuring n > 1 tuples require an OCL invariant to unambiguously state multiplicity and/or valid element combinations. In particular, where n > 1 tuple items must be and-ed together to form an alternative, or where on subtraction of common elements, the remaining subsets include either zero-tuples (e.g. the middle alternative in c:: = ab I b I bc) or tuples whose elements are not disjoint (e.g., c:: = abd I bd I b e); note elements that are 'common' to alternatives, but which differ in terms of optionality (e.g. [b], b in c:: = a [b] I b c) are treated as non-common.

9. The modelling of a defining sequence in which an item type appears at least once, but which may be repeated (as a comma or semi-colon delimited list) - i.e., category ::= item [, item] - features a subtle variation from the BNF syntax; we represent the mandatory single item as a class (corresponding to a 'list head') and the optional repeated part (enclosed between braces in BNF) as
a separate ‘list-item’ class containing an appropriate delimiter attribute, together with an
aggregation association (multiplicity of one) to the afore-mentioned item class.

It should be noted that there is some information loss in terms of sequencing when moving from a BNF to a UML representation. This is further to the loss (in this particular work) when moving from UML to O-Telos due to our not explicitly re-stating assertions for multiplicity, exclusive-or, etc.

4.5.3.2 SPARK Ada Meta-model Definitions

In this subsection we present a series of schemas representing elements for a SPARK Ada structure (an instantiation of the DevelopmentStructure metaclass) capturing the language subset outlined above. Each element is systematically expressed in terms of its source BNF syntax, corresponding UML representation and implementation in O-Telos using ConceptBase; OCL constraints are also introduced where appropriate to supplement standard UML diagramming conventions. We further highlight the influence of our modelling philosophy (from 4.5.3.1) in deriving these structures. Readers without an Ada background are again referred to Barnes (1997) for details on the featured syntax.

Many of the SPARK BNF productions that follow are identical in Ada. However some - those indicated by an asterisk (*) - have been modified despite retaining, the Ada category name. Others - those highlighted with a plus sign (+) - are additional SPARK constructs not present in the core language.

We begin by describing the library_item syntax which marks the starting point for our structure. A SPARK Ada program is normally written as a number of units that may be compiled and linked separately, or else grouped together into one or more compilations. A compilation is therefore a succession of compilation units, of which one permutation comprises a library_item and a context clause specifying interdependence among individual units. For information and to highlight the origin for our modelling, the related BNF syntax is shown below:

```
library_item :: = (compilation-unit)

compilation_unit ::= context_clause library_item | context_clause subunit

* library_item ::= [private] package_declaration | package_body | main_subprogram

+ main_subprogram ::= [inherit_clause] main_subprogram_annotation subprogram_body

+ main_subprogram_annotation ::= --# main_program;
```

- **library_item**

A library_item (figure 4.25) contains either the optional 'private' (private_token) reserved word, together with a package declaration (package declaration), or else a package body (package_body) or main

---

36 As with project NATURE, we make the assumption that issues of layout are addressed by user interfaces to this and all structures featured in the thesis.
37 The SPARK DevelopmentStructure class (not shown) is simply an aggregation of all elements featured in this section.
38 OCL is mainly used to state mutual dependencies (among optional categories and tokens), as well in place of standard diagram annotations for multiplicity in complex cases (where for example elements are nested).
subprogram (main_subprogram). Note how we promote private_token to a class in UML in order to specify its optionality, as per rule 6b (from subsection 4.5.3.1.1); rule 8b also applies.

<table>
<thead>
<tr>
<th>BNF Syntax</th>
<th>OCL</th>
<th>O-Telos (base classes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>library_item ::= [private] package_declaration l package_body l main_subprogram</td>
<td>library_item Invariant self.allinstances-&gt;forall(p</td>
<td>library_item in StructureElement, SimpleClass isA AerospaceEngineeringObject with has_property private_token : private_token has_part package_declaration : package_declaration; package_body : package_body; main_subprogram : main_subprogram end private_token in ArtifactProperty, SimpleClass isA String with constraint private_Is_Private: $ forall p/private_token p = &quot;private&quot; $ end</td>
</tr>
<tr>
<td>constraint (in conjunction with 'exclusive-or' in figure) ensuring private_token can exist only when package_declaration exists</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.25 - 'library_item schema'

All remaining featured elements enable representation of package declarations, the syntax for which is:

package_declaration ::= package_specification;

*package_specification ::=  
  [inherit_clause]  
  package defining_program_unit_name package_annotation  
  is  
  {relisting_declarative_item}  
  end [parent_unit_name .] identifier;

Note, again the above syntax is for reader orientation only; individual schema representations will now
be developed for each of these constructs (except renaming_declaration).

- **package_declaration**
  Each package_declaration is defined in terms of a single package specification (package_specification) and terminates with a semi-colon (semi_colon_token) as the schema below (figure 4.26) demonstrates:

  ![Package Declaration Schema](image)

  **BNF Syntax**
  ```plaintext
  package_declaration ::= package_specification;
  ```

  **O-Telos**
  ```plaintext
  package_declaration in StructureElement,
  SimpleClass isA AerospaceEngineeringObject
  with
  has_property
  semi_colon_token : semi_colon_token
  has_part
  package_specification
  end
  ```

  ```plaintext
  semi_colon_token in ArtifactProperty,
  SimpleClass isA String with constraint
  semicolon_Is_Semicolon:
  $ forall s/semi_colon_token
  s = ";" $ 
  end
  ```

  **Figure 4.26 - ‘package_declaration schema’**

- **package_specification**
  In addition to the ‘package’ (package_token), ‘is’ (is_token) and ‘end’ (end_token) reserved words, and semi_colon_token and period (period_token) punctuation, package_specification includes the inherit clause (inherit_clause), defining program unit name (defining_program_unit_name), package annotation (package_annotation), renaming declaration (renaming_declaration), package declarative item (package_declarative_item), parent unit name (parent_unit_name) and identifier category elements.

  In UML, we again promote an attribute - period_token - to a class in order to express its optionality (rule 6b); the zero-or-one (0..1) multiplicity constraint on this class and on parent_unit_name is supplemented by an OCL constraint to preserve the BNF optionality syntax such that both or neither of these elements exist. Further, both parent_unit_name and identifier are treated as atomic (and hence as specialisations of the built-in String class which is not shown on the UML diagram, but does feature in the corresponding O-Telos representation) since this granularity is sufficient for traceability purposes (as per rule 3). The package_specification schema is shown in figure 4.27.
Structuring Development Artifacts

**BNF Syntax**

```plaintext
* package_specification ::=  
  [inherit_clause]  
  package_defining_program_unit_name  
  package_annotation  
  is  
  {renaming_declaration}  
  {package_declarative_item}  
  end [parent_unit_name .] identifier;  
parent_unit_name ::= name  
identifier ::= identifier_letter {{underlined} letter_or_digit}
```

**OCL**

```plaintext
package_specification invariant  
self.allInstances->forAll(p|  
  (p.package_token->isEmpty and p.parent_unit_name->isEmpty) xor  
  (p.package_token->notEmpty and p.parent_unit_name->notEmpty))  
has_property  
  package_token : package_token;  
is_token : is_token;  
end_token : end_token;  
period_token : period_token;  
semi_colon_token : semi_colon_token  
has_part  
  inherit_clause : inherit_clause;  
  defining_program_unit_name ;  
  package_annotation :  
  package_declarative_item ;  
  package_declarative_item;  
  parent_unit_name : parent_unit_name;  
  identifier : identifier
```

**O-Telos (continued)**

```plaintext
end_token in ArtifactProperty, SimpleClass  
isA String with constraint end_Is_End:  
$forall e/end_token e = *end* $  
end  
period_token in ArtifactProperty, SimpleClass isA String with constraint  
period_Is_Period:  
$forall p/period_token p = .* $  
end  
parent_unit_name in StructureElement, SimpleClass isA String, AerospaceEngineeringObject  
end  
identifier in StructureElement, SimpleClass isA String, AerospaceEngineeringObject end
```

**O-Telos**

```plaintext
--package_specification class  
package_specification in StructureElement, SimpleClass isA AerospaceEngineeringObject  
with  
has_property  
  package_token : package_token;  
is_token : is_token;  
end_token : end_token;  
period_token : period_token;  
semi_colon_token : semi_colon_token  
has_part  
  inherit_clause : inherit_clause;  
  defining_program_unit_name ;  
  package_annotation :  
  package_declarative_item ;  
  package_declarative_item;  
  parent_unit_name : parent_unit_name;  
  identifier : identifier
```

**Figure 4.27 - 'package_specification schema'**

169
• **inherit_clause**

This construct includes the '--# inherit' (inherit_token) reserved word annotation together with at least one, but optionally a comma delimited list of package name (package_name) elements. It too terminates with a semi-colon.

In accordance with rule 9, our UML representation (and hence the O-Telos implementation) introduces an additional class - package_name_list_item - as a means of representing elements following the mandatory initial package_name instance. Further, package_name is treated as atomic in accordance with rule 3. The inherit_clause schema is shown in figure 4.28.

**BNF Syntax**

\[+	ext{inherit_clauses} ::= \]
\[-->\text{# inherit} \text{package_names}\{, \text{package_names}\};\]
\[\text{package_names} ::= \text{identifier}\]

**O-Telos**

```
inherit_clause in StructureElement, SimpleClass isA AerospaceEngineeringObject with
  has_property
    inherit_token : inherit_token;
    semi_colon_token : semi_colon_token
  has_part
    package_name : package_name;
    package_name_list_item : package_name_list_item
end

inherit_token in ArtifactProperty, SimpleClass isA String with
  constraint
    inherit_Is_Inherit:
      $ forall i/inherit_token
        i = "--># inherit"$
end

package_name in StructureElement, SimpleClass isA String, AerospaceEngineeringObject end

package_name_list_item in StructureElement, SimpleClass isA AerospaceEngineeringObject with
  has_property
    comma_token : comma_token
  has_part
    package_name : package_name
end
```

**Figure 4.28 - 'inherit_clause schema'**
- **defining_program_unit_name**

A defining_program_unit_name (figure 4.29) contains defining identifier (defining_identifier) and parent_unit_name elements, together with the period_token punctuation.

The UML representation of this category includes certain characteristics present in package_specification. Again, period_token is promoted to a class to allow specification of optionality (as per rule 6b); again, the zero-or-one multiplicity imposed on this class and on parent_unit_name is supplemented by an OCL constraint to preserve the BNF optionality syntax such that both or neither of these elements exists; and again defining_identifier is treated as atomic.

### Figure 4.29 - ‘defining_program_unit_name schema’

<table>
<thead>
<tr>
<th>BNF Syntax</th>
<th>OCL</th>
<th>O-Telos (base classes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>defining_program_unit_name ::= [parent_unit_name .]</td>
<td>defining_program_unit_name Invariant self.allInstances-&gt;forall(d</td>
<td>defining_program_unit_name in StructureElement, SimpleClass isA AerospaceEngineeringObject with has_property period_token : period_token has_part parent_unit_name : parent_unit_name; defining_identifier : defining_identifier end defining_identifier in StructureElement, SimpleClass isA String, AerospaceEngineeringObject end</td>
</tr>
<tr>
<td>defining_identifier ::= identifier</td>
<td>Constraint preserving the BNF optionality syntax such that both period_token and parent_unit_name exist or neither exists</td>
<td></td>
</tr>
</tbody>
</table>

- **package_annotation**

This construct comprises an optional own variable clause (own_variable_clause); if this element exists, then it may also contain an optional initialization specification (initialization_specification). The semantics of this statement are captured in BNF by nesting the square brackets framing these constructs and in the corresponding UML diagram using an OCL constraint (see figure 4.30).

- **own_variable_clause**

An own_variable_clause includes the ‘--# own’ (own_token) reserved word annotation, together with a
Towards a SPARK Ada Programming Language Structure

Figure 4.30 - ‘package_annotation schema’

<table>
<thead>
<tr>
<th>BNF Syntax</th>
<th>OCL</th>
<th>O-Telos (base classes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ package_annotation ::= [own_variable_clause [initialization_specification]]</td>
<td>package_annotation invariant self.allinstances-&gt;forall(p</td>
<td>p.own_variable_clause-&gt;size &lt;=1 and p.initialization_specification-&gt;size &lt;=1 and not (p.initialization_specification-&gt;size = 1 and p.own_variable_clause-&gt;isEmpty))</td>
</tr>
</tbody>
</table>

Constraint preserving the BNF nested square brackets syntax

Figure 4.31 - ‘own_variable_clause schema’

<table>
<thead>
<tr>
<th>BNF Syntax</th>
<th>O-Telos</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ own_variable_clause ::= --# own own_variable_list;</td>
<td>own_variable_clause in StructureElement, SimpleClass isA AerospaceEngineeringObject with has_property own_variable_token : own_variable_token; semi_colon_token : semi_colon_token has_part own_variable_list : own_variable_list end own_variable_token in ArtifactProperty, SimpleClass isA String with constraint own_is_Own: $forall v/own_variable v = &quot;--# own&quot;$ end</td>
</tr>
</tbody>
</table>

172
single own variable list (own_variable_list) element; the construct terminates with a semi Colon. This schema is shown in figure 4.31.

- own_variable_list
As its name suggests, own_variable_list (figure 4.32) contains one or more own variable (own_variable) elements as a comma-delimited list.

Again (in accordance with rule 9), the UML and O-Telos representations introduce an additional class - own_variable_list_item - to represent optional own_variable elements that may follow the single mandatory instance; own_variable itself is treated as atomic (as per rule 3).

<table>
<thead>
<tr>
<th>own_variable_list</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;Structure Element&gt;&gt;</td>
</tr>
<tr>
<td>own_variable_list</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>&lt;&lt;Structure Element&gt;&gt;</td>
</tr>
<tr>
<td>own_variable</td>
</tr>
<tr>
<td>own_variable_list_item</td>
</tr>
</tbody>
</table>

BNF Syntax

```
+ own_variable_list ::= 
  own_variable { , own_variable }
+ own_variable ::= identifier
```

O-Telos

```
own_variable_list in StructureElement, 
  SimpleClass isA AerospaceEngineeringObject 
  with has_part 
    own_variable : own_variable; 
    own_variable_list_item : 
    own_variable_list_item 
  end 

  own_variable_list_item in StructureElement, 
  SimpleClass isA AerospaceEngineeringObject 
  with has_property 
    comma_token : comma_token 
    has_part 
    own_variable : own_variable 
  end 

  own_variable in StructureElement, 
  SimpleClass isA String, 
  AerospaceEngineeringObject end
```

Figure 4.32 - ‘own_variable_list schema’

- initialization_specification
An initialization_specification (figure 4.33) includes the ‘--# initializes’ (initializes_token) reserved word annotation, together with a single own_variable_list element; this construct also terminates with a semi Colon.

- package_declarative_item
A package_declarative_item contains either a basic declarative item (basic_declarative_item), subprogram declaration (subprogram_declaration) or external subprogram declaration (external_subprogram_declaration); therefore rule 8a applies. The corresponding schema is shown in figure 4.34.
initialization_specification

<<Structure Element>>

1 initialization_specification

<<Structure Element>>

1 initialization_token: initialization_token

<<Structure Element>>

1 semi_colon_token: semi_colon_token

<<Structure Element>>

own_variable_list

BNF Syntax

+ initialization_specification ::= 
  "# initializes own_variable_list;

O-Telos

initialization_specification in StructureElement, SimpleClass isA AerospaceEngineeringObject with has_property initialization_token: initialization_token;
  semi_colon_token: semi_colon_token
  has_part own_variable_list : own_variable_list
end

initialization_token in ArtifactProperty, SimpleClass isA String with constraint init_Is_Init:
  $ forall i/initialization_token i = "# initializes" $
end

Figure 4.33 - ‘initialization_specification schema’

package_declarative_item

<<Structure Element>>

package_declarative_item

<<Structure Element>>

1 basic_declarative_item

<<Structure Element>>

1 subprogram_declaration

<<Structure Element>>

1 external_subprogram_declaration

BNF Syntax

+ package_declarative_item ::= 
  basic_declarative_item | subprogram_declaration | external_subprogram_declaration

O-Telos

package_declarative_item in StructureElement, SimpleClass isA AerospaceEngineeringObject with has_part 
  basic_declarative_item : basic_declarative_item;
  subprogram_declaration : subprogram_declaration;
  external_subprogram_declaration : external_subprogram_declaration
end

Figure 4.34 - ‘package_declarative_item schema’
- **basic_declarative_item**
  Each basic_declarative_item (figure 4.35) consists of either a representation clause (representation_clause) or basic declaration (basic_declaration); again rule 8a applies.

![Diagram of basic_declarative_item](image)

<table>
<thead>
<tr>
<th>BNF Syntax</th>
<th>O-Telos</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic_declarative_item ::=</td>
<td>basic_declarative_item in StructureElement, SimpleClass isA AerospaceEngineeringObject with has_part basic_declaration : basic_declaration; representation_clause : representation_clause end</td>
</tr>
<tr>
<td>basic_declaration</td>
<td>representation_clause</td>
</tr>
</tbody>
</table>

Figure 4.35 - ‘basic_declarative_item schema’

- **basic_declaration**
  Similarly, each basic_declaration comprises either an object declaration (object_declaration), a number declaration (number_declaration), a type declaration (type_declaration) or a subtype declaration (subtype_declaration). This is represented by the schema in figure 4.36 (again based on rule 8a).

- **type_declaration**
  A type_declaration features either a full type declaration (full_type_declaration) or private type declaration (private_type_declaration); the corresponding schema (derived using rule 8a) is shown in figure 4.37.

- **full_type_declaration**
  Each full_type_declaration contains the ‘type’ (type_token) and ‘is’ (is_token) reserved words, together with a defining_identifier and type definition (type_definition) as shown in figure 4.38; full_type_declaration terminates with a semi-colon.

- **type_definition**
  Each type_definition consists of an enumeration type definition (enumeration_type_definition), an integer type definition (integer_type_definition), a real type definition (real_type_definition), an array type definition (array_type_definition) or a record type definition (record_type_definition); this schema is also based on rule 8a and is shown in figure 4.39.
Towards a SPARK Ada Programming Language Structure

Figure 4.36 - 'basic_declaration schema'

Figure 4.37 - 'type_declaration schema'
Structuring Development Artifacts

### full_type_declaration

<table>
<thead>
<tr>
<th><strong>BNF Syntax</strong></th>
<th><strong>O-Telos</strong></th>
</tr>
</thead>
</table>
| * full_type_declaration ::=  
  type defining_identifier is type_definition; | full_type_declaration in StructureElement,  
  SimpleClass isA AerospaceEngineeringObject  
  with  
  has_property  
  type_token : type_token;  
  is_token : is_token;  
  semi_colon_token : semi_colon_token  
  has_part  
  defining_identifier :  
  defining_identifier;  
  type_definition : type_definition  
  end  
  type_token in ArtifactProperty,  
  SimpleClass isA String  
  with constraint  
  type_Is_Type:  
  $forall t/ type_token  
  t = "type" $  
  end |

**Figure 4.38 - ‘full_type_declaration schema’**

- **enumeration_type_definition**
  An enumeration_type_definition consists of one or more defining_identifier elements represented as a comma-delimited list; the construct is enclosed between left (left_parenthesis_token) and right (right_parenthesis_token) parentheses.

Again (as per rule 9), the UML and O-Telos representations introduce an additional class - defining_identifier_list_item - to represent optional defining_identifier elements following the single mandatory instance. Figure 4.40 depicts the corresponding enumeration_type_definition schema.

- **subprogram_declaration**
  A subprogram_declaration consists of either a procedure specification (procedure_specification) and procedure annotation (procedure_annotation) or a function specification (function_specification) and function annotation (function_annotation), separated by a semi-colon.

The common semi_colon_token element is included once as an attribute of the subprogram_declaration class (in accordance with rule 8b); an OCL constraint describes the alternative combinations (see figure 4.41).
Towards a SPARK Ada Programming Language Structure

**type_definition**

```
<<Structure Element>>
  type_definition

<<Structure Element>>
  enumeration_type_definition

<<Structure Element>>
  Integer_type_definition

<<Structure Element>>
  real_type_definition

<<Structure Element>>
  array_type_definition

<<Structure Element>>
  record_type_definition
```

**BNF Syntax**

```
* type_definition ::=  
enumeration_type_definition |  
integer_type_definition | real_type_definition |  
array_type_definition | record_type_definition
```

**O-Telos**

```
type_definition in StructureElement,  
SimpleClass isA AerospaceEngineeringObject  
with has_part  
enumeration_type_definition;  
enumeration_type_definition;  
integer_type_definition;  
integer_type_definition;  
real_type_definition;  
real_type_definition;  
array_type_definition;  
array_type_definition;  
record_type_definition;  
record_type_definition  
end
```

**Figure 4.39 - 'type_definition schema'**

- **procedure_specification**
  Each procedure_specification (figure 4.42) consists of the 'procedure' (procedure_token) reserved word, followed by defining_identifier and parameter profile (parameter_profile) elements.

- **parameter_profile**
  A parameter_profile simply comprises an optional formal part (formal_part) element (see figure 4.43).

- **formal_part**
  The formal_part element (figure 4.44) consists of one or more parameter specifications (parameter_specification) represented as a semi-colon-delimited list between left (left_parenthesis_token) and right (right_parenthesis_token) parentheses.

As with all featured list based constructs, the UML and O-Telos representations introduce an additional class (in accordance with rule 9) - in this case parameter_specification_list_item - to represent optional parameter_specification elements following the single mandatory instance.
Structuring Development Artifacts

Figure 4.40 - 'enumeration_type_definition schema'

- parameter_specification
A parameter_specification contains a defining identifier list (defining_identifier_list), the colon punctuation (colon_token), mode and subtype mark (subtype_mark) elements.

In the UML model, mode and subtype_mark are both treated as specialisations of the String class as per rule 3; the former bears an OCL constraint restricting instantiations to the BNF alternatives shown in figure 4.45.

```
<table>
<thead>
<tr>
<th>BNF Syntax</th>
<th>O-Telos</th>
</tr>
</thead>
<tbody>
<tr>
<td>* enumeration_type_definition ::= (defining_identifier , defining_identifier)</td>
<td>enumeration_type_definition in StructureElement, SimpleClass isA AerospaceEngineeringObject with</td>
</tr>
<tr>
<td></td>
<td>has_property</td>
</tr>
<tr>
<td></td>
<td>left_parenthesis_token : left_parenthesis_token;</td>
</tr>
<tr>
<td></td>
<td>right_parenthesis_token : right_parenthesis_token;</td>
</tr>
<tr>
<td></td>
<td>has_part</td>
</tr>
<tr>
<td></td>
<td>defining_identifier : defining_identifier; defining_identifier_list_item : defining_identifier_list_item end</td>
</tr>
<tr>
<td></td>
<td>left_parenthesis_token in ArtifactProperty, SimpleClass isA String with</td>
</tr>
<tr>
<td></td>
<td>constraint left_Is_Left:</td>
</tr>
<tr>
<td></td>
<td>$ forall 1/left_parenthesis_token 1 = &quot;(&quot; $</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td></td>
<td>right_parenthesis_token in ArtifactProperty, SimpleClass isA String with</td>
</tr>
<tr>
<td></td>
<td>constraint right_Is_Right:</td>
</tr>
<tr>
<td></td>
<td>$ forall r/right_parenthesis_token r = &quot;)&quot; $</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td></td>
<td>defining_identifier_list_item in StructureElement, SimpleClass isA AerospaceEngineeringObject with</td>
</tr>
<tr>
<td></td>
<td>has_property</td>
</tr>
<tr>
<td></td>
<td>comma_token : comma_token;</td>
</tr>
<tr>
<td></td>
<td>has_part</td>
</tr>
<tr>
<td></td>
<td>defining_identifier : defining_identifier end</td>
</tr>
</tbody>
</table>
```
Towards a SPARK Ada Programming Language Structure

**Figure 4.41 - 'subprogram_declaration schema'**

**Figure 4.42 - 'procedure_specification schema'**
### Structuring Development Artifacts

#### parameter_profile

<table>
<thead>
<tr>
<th>BNF Syntax</th>
<th>O-Telos</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter_profile ::= [formal_part]</td>
<td>parameter_profile in StructureElement, SimpleClass isA AerospaceEngineeringObject with has_part formal_part : formal_part end</td>
</tr>
</tbody>
</table>

**Figure 4.43 - 'parameter_profile schema'**

#### formal_part

<table>
<thead>
<tr>
<th>BNF Syntax</th>
<th>O-Telos</th>
</tr>
</thead>
<tbody>
<tr>
<td>formal_part ::= (parameter_specification ; parameter_specification)</td>
<td>formal_part in StructureElement, SimpleClass isA AerospaceEngineeringObject with has_property left_parenthesis_token : left_parenthesis_token; right_parenthesis_token : right_parenthesis_token has_part parameter_specification : parameter_specification; parameter_specification_list_item : parameter_specification_list_item end parameter_specification_list_item in StructureElement, SimpleClass isA AerospaceEngineeringObject with has_property semiColon_token : semiColon_token has_part parameter_specification : parameter_specification end</td>
</tr>
</tbody>
</table>

**Figure 4.44 - 'formal_part schema'**
Towards a SPARK Ada Programming Language Structure

**parameter specification**

```
<<Structure Elements>>
parameter_specification
   colon_token : colon_token

<<Structure Elements>>
defining_identifier_list

<<Structure Elements>>
mode

<<Structure Elements>>
subtype_mark
```

**BNF Syntax**

```
* parameter_specification ::= parameter_specification in
defining_identifier_list : mode subtype_mark

mode ::= in | in out | out
defining_identifier_list ::= name

subtype_mark ::= name
```

**O-Telos**

```
parameter_specification in StructureElement, SimpleClass isA AerospaceEngineeringObject with
   has_property
      colon_token : colon_token
   has_part
      defining_identifier_list :
         defining_identifier_list;
         mode : mode;
         subtype_mark : subtype_mark
end

colon_token in ArtifactProperty,
SimpleClass isA String with constraint
   colon_is_Colon:
      $forall c/colon_token
      c =$
end

mode in StructureElement, SimpleClass isA String, AerospaceEngineeringObject with
   constraint
   enum_Mode:
      $forall m/mode
         (m = "in") or
         (m = "in out") or
         (m = "out")$
end

subtype_mark in StructureElement,
SimpleClass isA String,
AerospaceEngineeringObject
end
```

**Figure 4.45** - ‘parameter_specification schema’

- **defining_identifier_list**

The defining_identifier_list (figure 4.46) consists of one or more comma-delimited defining identifiers (defining_identifier).

As per rule 9, the UML and O-Telos representations again employ an additional class - defining_identifier_list_item - to represent optional defining_identifier elements following the single mandatory instance.

- **procedure_annotation**

A procedure_annotation features an optional global definition (global_definition) or moded global
Structuring Development Artifacts

defining_identifier_list

<<Structure Element>>
defining_identifier_list

<<Structure Element>>
defining_identifier

<<Structure Element>>
defining_identifier_list_item

BNF Syntax

defining_identifier_list ::= defining_identifier {, defining_identifier}

O-Telos

defining_identifier_list in StructureElement, SimpleClass isA AerospaceEngineeringObject with
has_part
defining_identifier :
defining_identifier;
defining_identifier_list_item :
defining_identifier_list_item_item
end

Figure 4.46 - 'defining_identifier_list schema'

procedure_annotation

<<Structure Element>>
procedure_annotation

<<Structure Element>>
global_definition

<<Structure Element>>
moded_global_definition

<<Structure Element>>
dependency_relation

BNF Syntax

+ procedure_annotation ::= [global_definition | moded_global_definition] [dependency_relation]

O-Telos

procedure_annotation in StructureElement, SimpleClass isA AerospaceEngineeringObject with
has_part
global_definition : global_definition;
moded_global_definition :
moded_global_definition;
dependency_relation :
dependency_relation
end

Figure 4.47 - 'procedure_annotation schema'
Towards a SPARK Ada Programming Language Structure

definition (moded_global_definition), and an optional dependency relation (dependency_relation).

As per rule 8a, an exclusive-or annotation and dashed line connecting global_definition and moded_global_definition expresses the alternatives semantics for this element (figure 4.47).

- moded_global_definition
A moded_global_definition contains the '--# global' (global_token) reserved word annotation, together with global mode (global_mode) and entire variable list (entire_variable_list) elements, followed by a semi-colon. This sequence exists once, but can be repeated (--# global annotation apart).

Modelling this repetition in UML (and hence O-Telos) requires introduction of a moded_global_definition_list_item class (as per rule 9) comprising the elements described above.

The global_mode element is defined as a specialisation of the String class (as per rule 3), with an OCL constraint restricting its instantiation to the BNF alternatives shown in figure 4.48.

- entire_variable_list
An entire_variable_list (figure 4.49) consists of one or more comma-delimited entire variables (entire_variable).

Again, in accordance with rule 9, the UML and O-Telos representations introduce an additional class - entire_variable_list_item - to represent optional entire_variable elements following the single mandatory instance.

- entire_variable
The entire_variable element (figure 4.50) consists of direct name (direct_name) and package_name elements, together with the period_token punctuation.

In accordance with rule 6b, period_token is promoted to a class allowing specification of optionality; again, the zero-or-one multiplicity imposed on this class and on package_name is supplemented by an OCL constraint to preserve the BNF optionality syntax such that both or neither of these elements exist. Note also that direct_name is treated as atomic (as per rule 3).

- dependency_relation
A dependency_relation comprises the '--# derives' (derives_token) reserved word annotation, together with zero or more ampersand-delimited dependency clause (dependency_clause) elements; the construct terminates with a semi-colon.

Once again, following rule 9, the UML and O-Telos representations introduce an additional class - dependency_clause_list_item - to represent dependency_clause elements following the initial instance. However, unlike similar list-base constructs making up the SPARK model (cf. own_variable_list,
Structuring Development Artifacts

**moded_global_definition**

<table>
<thead>
<tr>
<th>BNF Syntax</th>
<th>O-Telos</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ moded_global_definition ::=</td>
<td>moded_global_definition in</td>
</tr>
<tr>
<td>--# global global_mode entire_variable_list;</td>
<td>StructureElement, SimpleClass isA</td>
</tr>
<tr>
<td>{global_mode entire_variable_list;}</td>
<td>AerospaceEngineeringObject</td>
</tr>
<tr>
<td>+ global_mode ::= in</td>
<td>lin</td>
</tr>
<tr>
<td>global_token : global_token;</td>
<td></td>
</tr>
<tr>
<td>semi_colon_token : semi_colon_token</td>
<td>with constraint global_Is_Global:</td>
</tr>
<tr>
<td>has_part</td>
<td>$ forall g/global_token</td>
</tr>
<tr>
<td>global_mode : global_mode;</td>
<td>g = &quot;--# global&quot; $</td>
</tr>
<tr>
<td>entire_variable_list :</td>
<td>end</td>
</tr>
<tr>
<td>entire_variable_list;</td>
<td>global_token in ArtifactProperty,</td>
</tr>
<tr>
<td>moded_global_definition_list_item :</td>
<td>SimpleClass isA String</td>
</tr>
<tr>
<td>moded_global_definition_list_item</td>
<td>with constraint enum_Global_Mode:</td>
</tr>
<tr>
<td>end</td>
<td>for all g/global_mode</td>
</tr>
<tr>
<td>g = &quot;in&quot; or</td>
<td>(g = &quot;in out&quot;) or</td>
</tr>
<tr>
<td>g = &quot;out&quot;)$</td>
<td>end</td>
</tr>
</tbody>
</table>

**Figure 4.48 - 'moded_global_definition schema'**
Towards a SPARK Ada Programming Language Structure

Figure 4.49 - 'entire_variable_list schema'

Figure 4.50 - 'entire_variable schema'
Structuring Development Artifacts

enumeration_type_definition), a dependency_relation list can be empty; we therefore specify an OCL constraint ensuring that if dependency_clause is empty, then dependency_clause_list_item is also empty.

The dependency_relation schema appears in figure 4.51.

**BNF Syntax**

```
+ dependency_relation ::=  
                        --# derives [dependency_clause  
                        {& dependency_clause}];
```

**OCL**

```
dependency_relation invariant
set.allInstances->forall(d | 
not(d.dependency_clause->isEmpty and 
d.dependency_clause_list_item->notEmpty))
```

*Constraint preserving the BNF optionality syntax*

```
dependency_relation in 
StructureElement, 
SimpleClass isA 
AerospaceEngineeringObject 
with 
hs_property 
derives_token : 
derives_token; 
semi_colon_token : 
semi_colon_token 
has_part 
dependency_clause : 
dependency_clause; 
dependency_clause_list_item : 
dependency_clause_list_item 
end
```

```
derives_token in 
ArtifactProperty, 
SimpleClass isA String 
with 
constraint 
derives_Is_Derives: 
$forall$ 
d/derives_token d = "--# 
derives"$ 
end
```

```
dependency_clause_list_item in 
StructureElement, 
SimpleClass isA 
AerospaceEngineeringObject 
with 
hs_property 
ampersand_token : 
ampersand_token 
has_part 
dependency_clause : 
dependency_clause 
end
```

```
ampersand_token in 
ArtifactProperty, 
SimpleClass isA String 
with 
constraint 
ampersand_Is_Ampersand: 
$forall$ 
a/ampersand_token a = "&"$ 
end
```

**Figure 4.51 - ‘dependency_relation schema’**
- dependency_clause

A dependency_clause is defined in terms of the 'from' (from_token) reserved word which separates entire_variable_list and imported_variable_list (imported_variable_list) elements. The schema for this construct is as follows (figure 4.52):

![Dependency Clause Diagram]

**BNF Syntax**

```
+ dependency_clause ::= entire_variable_list from imported_variable_list
```

**O-Telos**

```o-telos
dependency_clause in StructureElement,
SimpleClass isA AerospaceEngineeringObject
with
  has_property
  from_token : from_token
  has_part
  entire_variable_list :
  entire_variable_list;
  imported_variable_list :
  imported_variable_list
end

from_token token in ArtifactProperty,
SimpleClass isA String
with
constraint
from_Is_From:
  $ forall f/from_token
  f = "from" $
end
```

Figure 4.52 - 'dependency_clause schema'

- imported_variable_list

Finally, imported_variable_list contains either an optional asterisk '*' (asterisk_token), or an optional asterisk-comma '*', (asterisk_comma_token) and an entire_variable_list element.

As per rule 6b, asterisk_token and asterisk_comma_token are promoted to classes allowing their optionality and exclusivity to be described. This is done using an OCL constraint (according to rule 8b), as the schema in figure 4.53 illustrates.
Structuring Development Artifacts

**BNF Syntax**

+ imported_variable_list ::= [*] |[/,*] entire_variable_list

**OCL**

imported_variable_list invariant
self.allInstances->forall(i |
(L::asterisk_token)->isEmpty xor
L::asterisk_comma_token->isEmpty and
L::entire_variable_list->isEmpty)
xor
(L::asterisk_token)->isEmpty and
(L::asterisk_comma_token)->isEmpty xor
L::entire_variable_list->size = 1)
end

**O-Telos (base classes)**

imported_variable_list in StructureElement,
SimpleClass isA AerospaceEngineeringObject
with
has_property
asterisk_token :
asterisk_comma_token :
has_part
entire_variable_list :
end

asterisk_token in ArtifactProperty,
SimpleClass isA String with constraint
asterisk_Is_Asterisk:
$forall
a/asterisk_token a = "*"$
end

asterisk_comma_token in ArtifactProperty,
SimpleClass isA String with constraint
asterisk_comma_Is_Asterisk_Comma:
$forall
a/asterisk_comma_token a = ","$
end

Figure 4.53 - ‘imported_variable_list schema’

189
4.5.3.3 SPARK Ada Meta-model: Worked Examples

This subsection applies the SPARK Ada structure developed above to two program fragments (taken from the appendix to Barnes - 1997) from an autopilot system controlling altitude and heading for a hypothetical aircraft. The autopilot includes a control panel containing three binary-state switches with positions on and off. A package declaring access to the switches as a private child of the main autopilot package AP is specified as follows:

```ada
private package AP.Controls
--# own Master_Switch, Altitude_Switch, Heading_Switch;
--# initializes Master_Switch, Altitude_Switch, Heading_Switch;
is
type Switch is (On, Off);

procedure Read_Master_Switch(Position: out Switch);
--# global in out Master_Switch;
--# derives Position, Master_Switch from Master_Switch;

procedure Read_Altitude_Switch(Position: out Switch);
--# global in out Altitude_Switch;
--# derives Position, Altitude_Switch from Altitude_Switch;

procedure Read_Heading_Switch(Position: out Switch);
--# global in out Heading_Switch;
--# derives Position, Heading_Switch from Heading_Switch;
end AP.Controls;
```

We now present the O-Telos representation for this code fragment:

```otelos
Representation of the 'private' library-item clause
ap_controls_library_item in library_item, Token with private_token
private_token : "private"
package_declaration
packageDeclaration : ap_controls_package_declaration
end

Representation of the AP.Controls package
ap_controls_package_declaration in package_declaration, Token with package_specification
packageSpecification : ap_controls_package_specification
end

ap_controls_package_specification in package_specification, Token with package_token
packageToken : "package"
defining_program_unit_name
definingProgramUnitName : ap_controls_defining_program_unit_name
package_annotation
packageAnnotation : 
```

39 We make the observation that while fifteen lines of SPARK has yielded five and a half pages of O-Telos code, there is a 1:1 mapping between the underlying BNF syntactic language elements and schemas for these syntactic elements.
Structuring Development Artifacts

ap_controls_package_annotation
  is_token
    isToken : "is"
  package_declarative_item
    packageDeclarativeItem1 : type_package_declarative_item;
    packageDeclarativeItem2 : procl_package_declarative_item;
    packageDeclarativeItem3 : procl2_package_declarative_item;
    packageDeclarativeItem4 : procl3_package_declarative_item
end_token
  endToken : "end"
  parent_unit_name
    parentUnitName : "AP"
  period_token
    periodToken : "."
  identifier
    definingIdentifier : "Controls"
  semi_colon_token
    semiColonToken : ";"
end

Representation of the AP.Controls package name

ap_controls_defining_program_unit_name
in defining_program_unit_name, Token with
  parent_unit_name
    parentUnitName : "AP"
  period_token
    periodToken : "."
  defining_identifier
    definingIdentifier : "Controls"
end

Representation of package annotations

ap_controls_package.annotation in package_annotation, Token with
  own_variable_clause
    ownVariableClause :
      ap_controls_own_variable_clause
        initialization_specification:
          initializes_annotation_ap_controls_own_variable_list
            semi_colon_token
              semiColonToken : ";"
end

Representation of the `--# own' package annotation

ap_controls_own_variable_clause in own_variable_clause, Token with
  own_variable
    ownVariable :
      initializes_altitude_switch own_variable_list
        initializes_annotation_ap_controls_own_variable_list
          semi_colon_token
            semiColonToken : ";"
end

Representation of the Switch type declaration

type_package_declarative_item in package_declarative_item, Token with
  basic_declarative_item
    switch_basic_declarative_item:
      switch_basic_declarative_item
        end
Towards a SPARK Ada Programming Language Structure

basic_declarative_item, Token with
basic_declaration
  basicDeclaration :
    switch_basic_declaration
end

switch_basic_declaration in
basic_declaration, Token with
type_declaration
typeDeclaration :
switch_type_declaration
end

switch_type_declaration in
type_declaration, Token with
full_type_declaration
fullTypeDeclaration :
switch_full_type_declaration
end

switch_full_type_declaration in
full_type_declaration, Token with
type_token
typeToken : "type"
defining_identifier
definingIdentifier : "Switch"
is_token
isToken : "is"
type_definition
typeDefinition : switch_positions
semi_colon_token
semiColonToken : ":;"
end

switch_positions in type_definition,
Token with
enumeration_type_definition
eNum_switch_positions : enum Switch
end

eNum_switch_positions in
enumeration_type_definition, Token with
left_parenthesis
leftParenthesis : "("
defining_identifier
definingIdentifier : "On"
defining_identifier_list
definingIdentifierList : switch_defining_identifier_list_item
right_parenthesis
rightParenthesis : ")"
end

switch_defining_identifier_list_item in
defining_identifier_list_item, Token with
comma_token
commaToken : ","
defining_identifier
definingIdentifier : "Off"
end


definition of the
Read_Master_Switch procedure

proc1_package_declarative_item in
package_declarative_item, Token with
subprogram_declaration
subprogramDeclaration :
read_master_switch_subprogram_declaration
end

read_master_switch_subprogram_declaration in
subprogram_declaration, Token with
procedure_specification
ProcedureSpecification :
read_master_switch_procedure_specification
ion
semi_colon_token
semiColonToken : ":;"
procedure_annotation
procedureAnnotation :
read_master_switch_procedure_annotation
end

read_master_switch_procedure_specification in
procedure_specification, Token with
procedure_token
procedureToken : "procedure"
defining_identifier
definingIdentifier : "Read_Master_Switch"
parameter_profile
parameterProfile :
read_master_switch_parameter_profile
end

read_master_switch_parameter_profile in
parameter_profile, Token with
formal_part
formalPart :
read_master_switch_formal_part
end

read_master_switch_formal_part in
formal_part, Token with
left_parenthesis_token
leftParenthesisToken : "("parameter_specification
parameterSpecification :
read_master_switch_parameter_specification
right_parenthesis_token
rightParenthesisToken : ")"
end

read_master_switch_parameter_specification in
parameter_specification, Token with
defining_identifier_list
definingIdentifierList :
read_master_switch_defining_identifier_list
colon_token
colonToken : ":;"
mode
parameterMode : "out"subtype_mark
subtypeMark : "Switch"
end

read_master_switch_defining_identifier_list
in defining_identifier_list, Token with
defining_identifier
definingIdentifier : "Position"
end

read_master_switch_procedure_annotation
in procedure_annotation, Token with
moded_global_definition
modedGlobalDefinition :
read_master_switch_moded_global_definition
dependency_relation
dependencyRelation :
read_master_switch_dependency_relation
end

Representation of Read_Master_Switch 'demo' global procedure annotation

read_master_switch_moded_global_definition in
moded_global_definition, Token with
Structuring Development Artifacts

global_token
  globalToken : '--# global'
global_mode
  globalMode : "in out"
entire_variable_list
  entireVariableList : read_master_switch_entire_variable_list
semi_colon_token
  semiColonToken : ';' end
read_master_switch_entire_variable_list
  in entire_variable, Token with
  entireVariable :
  master_switch_entire_variable end
master_switch_entire_variable in
  entire_variable, Token with
direct_name
  directName : "Master_Switch" end

Representation of Read_Master_Switch '---# derives' procedure annotation
read_master_switch_dependency_relation
  in dependency_relation, Token with
derives_token
  derivesToken : '---# derives'
dependency_clause
  dependencyClause : read_master_switch_dependency_clause
semi_colon_token
  semiColonToken : ';' end
read_master_switch_dependency_clause
  in dependency_clause, Token with
token
  from_token
  fromToken : "from"
imported_variable_list
  importedVariableList : read_master_switch_dependency_imported_variable_list
end
read_master_switch_dependency_imported_variable_list
  in imported_variable_list, Token with
token
  entireVariable :
  master_switch importend
end

Representation of the Read_Altitude_Switch procedure
proc2_package_declarative_item
  in package_declarative_item, Token with
subprogram_declaration
  subprogramDeclaration :
read_altitude_switch_subprogram_declaration
  end
read_altitude_switch_subprogram_declaration
  in subprogram_declaration, Token with
procedure_specification
  procedureSpecification :
read_altitude_switch_procedure_specification
  semi_colon_token
  semiColonToken : ';' procedure_annotation
  procedureAnnotation :
read_altitude_switch_procedure_annotation
  end
read_altitude_switch_procedure_annotation
  in procedure_specification, Token with
procedure_token
  procedureToken : "procedure"
defining_identifier
  definingIdentifier :
  "Read_Altitude_Switch" parameter_profile
  parameterProfile :
read_altitude_switch_parameter_profile
  end
read_altitude_switch_parameter_profile
  in parameter_profile, Token with
formal_part
  formalPart :
read_altitude_switch_formal_part
  end
read_altitude_switch_formal_part
  in formal_part, Token with
left_parenthesis_token
leftParenthesisToken : "(*
parameter_specification
parameterSpecification :
read_altitude_switch_parameter_specification
right_parenthesis_token
rightParenthesisToken : ")"
end

read_altitude_switch_parameter_specification in parameter_specification, Token with
defining_identifier_list
definingIdentifierList :
read_altitude_switch_defining_identifier_list
colon_token
colonToken : ":;
mode
parameterMode : "out"
subtype_mark
subtypeMark : "Switch"
end

read_altitude_switch_defining_identifier_list in defining_identifier_list, Token with
defining_identifier
definingIdentifier : "Position"
end

read_altitude_switch_procedure_annotaiton in procedure_annotaiton, Token with
dnunged_global_definition
dnungedGlobalDefinition :
read_altitude_switch_moded_global_definition
dependency_relation
dependencyRelation :
read_altitude_switch_dependency_relation
end

Representation of Read_Altitude_Switch
'--# global' procedure annotation
read_altitude_switch_moded_global_definition in moded_global_definition, Token with
global_token
globalToken : "--# global"
global_mode
globalMode : "in out"
entire_variable_list
tenireVariableList :
read_altitude_switch_entire_variable_list
semiColon_token
semiColonToken : ";
end

read_altitude_switch_entire_variable_list in entire_variable_list, Token with
dire_num
directName : "Altitude_Switch"
end

Representation of Read_Altitude_Switch
'--# derives' procedure annotation
read_altitude_switch_dependency_relation in dependency_relation, Token with
derives_token
derivesToken : "--# derives"
dependency_clause
dependencyClause :
read_altitude_switch_dependency_clause
semiColon_token
semiColonToken : ";
end

read_altitude_switch_dependency_clause in dependency_clause, Token with
entire_variable_list
tenireVariableList :
read_altitude_switch_dependency_entire_variable_list
from_token
fromToken : "from"
imported_variable_list
importedVariableList :
read_altitude_switch_dependency_imported_variable_list
end

read_altitude_switch_dependency_entire_variable_list in entire_variable_list, Token with
tire_variable
tenireVariable :
read_altitude_switch_dependency_position
entire_variable_list_item
tenireVariableListItem :
read_altitude_switch_dependency_altitude_switch_list_item
end

read_altitude_switch_dependency_position in entire_variable, Token with
dire_num
directName : "Position"
end

read_altitude_switch_dependency_altitude_switch_list_item in entire_variable, Token with
comma_token
commaToken :
entire_variable
entireVariable

read_altitude_switch_dependency_altitude_switch in entire_variable, Token with
dire_num
directName : "Altitude_Switch"
end

read_altitude_switch_dependency_imported_variable_list in imported_variable_list, Token with
entire_variable_list
entireVariableList :
read_altitude_switch_dependency_entire_variable_list_imported
end

read_altitude_switch_dependency_entire_variable_list_imported in entire_variable_list, Token with
tire_variable
tenireVariable :
read_altitude_switch_dependency_altitude_switch_imported
end

read_altitude_switch_dependency_altitude_switch_imported in entire_variable, Token with
Structuring Development Artifacts

direct_name

directName : "Altitude_Switch"
end

Representation of the
Read_Heading_Switch procedure

proc3_package_declarative_item in
package_declarative_item, Token with
subprogram_declaralve

subprogramDeclaration:
read_heading_switch_subprogram_declarat

ead subprogram_declaration, Token with

procedureSpecification

procedureSpecification

procedureAnnotation

procedureAnnotation

semi_colon_token

semiColonToken : ";"

procedure_annotation

read_heading_switch_procurement_annotation

end

read_heading_switch_subprogram_declaralve

in subprogram_declaralve, Token with

procedure_specification

procedureSpecification

parameter_profile

parameterProfile

read_heading_switch_parameter_profile

end

read_heading_switch_subprogram_profile

in parameter_profile, Token with

formal_part

formalPart

read_heading_switch_formal_part

end

read_heading_switch_formal_part in
formal_part, Token with

left_parenthesis_token

leftParenthesisToken : "(

parameter_specification

procedureSpecification

right_parenthesis_token

rightParenthesisToken : ")"
end

read_heading_switch_parameter_profile

in parameter_profile, Token with

formal_part

formalPart

read_heading_switch_formal_part

end

read_heading_switch_formal_part in
formal_part, Token with

left_parenthesis_token

leftParenthesisToken : "(

parameter_specification

procedureSpecification

right_parenthesis_token

rightParenthesisToken : ")"
end

read_heading_switch_dependency_clause

in dependency_clause, Token with
defining_identifier_list

definingIdentifierList : defining_identifier

colon_token

colonToken : "::"

mode

parameterMode : "out"

subtype_mark

subtypeMark : "Switch"
end

read_heading_switch_dependency_clause

end

Representation of Read_Heading_Switch
`--# global' procedure annotation

read_heading_switch_moded_global_definition

in moded_global_definition, Token with
global_token

globalToken : "--# global"
global_mode

globalMode : "in out"

entire_variable_list

entireVariableList : read_heading_switch_entire_variable_list

semi_colon_token

semiColonToken : ";"
end

read_heading_switch_entire_variable_list

in entire_variable_list, Token with
defining_identifier

definingIdentifier : "Position"
end

read_heading_switch_dependency_relation

in dependency_relation, Token with
derives_token

derivesToken : "--# derives"
derives_clause

dependencyClause : read_heading_switch_dependency_clause

semi_colon_token

semiColonToken : ";"
end

read_heading_switch_dependency_clause

in dependency_clause, Token with
defining_identifier_list

definingIdentifierList : defining_identifier

colon_token

colonToken : "::"

mode

parameterMode : "out"

subtype_mark

subtypeMark : "Switch"
end

read_heading_switch_dependency_relation

end

Representation of Read_Heading_Switch
`--# derives' procedure annotation

read_heading_switch_dependency_relation

in dependency_relation, Token with
derives_token

derivesToken : "--# derives"
derives_clause

dependencyClause : read_heading_switch_dependency_clause

semi_colon_token

semiColonToken : ";"
end

read_heading_switch_dependency_clause

in dependency_clause, Token with
defining_identifier_list

definingIdentifierList : defining_identifier

colon_token

colonToken : "::"

mode

parameterMode : "out"

subtype_mark

subtypeMark : "Switch"
end

read_heading_switch_dependency_relation

end
read_heading_switch_dependency_position
entire_variable_list_item
read_heading_switch_dependency_heading_switch_list_item
end

read_heading_switch_dependency_position
in entire_variable, Token with
direct_name
  directName : "Position"
end

read_heading_switch_dependency_heading_switch_list_item
in entire_variable_list_item, Token with
comma_token
  commaToken : ","
entire_variable
  entireVariable
read_heading_switch_dependency_heading_switch
end

read_heading_switch_dependency_heading_switch
in entire_variable, Token with
direct_name
  directName : "Heading_Switch"
end

read_heading_switch_dependency_imported_variable_list
in imported_variable_list, Token with
entire_variable_list
  entireVariableList : read_heading_switch_dependency_entire_variable_list_imported
end

read_heading_switch_dependency_entire_variable_list
imported
in entire_variable_list, Token with
entire_variable
  entireVariable : read_heading_switch_dependency_heading_switch_imported
end

read_heading_switch_dependency_heading_switch_imported
in entire_variable,
Token with
direct_name
  directName : "Heading_Switch"
end

Our second code fragment (taken from the same example) represents the autopilot package itself which features a procedure called Control. This is specified as follows (note abbreviation of the global annotation):

--# inherit Surfaces, Instruments;
package AP
--# own State;
--# initializes State;
is
  procedure Control;
  --# global in out State, Surfaces.Elevators, Surfaces.Ailerons ...
end AP;

We now consider the O-Telos representation for this code:

Representation of the AP package

ap_package_declaration in
  package_declaration, Token with
  package_specification
    packageSpecification : ap_package_specification
  end
ap_package_specification in
  package_specification, Token with
  inherit_clause
    inheritClause : ap_inherit_clause
  package_token
    packageToken : "package"
defining_program_unit_name
    definingProgramUnitName : ap_defining_program_unit_name
package_annotation
    packageAnnotation : ap_package_annotation
is_token
  isToken : "is"
package_declarative_item
  packageDeclarativeItem1 : ap_proc1_package_declarative_item
end_token
  endToken : "end"
identifier
  packageIdentifier : "AP"
semi_colon_token
  semiColonToken : ";"
end

Representation of the "--# inherit" AP package annotation

ap_inherit_clause in inherit_clause,
  Token with
  inherit_token
    inheritToken : "--# inherit"
Structuring Development Artifacts

packageName : "Surfaces"

instruments_package_name_list_item in

semi_colon_token :

semiColonToken ";"

end

instruments_package_name_list_item in

package_name_list_item, Token with

comma_token

commaToken",

package_name

packageName : "Instruments"

end

 Representation of the AP package name

ap_defining_program_unit_name in

defining_program_unit_name, Token with

defining_identifier

definingIdentifier : "AP"

end

 Representation of package annotations

ap_package_annotation in

package_annotation, Token with

own_variable_clause

ownVariableClause : ap_own_variable_clause

initialization_specification

initializationSpecification : ap_initialization_specification

end

 Representation of the '---# own' package annotation

ap_own_variable_clause in

own_variable_clause, Token with

own_variable_token

ownVariableToken : "---# own"

own_variable_list

OwnVariableList : ap_annotations_ap__own_variable_list

semi_colon_token

semicolonToken :

end

ap_annotations_ap__own_variable_list in

own_variable_list, Token with

own_variable

ownVariable : "State"

end

 Representation of the '---# initializes' package annotation

ap_initialization_specification in

initialization_specification, Token with

initialization_token

initializationToken : "---# initializes"

own_variable_list

OwnVariableList : ap__initialization_specification_ap__own_variable_list

semi_colon_token

semicolonToken :

end

ap__initialization_specification_ap__own_variable_list in

own_variable_list, Token with

own_variable

ownVariable : "State"

end

 Representation of the Control procedure

ap_procl_package_declarative_item in

package_declarative_item, Token with

subprogram_declaration

subprogramDeclaration:

control_subprogram_declaration

end

control_subprogram_declaration in

subprogram_declaration, Token with

procedure_specification

procedureSpecification : control_procedure_specification

semi_colon_token

semicolonToken :

end

control_procedure_specification in

procedure_specification, Token with

procedure_token

procedureToken : "procedure"

defining_identifier

definingIdentifier : "Control"

end

 Representation of Control '---# global' procedure annotation

control_procedure_annotation in

procedure_annotation, Token with

moded_global_definition

modeGlobalDefinition :

control_moded_global_definition

end

control_moded_global_definition in

moded_global_definition, Token with

global_token

globalToken : "---# global"

global_mode

globalMode : "in out"

type_variable_list

typeVariableList : control_type_variable_list

semi_colon_token

semicolonToken :

end

type_variable_list in
type_variable_list, Token with

type_variable

typeVariable

typeVariable in

type_variable, Token with

direct_name

directName : "State"

end

surfaces_elevators_entire_variable_list

_entire_variable_list1:
surfaces_ailerons_entire_variable_list

_entire_variable_list2:
surfaces_elevators_entire_variable_list

end

state_entire_variable in

entire_variable, Token with

direct_name

directName : "State"

end

surfaces_elevators_entire_variable_list

_entire_variable_list1 in entire_variable_list_item,

Token with

comma_token

commaToken :

",”
Towards a SPARKAda Programming Language Structure

entire_variable
entireVariable :

surfaces_elevators_entire_variable
end

surfaces_elevators_entire_variable in
entire_variable, Token with
package_name
packageName "Surfaces"
period_token
periodToken : "."
direct_name
directName : "Elevators"
end

surfaces_ailerons_entire_variable_list_item in entire_variable_list_item,

4.5.4 Application to RTN-SL Textual Specifications

To demonstrate versatility of the modelling philosophy introduced in subsection 4.5.3.1.1 (and used to develop the SPARK structure in 4.5.3.2), we now employ it to follow option two for constructing meta-models in MATrA (subsection 4.4.3) for a subset of RTN-SL (Paynter, 2000). A comparison of models yielded by both approaches appears in 4.5.4.4.

4.5.4.1 Towards An RTN-SL (Textual) Structure

RTN-SL is a textual notation for the specification of flat Real-Time Networks; the graphical RTN-SLg notation described in section 4.4 provides a subset of features contained in this language and is therefore mainly an aid to comprehension. We note that RTN-SL is the subject of ongoing research within MBDA (UK) and hence the syntax is subject to revision (although RTN-SLg is altogether more stable). We also reiterate that our purpose here is purely to provide a 'flavour' of how another textual language with a formal foundation may be represented using the approach set out above and not to discuss features of RTN-SL in depth. Interested readers are instead referred to Paynter (2000).

The RTN-SL subset of interest in this thesis is the set of constructs that enable specification of ports; recall from section 4.4 that ports provide the interface between activities (single-threaded processes) within a Real-Time Network. The syntax is as follows:-

```
ports
  port_id_list : (protocol, data_type, direction);
end ports
```

For each list of port names or identifiers, a three tuple describing the interface to these ports is defined. Specifically, the protocol on paths connected to these ports, a type of data communicable through that protocol and the direction of data-flow (in or out) along its path.

4.5.4.2 Meta-model Definitions

This section introduces schemas towards a partial RTN-SL meta-model. Again the BNF syntax, UML meta-model and O-Telos implementation of base classes are all considered, together with any OCL
Structuring Development Artifacts

structural constraints arising from modelling decisions. We begin by describing the activity element in order to place our schemas for ports in context.

- **activity**
  An activity (figure 4.54) is described in terms of its name (followed by the reserved word is) and a list of imported ADTs (with_list), together with declarations of the ports, auxiliary definitions (auxiliary_definitions), local state (local_state), operations and state machine (state_machine) elements; with_list, auxiliary_definitions and local_state are optional. All elements are framed between activity and end activity reserved words; the construct terminates with a semi-colon.

- **ports**
  The ports statement (figure 4.55) contains a port definitions (port_defs) element, framed between ports and end ports reserved words; again this construct terminates with a semi-colon.

- **port_defs**
  Port definitions (port_defs) is defined in terms of one or more port definition (port_def) statements. This is shown in figure 4.56.

- **port_def**
  Each port_def (figure 4.57) is made up of a port id-list and a description of the port type (port_type) applicable to these ports (the two are separated by a colon character); this construct also terminates with a semi-colon.

- **id_list**
  As its name suggests, the id_list (figure 4.58) contains one or more port identifiers (names) as a comma-delimited list.

- **port_type**
  This element defines the interface to ports named in the id_list which, as previously described, contains their communication protocol (signal, pool, channel, dataless-channel or stimulus), data type (a_type_ref) and a reserved word - in or out - indicating direction of data flow along the path connected; port_type is enclosed in parentheses and shown in figure 4.59.

- **a_type_ref**
  From figure 4.60 it can be seen that a data type (a_type_ref) is defined either as a built-in or basic type (basic_type) such as bool, integer, character, etc., or else using the name of an Abstract Data Type (ADT). As indicated in section 4.4, ADTs may be used in a hierarchical manner, in which case it is necessary to prefix imported ADTs with the name of the ADT in which they are defined; the two are separated by a period token. This is modelled by introducing rolenames for ADT and parent (adt_name and parent_name respectively) to differentiate references to the name class (cf. the BNF representation).
Towards a SPARK Ada Programming Language Structure

BNF Syntax

activity ::= activity name is [with_list] ports [auxiliary_definitions] [local_state] operations state_machine end activity;

O-Telos

activity in StructureElement, SimpleClass isA AerospaceEngineeringObject with has_property
  begin_activity : activity_token;
  is_token : is_token;
  end_token : end_token;
  end_activity : activity_token;
  semi_colon_token : semi_colon_token
has_part
  name : name;
  with_list : with_list;
  ports : ports;
  auxiliary_definitions : auxiliary_definitions;
  local_state : local_state;
  operations : operations;
  state_machine : state_machine
end

name in StructureElement, SimpleClass isA String, AerospaceEngineeringObject end

activity_token in ArtifactProperty, SimpleClass isA String with constraint activity_Is_Activity:
  $forall a/activity_token a = "activity"$
end

Figure 4.54 - 'activity schema'
Structuring Development Artifacts

### BNF Syntax

| ports ::= ports port_defs end ports ; |

| ports : \=
| begin_ports : ports_token ;
| end_token : end_token ;
| end_ports : ports_token ;
| semi_colon_token : semi_colon_token ;

| port_defs : port_def |

| port_defs ::= port_def (port_def) |

### O-Telos

ports in StructureElement, SimpleClass isA AerospaceEngineeringObject with

| has_property |
| begin_ports : ports_token ;
| end_token : end_token ;
| end_ports : ports_token ;
| semi_colon_token : semi_colon_token ;

| has_part |
| port_defs : port_defs |

| end |

ports_token in ArtifactProperty,
SimpleClass isA String with constraint

| ports_Is_Ports: |
| $ \text{forall p/ports_token p = 'ports'} \$ |

| end |

---

Figure 4.55 - 'ports schema'

---

Figure 4.56 - 'port_defs schema'
Towards a SPARK Ada Programming Language Structure

**port_def**

<table>
<thead>
<tr>
<th>BNF Syntax</th>
<th>O-Telos</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_def ::= id_list : port_type ;</td>
<td>port_def in StructureElement, SimpleClass with has_property</td>
</tr>
<tr>
<td></td>
<td>colon_token : colon_token;</td>
</tr>
<tr>
<td></td>
<td>semi_colon_token : semi_colon_token</td>
</tr>
<tr>
<td></td>
<td>has_part id_list : id_list;</td>
</tr>
<tr>
<td></td>
<td>port_type : port_type</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

Figure 4.57 - 'port_def schema'

**id_list**

<table>
<thead>
<tr>
<th>BNF Syntax</th>
<th>O-Telos</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_list ::= name {, name}</td>
<td>id_list in StructureElement, SimpleClass with has_part name : name;</td>
</tr>
<tr>
<td></td>
<td>id_list_item : id_list_item end</td>
</tr>
<tr>
<td></td>
<td>id_list_item in StructureElement, SimpleClass with has_property</td>
</tr>
<tr>
<td></td>
<td>comma_token : comma_token</td>
</tr>
<tr>
<td></td>
<td>has_part name : name</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

Figure 4.58 - 'id_list schema'
Structuring Development Artifacts

Figure 4.59 - 'port_type schema'

**BNF Syntax**

<table>
<thead>
<tr>
<th>BNF Syntax</th>
<th>O-Telos</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_type ::= (protocol, a_type_ref, in)</td>
<td>port_type in StructureElement, SimpleClass</td>
</tr>
<tr>
<td></td>
<td>isA AerospaceEngineeringObject</td>
</tr>
<tr>
<td></td>
<td>with</td>
</tr>
<tr>
<td></td>
<td>has_property</td>
</tr>
<tr>
<td></td>
<td>left_parenthesis_token :</td>
</tr>
<tr>
<td></td>
<td>left_parenthesis_token;</td>
</tr>
<tr>
<td></td>
<td>protocol_comma : comma_token;</td>
</tr>
<tr>
<td></td>
<td>a_type_ref_comma : comma_token;</td>
</tr>
<tr>
<td></td>
<td>in_token : in_token;</td>
</tr>
<tr>
<td></td>
<td>out_token : out_token;</td>
</tr>
<tr>
<td></td>
<td>right_parenthesis_token :</td>
</tr>
<tr>
<td></td>
<td>right_parenthesis_token</td>
</tr>
<tr>
<td></td>
<td>has_part</td>
</tr>
<tr>
<td></td>
<td>protocol : protocol;</td>
</tr>
<tr>
<td></td>
<td>a_type_ref : a_type_ref</td>
</tr>
<tr>
<td>protocol ::= pool</td>
<td>protocol in StructureElement, SimpleClass</td>
</tr>
<tr>
<td></td>
<td>channel</td>
</tr>
<tr>
<td></td>
<td>isA String</td>
</tr>
<tr>
<td></td>
<td>with constraint in_Is_In:</td>
</tr>
<tr>
<td></td>
<td>$ forall i/in_token i = &quot;in&quot; $</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td></td>
<td>out_token in ArtifactProperty, SimpleClass</td>
</tr>
<tr>
<td></td>
<td>isA String</td>
</tr>
<tr>
<td></td>
<td>with constraint out_Is_Out:</td>
</tr>
<tr>
<td></td>
<td>$ forall o/out_token o = &quot;out&quot; $</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td></td>
<td>protocol in StructureElement, SimpleClass</td>
</tr>
<tr>
<td></td>
<td>isA String, AerospaceEngineeringObject</td>
</tr>
<tr>
<td></td>
<td>with constraint</td>
</tr>
<tr>
<td></td>
<td>enum_Basic_Type:</td>
</tr>
<tr>
<td></td>
<td>$forall p/protocol</td>
</tr>
<tr>
<td></td>
<td>(p = &quot;pool&quot;) or</td>
</tr>
<tr>
<td></td>
<td>(p = &quot;channel&quot;) or</td>
</tr>
<tr>
<td></td>
<td>(p = &quot;signal&quot;) or</td>
</tr>
<tr>
<td></td>
<td>(p = &quot;stimulus&quot;) or</td>
</tr>
<tr>
<td></td>
<td>(p = &quot;dataless_channel&quot;) $</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

Figure 4.59 - 'port_type schema'
Towards a SPARK Ada Programming Language Structure

**BNF Syntax**

```
a_type_ref ::= basic_type | name
   . name | name

basic_type ::= bool | nat | nat1 |
           integer | real | char | null
```

**OCL**

```
a_type_ref invariant
   self.allInstances->forAll(a | 
   (a.adt_name->size = 1 and 
    a.parent_name->isEmpty and 
    a.period_token->isEmpty and 
    a.basic_type->isEmpty)
   xor
   (a.parent_name->size = 1 and 
    a.period_token->size = 1 and 
    a.adt_name->size = 1 and 
    a.basic_type->isEmpty)
   xor
   (a.basic_type->size = 1 and 
    a.parent_name->isEmpty and 
    a.period_token->isEmpty and 
    a.adt_name->isEmpty)
```

**O-Telos (base classes)**

```
a_type_ref in 
   StructureElement, 
   SimpleClass isA 
   AerospaceEngineeringObject 
   with 
   has_property 
   period_token : period_token 
   has_part 
   basic_type : 
   parent_name : name; 
   adt_name : name 
end 

basic_type in 
   StructureElement, 
   SimpleClass isA String, 
   AerospaceEngineeringObject 
   with constraint 
   enum_Basic_Type: $forall b/basic_type 
   (b = "bool") or 
   (b = "nat") or 
   (b = "nat1") or 
   (b = "integer") or 
   (b = "real") or 
   (b = "char") or 
   (b = "null") $ 
end
```

**Constraint preserving the BNF alternatives syntax**

```
$forall b/basic_type 
   (b = "bool") or 
   (b = "nat") or 
   (b = "nat1") or 
   (b = "integer") or 
   (b = "real") or 
   (b = "char") or 
   (b = "null")$
```

Figure 4.60 - ‘a_type_ref schema’
4.5.4.3 Specifying Activities and Ports using the RTN-SL Structure: A Worked Example

We now demonstrate how an O-Telos implementation of the structure elements introduced in subsection 4.5.4.2 may be used to represent ports for a simple Real-Time Network activity specified using the RTN-SL textual notation. The activity is stated as follows:-

```
activity a1 is
  ports
    p1 : (signal, image.raw_image, in);
    p2 : (channel, image.processed_image, out);
    p3, p4 : (pool, integer, in);
    p5 : (channel, bool, out);
  end ports
end activity;
```

Activity a1 has five ports. Each has a name (identifier) - p1, p2, p3, p4 and p5 - and an interface defining the protocol, data type and direction as previously described. Observe that data types communicated are user-defined in the case of p1 and p2, and built-in for p3, p4 and p5. Also note that ports p3 and p4 share the same interface characteristics. We now present the O-Telos representation of this RTN-SL code fragment:-

```
Sample population of port constructs for activity a1

Definition of a1
a1_activity in activity, Token with begin_activity
  beginActivity : "activity"
  name
    _name : "a1"
  is_token
    isToken : "is"
  ports
    ports : a1_ports
    auxilliary_definitions : -not shown
    local_state
      localState : -not shown
    operations
      _operations : -not shown
      state_machine
        stateMachine : -not shown
    end_token
end_activity
semi_colon_token
semiColonToken :
end

Definition of ports for a1
a1_ports in ports, Token with begin_ports
  beginPorts : "ports"
  port_defs : a1_port_defs
  end_token
end_ports
semi_colon_token
semiColonToken :
end
```

205
Towards a SPARK Ada Programming Language Structure

id_list
  idList : p5_port_def_id_list
  colon_token
  colonToken : ";"
end

port_type
  portType : p5_port_type
  semi_colon_token
  semiColonToken : ";"
end

p1_port_def_id_list in id_list, Token
  name
  _name : "p1"
end

p2_port_def_id_list in id_list, Token
  name
  _name : "p2"
end

p3_and_p4_port_def_id_list in id_list, Token
  name
  _name : "p3"
end

p4_list_item in id_list_item, Token
  comma_token
  commaToken : ","
  name
  _name : "p4"
end

p5_port_def_id_list in id_list, Token
  name
  _name : "p5"
end

p1_port_type in port_type, Token with
  left_parenthesis_token
  leftParenthesisToken : ";"
  protocol
  _protocol : "signal"
  protocol_comma
  protocolComma : ","
  a_type_ref
  aTypeRef : p1_a_type_ref
  a_type_ref_comma
  aTypeRefComma : "",
  in_token
  inToken : "in"
  right_parenthesis_token
  rightParenthesisToken : ");"
end

p2_port_type in port_type, Token with
  left_parenthesis_token
  leftParenthesisToken : ";"
  protocol
  _protocol : "channel"
  protocol_comma
  protocolComma : ","
  a_type_ref
  aTypeRef : p2_a_type_ref
  a_type_ref_comma
  aTypeRefComma : "",
  out_token
  outToken : "out"
  right_parenthesis_token
  rightParenthesisToken : ");"
end

p3_and_p4_a_type_ref in a_type_ref, Token
  basic_type
  basicType : "integer"
end

p5_a_type_ref in a_type_ref, Token
  basic_type
  basicType : "bool"
end

4.5.4.4 Relationship Between RTN Meta-Models

Having now proposed RTN-SL meta-models developed using both modelling options enumerated in subsection 4.4.3, it is worth briefly reflecting on the results yielded by these differing approaches.
Figure 4.61 juxtaposes the featured textual RTN-SL model fragment (from figures 4.54 and 4.55, subsection 4.4.3.2), with the corresponding RTN-SLg fragment from figure 4.22 (subsection 4.4.3.2.1); table 4.3 demonstrates the mapping between elements of these two representations: i.e., activity to Activity, with_list to Adt, ports to Port and state_machine to StateMachine (note local_state, auxiliary_definitions and operations are not represented in RTN-SLg and have therefore been removed).

Table 4.3 - ‘Mapping Table : RTN-SL (Textual) to RTN-SLg Graphical Meta-model Elements’

The two UML fragments can be considered isomorphic under the mapping of table 4.3. We make the observation that in RTN-SLg the name attribute of Activity (inherited from the abstract Component class - not shown) is represented as a class in the RTN-SL model, but that (as indicated in subsection 2.2.2.1.1) relationships between a class (or entity) and its attributes essentially present the same Conceptual Modelling abstraction as aggregation (Peckham & Maryanski, 1988); alternatively, in the RTN-SLg model, name could be promoted to a class. With this minor cosmetic change, the two models are isomorphic.
4.5.5 Relationship to the Traceability Dimensions

Ada and SPARK are the languages most commonly used with Real-Time Network notations such as MASCOT, DORIS and (it is anticipated) RTN-SL. Indeed, the relationship between MASCOT 3 and Ada constructs was investigated in Jackson (1986). From a traceability stand-point, engineers may wish to establish intra-micro vertical trace relations (again using the approach outlined in 3.3.6.3.2) between network Activities or IDAs and their corresponding Ada package specifications.

Note however that this is but one possible source and target pairing for one possible (and unspecified) association. Having provided means to establish links at a user determined granularity (including potentially very fine-grained associations for critical code fragments), we have, as previously indicated, earmarked as a future work item the need for a through investigation to establish a comprehensive set of systems engineering trace relation types (see subsection 7.4.3).

4.5.6 Summary

An alternative to developing new languages specifically for use in critical software is to remove features from an existing language that prevent verification, and to add mechanisms supporting analysis and proof. One such language is SPARK, a safe subset of Ada which is used extensively through the aerospace industry. For that reason, it was chosen as our representative (software) implementation notation for the MATrA traceability framework.

Accordingly we proposed a novel meta-model capturing a small number of SPARK constructs based on the BNF string grammar. Its granularity of expression supports verification of primitives against the PDS, as well as allowing flexibility in the setting up of trace-relations, both among SPARK elements and also between SPARK elements and those from other structures; for critical code fragments, these relations can if necessary be extremely fine-grained (e.g., between individual data elements in the specification and implementation). This, together with a desire to provide full compatibility with the source language were the main motivation in our decision to ‘pitch’ the model at such a detailed level.

A ‘modelling philosophy’ was also introduced detailing a series of mechanical steps that may be used to extend the SPARK model to include the remaining language features, or as subsequently demonstrated, to produce models for other languages expressed as a string grammar - in this case RTN-SL. To provide proof of concept, partial O-Telos implementations of both models were populated using sample code fragments.

Again, the SPARK Ada structure is fully evaluated in Chapter Seven.
4.6 Chapter Summary

This chapter has presented a number of novel meta-models (traceability structures) for well-defined and flexible development notations used by the aerospace industry. These included a Natural Language structure, a User Centred Requirements Structure (featuring Use Case Models, Scenarios and Message Sequence Charts), a structure for the representation of Real-Time Networks and a structure for the representation of SPARK Ada program code.

Each notation is potentially supported by disparate tools, with all the attendant difficulties for traceability discussed in Chapters One and Three. Our aim was therefore to provide a common representation to enable traceability across notations and hence data originating from these tools. Thus, for each notation we expressed key syntactic elements as a UML Class Diagram, with well-formedness, consistency and other constraints (for all except the Ada structure) stated in OCL. Base classes were then implemented in O-Telos (using ConceptBase) to show one possible means of automation, with worked examples also included for the Real-Time Network and Ada structures.

As indicated in Chapter Three, each structure has a common basis assured by the System Engineering Notation Meta-model, and is intended to be populated via the tool2matrix transfer mechanism and verified for overall consistency against the Product Data Synthesis. Collectively the featured structures provide representative coverage of one Workspace viewpoint, i.e. development; the assessment and product management perspectives are considered in Chapter Five.
Chapter 5  Structuring Safety Assessment and Product Management Artifacts

5.1 Introduction
Chapter Five introduces further meta-models (traceability structures) which again provide input to the MATrA Workspace, this time representing well-defined and flexible notations for safety assessment and product management. The product management theme is further investigated through a structure to support traceability of revisions and variants. As in the previous chapter, we state factors motivating the inclusion of a structure, summarise the main concepts and relate it to the dimensions (from Chapter One). We also present an appropriate UML Class Diagram, OCL constraints and partial O-Telos implementation. Again, worked examples are provided to demonstrate key aspects for meta-models not featured in our main case studies in Chapter Six.

5.2 Fault Tree Analysis Structure

5.2.1 Introduction
In this section we propose a structure capturing the graphical syntax (again with a ‘light-weight’ formal semantics amenable to traceability) for Fault Tree Analysis, a technique used extensively in the aerospace industry for identifying and relating all events which alone or in combination may lead to an undesirable (safety significant) failure condition.

5.2.2 Motivation
Safety engineers use a range of analytical techniques, each one providing different coverage of the target system. One such technique is Fault Tree Analysis (FTA) which is widely used in the defence, aerospace and electronics industries. It is also recommended by several product assurance standards, including those by the International Electrotechnical Commission (IEC 61508, 1998), UK MoD (MoD, 1996) and Society of Automotive Engineers EUROCAE (1996b); indeed, we use case study material from the latter in Chapter Six as a platform for demonstrating work in this thesis. FTA was originally conceived in the 1960s as a means of analysing hardware failures, having been jointly developed by Bell Laboratories and the United States Airforce to help investigate inadvertent launch conditions for the Minuteman missile system. However, Leveson & Harvey have since extended the principles of FTA to software systems (1983).

Two complementary strategies underpin safety analysis techniques; these have been termed deductive and inductive approaches (Vesely et al., 1981). Deductive techniques such as Fault Tree Analysis start from a system failure and then reason about system or component states contributing to that failure. Conversely, inductive techniques such as Failure Modes and Effects Analysis (see subsection 5.3) consider a particular fault in a system component and then attempt to ascertain its consequences. We therefore consider it appropriate to incorporate both approaches within the MATrA framework; thus Fault Tree Analysis provides our deductive example.
Results of the various analytical techniques are typically represented either graphically (as per Fault Tree Analysis), or else in tabular form (as per Failure Modes and Effects and Analysis). Again, it is appropriate to include both formats within the MATrA framework; thus Fault Tree Analysis also provides our graphical example.

5.2.2.1 Fault Tree Analysis Overview

As indicated above, Fault Tree Analysis is a deductive technique in that it starts from one particular undesirable event - termed the top event - and provides an approach to investigating potential causes. The choice of undesirable event is particularly important; too general and the fault tree becomes unmanageable; too specific and the analysis may fail to provide a sufficiently broad view. Analysts then examine the system (or a model of the system) to determine ways in which a top event may occur. In doing so a range of factors are considered, from component and human failures, to random events occurring in the system environment.

The fault tree itself provides a graphical representation of event combinations that can lead to the undesirable event. A number of different event types commonly occur in fault trees, including intermediate, basic, undeveloped, external and conditional. These are explained in table 5.1. The different event types are represented by rectangle, circle, diamond, house and ellipse symbols respectively.

<table>
<thead>
<tr>
<th>EVENT TYPES</th>
<th>Description</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intermediate</td>
<td>A fault arising from one or more preceding causes acting through a logic gate</td>
<td></td>
</tr>
<tr>
<td>Basic</td>
<td>An event that is internal to the system under analysis and which requires no further decomposition</td>
<td></td>
</tr>
<tr>
<td>Undeveloped</td>
<td>An event that is not developed further, either because it has little impact on the top level event, or because information for its development is not readily available</td>
<td></td>
</tr>
<tr>
<td>External</td>
<td>An event that is normally expected to occur</td>
<td></td>
</tr>
<tr>
<td>Conditional</td>
<td>Restriction expressed over a logical connective (normally a PRIORITY-AND or INHIBIT gate - see table 5.2)</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1 - ‘Fault Tree Event Types’

Events are connected together by logical operators known as gates that either enable or prevent the flow of a fault up a tree. Common types of connective include the AND-gate, OR-gate, PRIORITY-AND-gate, EXCLUSIVE-OR-gate and INHIBIT-gate; these are described in table 5.2, which

1Readers are referred to (EUROCAE1996b) for a discussion on factors influencing the choice of top event and other procedural issues pertaining to Fault Tree Analysis.
Fault Tree Analysis Structure

also includes corresponding graphical symbols.

Note: given their diverse application, it is perhaps surprising that the symbols for events and gates used in constructing fault trees have remained largely 'standard'. Practitioners generally regard the Nuclear Regulatory Commission Fault Tree Handbook (Vesely et al., ibid.) - from which the descriptions here are taken - as providing the definitive definition.

<table>
<thead>
<tr>
<th>GATE TYPES</th>
<th>Description</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>• AND</td>
<td>Output fault occurs when all input faults occur</td>
<td>![AND symbol]</td>
</tr>
<tr>
<td>• OR</td>
<td>Output fault occurs when one or more input faults occur</td>
<td>![OR symbol]</td>
</tr>
<tr>
<td>• PRIORITY-AND</td>
<td>Output fault occurs when all input faults occur in a particular sequence (which is usually represented by a Conditional Event attached to the gate)</td>
<td>![PRIORITY-AND symbol]</td>
</tr>
<tr>
<td>• EXCLUSIVE-OR</td>
<td>Output fault occurs when exactly one of the input faults occur</td>
<td>![EXCLUSIVE-OR symbol]</td>
</tr>
<tr>
<td>• INHIBIT</td>
<td>Output fault occurs if the single input fault occurs in the presence of an enabling condition (represented by a Conditional Event attached to the gate)</td>
<td>![INHIBIT symbol]</td>
</tr>
</tbody>
</table>

Table 5.2 - 'Fault Tree Gate Types'

Once the fault tree logic is complete, it can be used to compute event probabilities. In order to do so, the tree must first be reduced to what is termed minimal-cut-set-form. That is, the smallest set of events capable of causing the top event.

Basic events are then annotated with their probability of occurrence. Calculation of intermediate probabilities then progresses up through the tree until probability of the top event can be calculated. Probabilities for output events of the two most common event connectives - and gates and or gates - are determined by the product and sum of their respective input events. Where applicable, it is also common to state failure rate (typically per flight hour for civil aircraft) and exposure time of basic events; event probabilities are then calculated from the product of these two values.

Figure 5.1 presents an example Fault Tree Analysis demonstrating the above concepts for the 'Loss of All Wheel Braking' hazard for an aircraft braking system.

Loss of All Wheel Braking occurs due to simultaneous loss of the three braking sub-systems; i.e., Loss of Normal Braking (NBS) AND Loss of Alternative Braking (ABS) AND Loss of Emergency Braking (EBS). Note that
probability of failure of the Emergency Brake System is set to 1.0 indicating a design intent to discount its contribution in meeting the requirement for the top event.

Downward development of Loss of Normal Braking shows that this event can occur due to Loss of Hydraulic Supply for the Green Hydraulic System OR Loss of All Hydraulic Components for the Normal Brake System OR Loss of Command Braking Ability for the BSCU (Braking System Control Unit).

The BSCU Loss of Command Braking Ability event is further decomposed into the causes BSCU Loss of Braking Commands OR BSCU1 Loss of Electrical Power WHEN BSCU2 Loss of Electrical Power. The latter differs from all other events in the tree which are expressed over a single (build) element and a condition from that element’s state space. In contrast, LOSSELECPWR captures coincidental failure of two elements (BSCU1 and BSCU2), suggesting a distinction can be made between different types of event. We return to this issue in subsection 5.2.3.1.

![Example Fault Tree for Aircraft Wheel Braking System](source: APR 4761)

Figure 5.1 - ‘Example Fault Tree for Aircraft Wheel Braking System - investigating causes of Loss of All Wheel Braking (source APR 4761)’
Finally, BSCU Loss of Braking Commands is defined by the basic events BSCU1 Fails to Operate AND BSCU2 Fails to Operate. As indicated previously, probabilities for both are derived from their respective values for rate and exposure.

5.2.3 Tracing Safety Properties in MATrA: An FTA Model

This subsection introduces a meta-model allowing integration of Fault Tree Analysis into the MATrA traceability framework. As with UCRS, bespoke tool support is assumed.

5.2.3.1 Concepts

Fault trees in MATrA support most of the standard concepts set out in subsection 5.2.2.1, including a subset of event types (Intermediate, Basic, External and Undeveloped) and gate types (And and Or), as well as means to create the minimal cut set expression. Our research indicates that these are sufficient for the majority of analyses.

In addition, MATrA fault trees reflect usage within the aerospace domain, notably as part of the ARP 4754/4761 assessment process outlined in subsection 1.4.6.2.1. Within this process, fault trees are used to determine budget probabilities - i.e. safety objectives (established prior to design) for the target aircraft, its systems and their items - as part of the Functional Hazard and Preliminary System Safety Assessments. Updated versions of these trees are subsequently produced by the System Safety Assessment to affirm whether the original safety objectives have been met; both the preliminary and updated trees form submissions to the appropriate regulatory body as evidence towards certification.

Accordingly, a MATrA Fault Tree Analysis is composed of an optional preliminary tree and a mandatory updated tree. As may be expected, there is a high degree of event commonality between the two. Therefore, to minimise redundancy (and increase traceability), MATrA fault trees support the sharing of events (or more specifically, elements of their descriptions) through the notion of 'event profiles'. Hence, an event has two profiles, the preliminary event profile and updated event profile. Preliminary profiles comprise an event type and label, together with budget probability, failure rate and exposure properties; failure rate and exposure time are applicable to basic event types only. An updated profile also has a type which may differ from that for the preliminary profile - as demonstrated by the case study in subsection 6.3 - although this is atypical. In addition, the budget probability is retained by the updated profile, while the actual probability is added. New exposure and failure rates (where applicable) may also be expressed, or alternatively, the originals reused. The same is true of event labels which, like event types are normally the same but (as the case study also demonstrates) exhibit the potential for change.

Following our observation on event types in 5.2.2.1, the labels themselves differentiate between 'Simple', 'Composed' and 'Synchronisation' categories as derived from a taxonomy proposed by Górski & Wardziński (1995). Simple events are described in terms of a subject entity and a condition;

---

2 Note the MATrA Fault Tree Structure can be easily extended to accommodate the complete range of event and gate types featured in 5.2.2.1.
Structuring Safety Assessment and Product Management Artifacts

e.g., Valve • Stuck Open. Composed events meanwhile refer to the coincidence of two or more simple events; this is often expressed as two events joined using the “when” conjunction. For example, Valve • Stuck Closed when Pump • Stuck On. Finally, Synchronisation describes the temporal relationship between two simple events E1, E2 (normally separated by the preposition “upon”) such that in boolean logic, E1 already holds when E2 becomes True; e.g. Tank • Full upon Engine • StartUp.

Event profiles are connected by gates. In contrast to the events themselves, these are not shared between the preliminary and updated trees as described above, but instead belong exclusively to either one or the other. This is to allow for changes in structure of the tree, for example the addition or removal of events. And because there are no further properties to be shared between the respective trees, nothing would be gained from using common gates.

Finally, a number of constraints are imposed over MATrA fault trees. These range from well formedness rules, to restrictions arising from modelling decisions. We further specify 'safety-criteria' identifying common cause and single failures. Checks against the Product Data Synthesis are also stated, though this time informally.

5.2.3.2 FTA Meta-model Definitions
We now introduce the UML class diagram for our Fault Tree Analysis structure (5.2.3.2.1), together with OCL constraints over elements of the model (5.2.3.2.2) and O-Telos implementation of its base classes (5.2.3.2.3).

5.2.3.2.1 FTA Meta-model
UML specification of the Fault Tree Analysis meta-model is shown in figures 5.2 (elements) and 5.3 (associations). Its core class (FaultTreeAnalysis) includes subject module (subject_module) and description (fta_description) attributes and is an instantiation of the AssessmentStructure meta-class. In turn, FaultTreeAnalysis is further defined as an aggregation of preliminary fault tree (PreliminaryFaultTree) and updated fault tree (UpdatedFaultTree) - also instantiations of AssessmentStructure - as well as the event (Event) class (with identifier attribute) which instantiates StructureElement.

It is worth noting from figure 5.2, the multiplicity of three-to-many that exists between FaultTreeAnalysis and Event. This arises from our view on what constitutes a minimal fault tree, namely two event inputs to an And-gate or Or-gate, yielding a single event output. We also note (from its zero-or-one multiplicity) that PreliminaryFaultTree is optional, reflecting our intent to increase utility of the structure such that it may be used without developing the preliminary tree (and hence is not hard-wired for a particular assessment process).

---

3 cf. LOSSELECPWR event in figure 5.1.
4 Görski & Wardziński's classification is not universally accepted, but for the purpose of this representation is assumed to be both coherent and well defined. Alternatively, we could have undertaken a study of the usage of time in fault trees within the aerospace industry and defined our own set of semantics and consistency rules; this was deemed beyond the scope of our work.
Figure 5.2 - 'Fault Tree Analysis Structure : Elements'
Figure 5.3 - 'Fault Tree Analysis Structure : Associations'
Figure 5.2 shows both PreliminaryFaultTree and UpdatedFaultTree to be specialisations of the abstract FaultTree class, an aggregation of StructureElement meta-classes representing (or capturing the principles behind) concepts from 5.2.3.1 that are common to both tree types. These include And-gate (AndGate) and Or-gate (OrGate) connectives, the minimal cut set expression (MinimalCutSetExpression) and event properties such as exposure (Exposure), rate (Rate) and budget probability (BudgetProbability). The latter are initiated as elements of preliminary trees, but are included in the updated tree (along with the ActualProbability - introduced below) for reasons of clarity; note that multiplicity of BudgetProbability is set as zero-to-many to accommodate updated fault trees for which no preliminary tree exists. Note too the inclusion of a MATrA Natural Language Structure (MatraNLS) enabling annotation of probabilities, rates and exposures. We also employ the notion of an event set (EventSet) which represents the collective input to And-gates and which simplifies expression of rules and constraints requiring examination of paths through a tree.

In addition, FaultTree includes means to represent the various different types of event label (also discussed in subsection 5.2.3.1) - namely Simple (SimpleLabel), Composed (ComposedLabel) and Synchronisation (SynchronisationLabel). Content of these labels is specified using the simple event description (SimpleEventDescription) class, with properties entity and condition.

Besides inheriting the common elements outlined above, PreliminaryFaultTree and UpdatedFaultTree also feature type specific preliminary event profile (PreliminaryEventProfile) and updated event profile (UpdatedEventProfile) elements respectively. UpdatedFaultTree further includes means to state the actual probability (ActualProbability).

Figure 5.3 indicates that a fault tree Event is made up of a single UpdatedEventProfile and an optional PreliminaryEventProfile. It also shows both forms of profile to be specialisations of the (abstract) EventProfile supertype, with attribute type - a specialisation of String (not shown) - restricted to int(ermidiate), top, bas(ic), ext(ernal) or und(eveloped).

It can be seen from figure 5.3 that PreliminaryEventProfile comprises mandatory BudgetProbability and EventLabel elements and optional Rate and Exposure elements (which are further restricted to use with basic events by a constraint in subsection 5.2.3.2.1). EventLabel is an abstract supertype with the SimpleLabel, ComposedLabel and SynchronisationLabel specialisations introduced previously. Each EventLabel subtype is described either by a single SimpleEventDescription instance for Simple events, two or more instances for Composed events, or two instances (differentiated using the initial event description (initial_event_description) and synchronisation event description (sync_event_description) rolenames) for Synchronisation events. A SimpleEventDescription can be given scope by nominating an optional qualifying entity (QualifyingEntity). This specialisation of the String class is strictly an attribute

---

5 Clearly it is beneficial to be able to view both budget and updated probabilities for events from within the updated tree in order to determine 'at-a-glance' whether original safety requirements have been met and whether further analysis is required.

6 A constraint can be defined to ensure instances of PreliminaryFaultTree include at least three budget probabilities.

7 The 'when' conjunction of composed events and the 'upon' preposition of synchronisation events are deemed to be implicit in their respective class types.
Structuring Safety Assessment and Product Management Artifacts

(indicated by it being an instance of ArtifactProperty), which is promoted to a class using composition in order to express optionality.

Like PreliminaryEventProfile, UpdatedEventProfile also includes a mandatory EventLabel, as well as optional Rate and Exposure elements. As per 5.2.3.1, EventLabel and Exposure objects created for a preliminary tree can normally be reused by the updated version, although the capability exists for new instances to be introduced if necessary; this facility is also available to Rate which is expected to change. UpdatedEventProfile is completed by the (mandatory) ActualProbability class and its corresponding BudgetProbability created for the preliminary tree; i.e., (as previously indicated), while Rate and Exposure are either retained or replaced, updated trees exhibit both forms of probability (budget and actual).

EventProfile (and hence PreliminaryEventProfile and UpdatedEventProfile) connects to the abstract gate (Gate) class, with subtypes OrGate and AndGate. In turn, OrGate takes as input two or more EventProfile elements, while AndGate takes a single EventSet (itself an aggregation of two or more event profiles). Both EventProfile and EventSet are further defined as subtypes of the abstract EventEntity class. This exists partly to allow propagation of the cause and trans_cause associations from EventProfile to both EventProfile itself and to EventSet. The cause and trans_cause associations are used in identifying paths through a fault tree and hence in the expression of constraints and deductive rules that are dependent on this ability (see subsection 5.2.3.2.2).

One such rule populates the minimal cut set expression (MinimalCutSetExpression). This class is defined as an aggregation of one or more EventEntity - i.e., EventProfile or EventSet elements. Presence of the former would indicate that a single failure can lead to the fault tree top event, something normally regarded as an undesirable system property. However, it was decided that to prevent such occurrences (by for example, making MinimalCutSetExpression an aggregation of EventSet elements) would be overly restrictive. We therefore express a ‘safety-criterion’ indicating the presence of, rather than preventing specification of fault trees that include single failures (see subsection 5.2.3.2.2-ii).

5.2.3.2.2 OCL Constraints
In this subsection, we express the following types of constraint over the Fault Tree Analysis metamodel:

- Well-formedness of fault tree elements;
- Fault tree ‘safety-criteria’;
- Constraints reflecting usage;
- Other structural and consistency restrictions attributable to modelling decisions;
- Verification of fault tree elements against the Product Data Synthesis (informal).

Note that a subset of these constraints (expressed in set theory and predicate logic rather than OCL) appeared in Mason & Saeed (1998).
We begin by establishing a rule capturing causality which populates the aggregation association cause for the EventProfile class (see figure 5.3); cause can then be used to populate the trans_cause association. The latter is not shown, but like cause can be expressed by restating the corresponding rule from Mason & Saeed (ibid.) in OCL.

- Rule to populate the cause association.

\[
\text{EventProfile} \\
\text{self.allInstances->forall(e | self.event_connective->exists(g | e.event_connective->includes(g))) implies e.cause->includesAll(g.input)}
\]

1. **Well-formed Fault Trees**

In defining the following well-formedness constraints and rules for fault trees, we note that a range of other structural constraints identified from the literature - e.g., that leaves are not gates and that children of events must be gates - are enforced by associations and multiplicities expressed in the UML model itself (figure 5.3).

1. **Rule to populate MinimalCutSetExpression.**

\[
\text{FaultTree} \\
\text{self.allInstances->forall(f | self.ft_event_profile->exists(e | f.ft_event_profile->includes(e) and e.type = "top")) implies f.ft_mcse.event_entity->includesAll(e.trans_cause->select ((e.trans_cause.oclIsKindOf(EventProfile) and e.trans_cause.type = "bas") or (e.trans_cause.oclType = EventSet and e.trans_cause.event_profile.type = "bas")))}
\]

2. **Constraint to ensure unique identifiers.**

\[
\text{FaultTreeAnalysis invariant} \\
\text{self.allInstances->forall(f | self.fta_event->forall(e1, e2 | not(f.fta_event->includes(e1) and f.fta_event->includes(e2) and e1 <> e2 and e1.identifier = e2.identifier)))}
\]

3. **Constraint to ensure non-primary event types have a child gate, while primary event types do not.**

\[
\text{EventProfile invariant} \\
\text{self.allInstances->forall(e | not( ((e.type = "bas" or e.type = "ext" or e.type = "und") and e.event_connective->size > 0) or ((e.type = "top" or e.type = "int") and e.event_connective->size = 0)))}
\]

4. **Constraint to ensure a fault tree contains only one top event.**

\[
\text{FaultTreeAnalysis invariant} \\
\text{self.allInstances->forall(f1, f2 | not(f1.fta_event->includes(e1) and f2.fta_event->includes(e2) and ((e1.preliminary_profile.type = "top" and e2.preliminary_profile.type = "top") or e1 <> e2 and e1.identifier = e2.identifier)))}
\]
(e1.updated_profile.type = "top" and e2.updated_profile.type = "top") and e1 <> e2))

5. Constraint to ensure that probabilities of output events are correct: i.e. a) outputs for Or-gates equate to the sum of input probabilities; and b) outputs for And-gates equate to the product of input probabilities.

a) Constraint for Or-gates

EventProfile invariant
self.allInstances->reject(self.allInstances.type = "bas" or self.allInstances.type = "ext" or self.allInstances.type = "und")->forall(e)
self.event_connective->exists(g)
self.preliminary_budget->union(self.actual_probability)->not exists(p)
(e.event_connective->includes(g) and goclType = OrGate and
((eoclType = PreliminaryEventProfile and e.preliminary_budget->includes(p) and p.probability <>
g.input.preliminary_budget.probability->sum) or
(eoclType = UpdatedEventProfile and e.actual_probability->includes(p) and p.probability <>
g.input.actual_probability.probability->sum)))

b) Constraint for And-gates

EventProfile invariant
self.allInstances->reject(self.allInstances.type = "bas" or self.allInstances.type = "ext" or self.allInstances.type = "und")->forall(e)
self.event_connective->exists(g)
self.preliminary_budget->union(self.actual_probability)->not exists(p)
(e.event_connective->includes(g) and goclType = AndGate and
((eoclType = PreliminaryEventProfile and e.preliminary_budget->includes(p) and
(g.input.event_profile->iterate(i : PreliminaryEventProfile; acc: real = 0
acc * i.preliminary_budget.probability) <> p.probability)) or
(eoclType = UpdatedEventProfile and e.actual_probability->includes(p) and
(g.input.event_profile->iterate(i : UpdatedEventProfile; acc: real = 0
acc * i.actual_probability.probability) <> p.probability))

6. Constraint(s) to ensure correct population of the (budget or actual) probability for basic events as the product of Rate and Exposure.

PreliminaryEventProfile invariant
self.allInstances->forall(e)
not(e.preliminary_rate->size = 1 and e.preliminary_exposure->size = 1 and
e.preliminary_budget.probability <> e.preliminary_rate.failure_rate * e.preliminary_exposure.period))

UpdatedEventProfile invariant
self.allInstances->forall(e)
not(e.actual_rate->size = 1 and e.actual_exposure->size = 1 and
e.actual_probability.probability <> e.actual_rate.failure_rate * e.actual_exposure.period))

7. Constraint ensuring that Rate and Exposure are stated for basic events only.

EventProfile invariant
self.allInstances->forall(e)
not(e.type <> "bas" and ((e.actual_rate->union(e.preliminary_rate)->size >= 1) or
(e.actual_exposure->union(e.preliminary_exposure)->size >= 1)))
Fault Tree Analysis Structure

Strictly, this constraint is motivated by our modelling decision to define event type as an enumerated property of EventProfile, rather than by using subtypes as we have with gates. This is because EventProfile has already been specialised through the preliminary and updated profile subtypes. Therefore, further specialisation along a different ‘dimension’ would lead to an exponential growth in classes in order to capture orthogonality - e.g., basic-preliminary, basic-updated, etc.

We note that Mason & Saeed (ibid.) contains further constraints and rules pertaining to well-formed fault trees (including an invariant to prevent cycles within the logical tree structure); again these could be restated in OCL.

ii. Fault Tree Safety-Criteria
The following define what we term ‘safety-criteria’ in the sense that they represent desirable properties of a fault tree, rather than invariants.

1. Restriction identifying single failure causes of a top event.

MinimalCutSetExpression
self.allInstances->forall(m | self.event_entity->forall(e | not (m.event_entity->includes(e) and eoclIsKindOf(EventProfile))))

2. Restriction identifying common cause failures through And-gates.

AndGate
self.allInstances->forall(g1, g2 | not((g1.input.event-profile->intersection(g2.input.event-profile)->not Empty) and g1 <> g2))

iii. Constraints Reflecting Usage
The following constraints arise from issues relating to usage, notably in terms of support for the ARP 4761 assessment process. In particular, the notion of preliminary and updated fault trees sharing events as constituents of a single Fault Tree Analysis, and the use of event profiles to reduce redundancy and promote traceability in realising this concept.

1. Constraint ensuring that preliminary and updated fault trees have the same top event.

FaultTreeAnalysis invariant
self.allInstances->forall(f | self.fta_event->forall(e1, e2 | not (f.fta_event->includes(e1) and f.fta_event->includes(e2) and e1.preliminary_profile.type = "top" and e2.updated_profile.type = "top" and e1 <> e2))

2. Restriction stating that ActualProbability of failure for the top event of the UpdatedFaultTree is less than or equal to the BudgetProbability of the PreliminaryFaultTree top event.

FaultTreeAnalysis
self.allInstances->forall(f | (f.fta_event.updated_profile->select(f.fta_event.updated_profile.type = "top").actual_probability.probability >= (f.fta_event.preliminary_profile->select(f.fta_event.preliminary_profile.type = "top").preliminary_budget.probability))
3. Constraint ensuring that updated event profiles employ the same BudgetProbability as the preliminary profile.

**Event Invariant**

\[
\text{self.allInstances->forall(e |}
\text{not (e.preliminary_profile->size = 1 and}
\text{e.preliminary_profile.preliminary_budget <>}
\text{e.updated_profile.preliminary_budget))}
\]

iv. Other Consistency Constraints

As a sample of constraints arising from modelling decisions, we capture an exclusive-or restriction that a SimpleEventDescription must belong to either a SimpleLabel, a ComposedLabel or a SynchronisationLabel.

**SimpleEventDescription Invariant**

\[
\text{self.allInstances->forall(s |}
\text{s.simpleLabel->size = 1 xor s.composedLabel->size = 1 xor s.synchronisationLabel->size = 1)}
\]

Also, and in keeping with our view on what constitutes a minimal fault tree (outlined in subsection 5.2.3.2.1) - specifically the proviso that a well-formed tree contains at least one gate - we state the following:

**FaultTree Invariant**

\[
\text{self.allInstances->forall(f | not(f.ft_and-gate->isEmpty and f.ft_or_gate->isEmpty))}
\]

v. Verification of Fault Tree Elements against the Product Data Synthesis

Given that we expressed PDS consistency checks for previous notations (and do so again for the Failure Modes and Effects Analysis structure in section 5.3), similar restrictions are not formally stated over the Fault Tree Analysis meta-model. However, the following (informal) constraints should hold for event labels:

- The entity attribute maps to a corresponding PDS Module or Function;
- The condition attribute maps to a PDS Condition for the corresponding Module or Function;
- For two events, where one is caused by the other, there exists in the PDS an OccurringIn association between corresponding conditions;
- QualifyingEntity maps to a corresponding PDS Module or Function and includes the qualified Entity within its decomposition.

5.2.3.2.3 O-Telos Implementation of FTA Base Classes

The following O-Telos code implements base class elements for the Fault Tree Analysis meta-model.

```plaintext
EventEntity in StructureElement,
SimpleClass isA
AerospaceEngineeringObject with
constraint
abstract_event_entity: $forall
  t/Token s/SimpleClass (t in s) =_> not (t
  in EventEntity)$
end

EventSet in StructureElement, SimpleClass
has_part
EventProfile in StructureElement, SimpleClass
has_property
type : EventType
```
event_connective : Gate;
cause : EventEntity
has_transitive_part
trans_cause : EventEntity
constraint
abstract_event_profile: $\forall t/\text{Token} s/\text{SimpleClass} (t \in s) \Rightarrow \neg (t \in \text{EventProfile})$

EventType in ArtifactProperty,
SimpleClass isA String with
constraint
enum_event_type: $\forall e/\text{EventType} (e \in \{	ext{"int"}, \text{"top"}, \text{"bas"}, \text{"ext"}, \text{"und"}\})$

Gate in StructureElement, SimpleClass isA AerospaceEngineeringObject with
constraint
abstract_gate: $\forall t/\text{Token} s/\text{SimpleClass} (t \in s) \Rightarrow \neg (t \in \text{Gate})$

PreliminaryEventProfile in StructureElement, SimpleClass isA EventProfile with
has_part
preliminary_budget : BudgetProbability;
preliminary_rate : Rate;
preliminary_exposure : Exposure;
preliminary_label : EventLabel

UpdatedEventProfile in StructureElement, SimpleClass isA EventProfile with
has_part
actual_probability : ActualProbability;
actual_budget : BudgetProbability;
actual_rate : Rate;
actual_exposure : Exposure;
actual_label : EventLabel

ActualProbability in StructureElement, SimpleClass isA AerospaceEngineeringObject with
has_property
probability : Real
has_structure
annotation : MatraNaturalLanguageStructure

BudgetProbability in StructureElement, SimpleClass isA AerospaceEngineeringObject with
has_property
probability : Real
has_structure
annotation : MatraNaturalLanguageStructure

Rate in StructureElement, SimpleClass isA AerospaceEngineeringObject with
has_property
failure_rate : Real
has_structure
annotation : MatraNaturalLanguageStructure

Exposure in StructureElement, SimpleClass isA AerospaceEngineeringObject with
has_property
period : Real
has_structure
annotation : MatraNaturalLanguageStructure

EventLabel in StructureElement, SimpleClass isA AerospaceEngineeringObject with
constraint
abstract_event_label: $\forall t/\text{Token} s/\text{SimpleClass} (t \in s) \Rightarrow \neg (t \in \text{EventLabel})$

ComposedLabel in StructureElement, SimpleClass isA EventLabel with
has_part
simple_description : SimpleEventDescription

SynchronisationLabel in StructureElement, SimpleClass isA EventLabel with
has_part
initial_event_description : SimpleEventDescription;
sync_event_description : SimpleEventDescription

SimpleEventDescription in StructureElement, SimpleClass isA AerospaceEngineeringObject with
has_property
type : String;
condition : String;
qualifying_entity : QualifyingEntity

QualifyingEntity in ArtifactProperty, SimpleClass isA String

AndGate in StructureElement, SimpleClass isA Gate with
has_part
input : EventSet

OrGate in StructureElement, SimpleClass isA Gate with
has_part
input : EventProfile

Event in StructureElement, SimpleClass isA AerospaceEngineeringObject with
has_property
identifier : String
has_part
preliminary_profile : PreliminaryEventProfile;
updated_profile : UpdatedEventProfile

MinimalCutSetExpression in StructureElement, SimpleClass isA AerospaceEngineeringObject with
has_part
event_entity : EventEntity

FaultTree in AssessmentStructure,
5.2.4 Relationship to the Traceability Dimensions

Trace relations involving fault trees can occur in the horizontal and vertical dimensions at inter/intra and macro/micro levels. For example, relating a PSSA fault tree for an aircraft braking system, to a PSSA fault tree for one of its components, can be said to capture an intra-macro-horizontal association; intra in the sense that our interest lies within a single system, macro in that it spans two levels of decomposition and horizontal as the artifacts concerned stem from the same phase of assessment.

Similarly, if we related the braking system fault tree to braking system requirements derived from analysis of failure conditions, this would constitute intra-micro-vertical traceability; again intra because our interest is a single system, micro as the scope is confined to a single decomposition level and vertical since traceability is between artifacts of different types (i.e., requirements and assessment).

However we make the observation that relationships between fault trees generated by PSSA and SSA are more difficult to position along the dimension axes since they depend in part on interpretation of the dependency between the preliminary and updated trees. Recall from Chapter One that revision traceability is defined as (the ability to navigate) relationships between instances of the same artifact at different stages of maturity. Therefore one view holds that preliminary trees and profiles mature into updated trees and profiles in response to findings from analysis (e.g., FMEA or Common Mode) and so relationships between them exist in the revision dimension. Yet since the stated aims of the two trees are quite different - PSSA is used to derive requirements and evaluate proposed architectures, whereas SSA provides verification that an implemented system satisfies these objectives (and so clearly one is not replacing the other in the true sense of a revision) - then it might also be argued such relationships are actually intra-micro-vertical; intra-micro as they concern descriptions within both a single system and single decomposition level and vertical as the artifacts concerned stem from different phases of assessment.

---

8 We consider the issue of revision (and more generally, version) traceability in greater detail in subsection 5.5.
5.2.5 Summary

Fault Tree Analysis is a top down method of failure analysis that is widely used throughout the aerospace sector. Accordingly, this subsection has proposed a novel meta-model supporting the main concepts underpinning Fault Tree Analysis, including common gate and event types and minimal cut set expressions. The model also reflects use within the aerospace industry, notably the assessment process prescribed by ARP 4754/4761. It therefore enables events to be shared between a preliminary fault tree (developed during PSSA) - expressing budget safety requirements - and its updated counterpart intended to show realisation of these requirements (developed during SSA). Event sharing is accomplished by means of ‘event profiles’ which eradicate much of the redundancy between preliminary and updated trees, whilst helping maintain consistency and increase traceability.

A number of well-formedness constraints were expressed over the logical fault tree structure, together with safety-criteria for identifying common cause and single failures. We also introduced a ‘light-weight’ approach to the formalisation of event labels by typing events according to a taxonomy from the literature and by specifying their content at a level of granularity that allows for their verification against the Product Data Synthesis.

We demonstrate application of the Fault Tree Analysis meta-model to a case study derived from ARP 4761 in subsection 6.3. This case study will also feature a meta-model for the complementary Failure Modes and Effects Analysis technique to be introduced in the next section. A full evaluation of the Fault Tree meta-model is discussed in Chapter Seven.
5.3 Failure Modes and Effects Analysis Structure

5.3.1 Introduction
In this section we propose a structure capturing elements of Failure Modes and Effects Analysis, a technique that tracks the effects of component or functional failures within a system to determine their ultimate consequences and which like FTA, is also widely used in the aerospace industry.

5.3.2 Motivation
Failure Modes and Effects Analysis (FMEA) was first applied to aeronautics in the 1960s and has since been used, among others, on projects ranging from Concorde to the lunar module (Bussolini, 1971); it continues to be a mainstay of safety assessment for contemporary aircraft (FAA, 2001). Accordingly, a number of aerospace related standards (or guidelines) prescribe procedures for conducting FMEA and FMECA (Failure Modes, Effects and Criticality Analysis - an extension to FMEA which also takes into account the importance of each failure), either specifically as in the case of IEC (1985), or as part of the overall safety process (EUROCAE, 1996b). Like Fault Tree Analysis, application of FMEA in the assessment of software has also been well documented (cf. Fragola & Spahn, 1973; Reifer, 1979).

Recall from subsection 5.2.2 that Failure Modes and Effects Analysis is an inductive technique in the sense that users are attempting to anticipate potential failures, so their source(s) can be eliminated. It is seen as a complimentary approach to Fault Tree Analysis such that a fault tree used to determine the causes of a particular hazard may utilise failure rates from relevant FMEAs9. Since we regard it appropriate to include both deductive and inductive approaches within the MATrA framework, Failure Modes and Effects Analysis serves as our inductive example.

We also stated in 5.2.2 that the results of several safety analysis technique are represented in tabular form. Again, as our aim is to incorporate examples of both graphical (as demonstrated by the previous FTA structure) and table-based formats, FMEA provides our example of the latter.

5.3.2.1 Failure Modes and Effects Analysis Overview
As previously indicated, Failure Modes and Effects Analysis allows analysts to investigate potential ways in which individual components or functions within a target system might fail. However, unlike Fault Tree Analysis (whose symbols and element types remain largely standard), a raft of different FMEA analyses with varying procedures, intent and reporting formats have emerged; the corollary being that little commonality exists between the various industries in which they are used. We therefore adhere to the format presented in ARP 4761 for aerospace engineering.

It is common to distinguish two types of FMEA, functional (where investigation is of the lowest level assemblies in a system - typically functional or block-diagram level) and piece-part (where investigation is at the level of individual components within an assembly). As shown by the case study

9 Wilson & McDermid (1995) identify a number of interdependencies among safety assessment techniques; for FTA and FMEA these are: i) basic events exists as effects; ii) probabilities are consistent with rates.
example in ARP 4761 (Appendix L), piece-part analyses are normally performed to refine failure rates produced from functional FMEA that do not allow a system or component to meet FTA budget failure probabilities.

Again procedural issues on conducting both forms of FMEA are outside the scope of this thesis; interested readers are instead referred to (EUROCAE, 1996b) and also to (NASA, 1994). However, a typical process framework will involve the definition of components and operating states, identification of their failure modes, determination of effects and investigation of factors for detection and protection, as well as some basis for making recommendations.

While the content of Failure Modes and Effects Analysis tables differs between industries, certain concepts are standard among all safety\(^\text{10}\) related variants; these reflect the template procedure outlined above and include columns recording component or function, known failure modes and the effects of each failure. Other columns that may be added include cause of failure, frequency of occurrence (based on historical data and service experience), severity (indicating how serious the failure would be), means for detection or mitigating against the failure and finally comments and recommendations.

ARP 4761, the FMEA reporting format considered in this thesis, suggests the following as possible ‘worksheet’ columns for functional and piece-part analyses:

<table>
<thead>
<tr>
<th>Functional FMEA</th>
<th>Piece-Part FMEA</th>
</tr>
</thead>
<tbody>
<tr>
<td>function name</td>
<td>component name</td>
</tr>
<tr>
<td>failure mode</td>
<td>component type</td>
</tr>
<tr>
<td>failure rate</td>
<td>failure mode</td>
</tr>
<tr>
<td>flight phase</td>
<td>failure mode rate</td>
</tr>
<tr>
<td>failure effect</td>
<td>failure effect</td>
</tr>
<tr>
<td>detection method</td>
<td>detection method</td>
</tr>
<tr>
<td>comments</td>
<td>comments</td>
</tr>
</tbody>
</table>

Table 5.3 - ‘ARP 4761 Functional and Piece-Part FMEA Contents’

Note: Failure effect codes may be assigned and used in place of effect descriptions to simplify the table presentation. A detailed description of each effect code is then included in the accompanying report.

Table 5.4 contains a Functional FMEA fragment for an aircraft Brake System Control Unit power supply (P/S) which is again taken from ARP 4761.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Failure Mode</th>
<th>Flight Phase</th>
<th>Failure Rate (E-6)</th>
<th>Failure Effect</th>
<th>Detection Method</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>+5 Volt Power Supply</td>
<td>+5 volt out of spec</td>
<td>All</td>
<td>0.2143</td>
<td>P/S Shutdown</td>
<td>The Power Supply Monitor is tripped.</td>
<td>The channel fails.</td>
</tr>
</tbody>
</table>

Table 5.4 - *(Functional) FMEA Fragment for Break System Control Unit Power Supply’

---
\(^\text{10}\) FMEA also has strong associations with reliability engineering; indeed Leveson (1995) argues it is strictly a reliability analysis that is inappropriate for use in a safety process.
5.3.3 Tracing Safety Properties in MATrA: An FMEA Model

This subsection proposes a meta-model for the representation of tables reporting FMEA results, allowing their integration into the MATrA traceability framework.

5.3.3.1 Concepts

As stated previously, industry variations prevent our developing a definitive FMEA meta-model. Instead we concentrate on supporting the suggested functional and piece-part worksheet formats summarised in table 5.3.

Given that results are table-based, we are able to capture well-formedness of FMEAs using multiplicity constraints expressed over meta-model elements. However, checks verifying consistency of table entries against the Product Data Synthesis are necessary and are therefore featured below.

5.3.3.2 FMEA Meta-model Definitions

We now introduce the UML meta-model representing elements of piece-part and functional FMEA tables (5.3.3.2.1), together with OCL constraints to verify table entries against the Product Data Synthesis (5.3.3.2.2) and O-Telos implementation of base classes (5.3.3.2.3).

5.3.3.2.1 FMEA Meta-model

Figures 5.4 (elements) and 5.5a/b (associations) depict the FMEA UML meta-model. Its abstract core class (FailureMode&EffectsAnalysis), with subject module (subject_module) and other configuration attributes instantiates the AssessmentStructure meta-class, as do the functional (FunctionalFMEA) and piece-part (PiecePartFMEA) specialisations. Each is defined as an aggregation of StructureElement meta-classes representing FMEA primitives.

The FailureMode&EffectsAnalysis class is an aggregation of elements common to both functional and piece-part analyses, namely failure modes (FailureMode) and effects (FailureEffect), both of which are defined as specialisations of the built-in String class (not shown). Means to record failure detection and comments are also provided through the MATrA Natural Language Structure (MatraNLS).

Besides the inherited elements described above, a FunctionalFMEA is defined as an aggregation of function identifier (FunctionID), failure rate (FailureRate) and flight phase (FlightPhase) elements. FunctionID and FailureRate specialise String and Real (domain <=1) respectively (again not shown), while FlightPhase is an enumerated type - a specialisation of String, whose instantiations are restricted to 'Taxi', 'Take off to Rotation', 'Landing Roll', 'Rejected Take Off' and 'Climb', etc.; or alternatively, the universal 'All' (i.e., all ground and airborne phases).

FunctionalFMEA also includes a number of classes expressly to preserve grouping of repeated elements within the table format. That is, to relate failure behaviours to functions, and to relate failure modes

11 This arises from the fact that each function may have many failure modes such that the effects and rates of each failure differ for, and must therefore be grouped by, flight phase; detection is grouped on the basis of mode rather than flight phase.
Figure 5.4 - ‘FMEA Structure : Elements’

to effects and rates of occurrence for each flight phase. Thus, a FunctionalFMEA is described as an
aggregation of function failure descriptions (FunctionFailureDescription) which in turn comprise a single
FunctionID and one or more function failure mode descriptions (FunctionFailureModeDescription).
Accordingly, instances of the latter include a single FailureMode and one or more flight phase failure
descriptions (FlightPhaseFailureDescription) allowing the effects (FailureEffect) and rate (FailureRate) of
failures to be described by flight phase.

Likewise, in addition to common FMEA elements, a PiecePartFMEA includes String class specialisations
(again not shown) for component (Component), and component type (ComponentType), together with the
failure mode rate (FailureModeRate) element - a subtype of Real also in range \(< 1\) (not shown).

As with its functional counterpart, the PiecePartFMEA further includes classes to preserve grouping
within tables; each component can have many failure modes, which in turn can have many effects.
Therefore, a PiecePartFMEA is as an aggregation of component failure descriptions
(ComponentFailureDescription) with (single) Component and ComponentType elements and one or more
component failure mode descriptions (ComponentFailureModeDescription); the latter includes single
FailureMode and FailureModeRate and, one or more FailureEffect elements.
Failure Modes and Effects Analysis Structure

Figure 5.5-a - 'Functional FMEA Structure : Associations'

Figure 5.5-b - 'Piece-Part FMEA Structure Associations'
5.3.3.2.2 OCL Constraints

The tabular format in which results of Failure Modes and Effects Analyses are presented and our ability to preserve this structure through the models presented in subsection 5.3.3.2.1 means that no well-formedness constraints are necessary. However, a number of checks verifying FMEA elements against the Product Data Synthesis may be defined as follows:

1. The FMEA subject module exists in the PDS.

   \[
   \text{FailureMode&EffectsAnalysis invariant} \quad \text{self.allInstances->forall(f | self.bElementAEO.build_element->exists(be | f.subject_module = be.module_name))}
   \]

2. a) PiecePartFMEA table components exist in the PDS as sub-modules of the subject module (a) and FunctionalFMEA identifiers exist in the PDS as functions encapsulated by the subject module (b).

   a) \text{Component invariant} \quad \text{self.allInstances->forall(c | self.bElementAEO.build_element->exists(m | self.piecePartFMEA.bElementAEO.build_element->exists(be | be.module_name = c.piecePartFMEA.subject_module and c = m.module_name and be.hasSubmodule.target->includes(m))))}

   b) \text{FunctionID invariant} \quad \text{self.allInstances->forall(f | self.bElementAEO.build_element->exists(e | self.functionalFMEA.bElementAEO.build_element->exists(be | be.module_name = f.functionalFMEA.subject_module and f = e.function_name and be.encapsulates.target->includes(e))))}

3. For components (a) and functions (b), failure modes map to Conditions of the corresponding PDS Module or Function.

   a) \text{FailureMode invariant} \quad \text{self.allInstances->select(self.piecePartFMEA->size = 1)->forall(f | self.bElementAEO.build_element->exists(c | self.piecePartFMEA.bElementAEO.build_element->exists(be | self.piecePartFMEA.bElementAEO.build_element.hasSubmodule.target->exists(m | f.piecePartFMEA.subject_module = be.module_name and be.hasSubmodule.target->includes(m) and m.module_name = f.componentFailureModeDescription.componentFailureDescription.com_name and m.hasCondition.target->includes(c) and f = c.condition.label ))))}

   b) \text{FailureMode invariant} \quad \text{self.allInstances->select(self.functionalFMEA->size = 1)->forall(f | self.bElementAEO.build_element->exists(c | self.functionalFMEA.bElementAEO.build_element->exists(be | self.functionalFMEA.bElementAEO.build_element.encapsulates.target->exists(p | f.functionalFMEA.subject_module = be.module_name and f.functionalFMEA.subject_module = be.module_name and)}
be.encapsulates.target->includes(p) and
p.function_name = e.functionFailureModeDescription.functionFailureDescription.fnc_id and
p.hasCondition.target->includes(c) and f = c.condition.label)))

4. The PDS contains conditions corresponding to (a) component or (b) function failure effects; these
may be effects on the same Module or Function over which the causal FailureMode is expressed, or
else on another Module or Function within the same host (i.e., subject_module), or else effects
impacting on the subject_module itself (note this is not an invariant).

a) FailureEffect
self.allInstances->select(self.piecePartFMEA->size =1)->forall(e l
self.bElementAEO.build_element->exists(c l
self.piecePartFMEA.bEmodelAEO.build_element->exists(l
self.piecePartFMEA.bEmodelAEO.build_element.hasSubmodule.target->exists(m l
self.piecePartFMEA.bEmodelAEO.build_element.hasSubmodule.target.hasCondition.target->exists(c l
self.piecePartFMEA.bEmodelAEO.build_element.hasSubmodule.target->exists(m' l
e.piecePartFMEA.subject_module = be.module_name and
be.hasSubmodule.target->includes(m) and
m.module_name = e.componentFailureModeDescription.componentFailureDescription.com_name and
m.hasCondition.target->includes(c) and
c'.condition_label = e.componentFailureModeDescription.com_fail_mode and
(be.hasCondition.target->includes(c) or m.hasCondition.target->includes(c) or
be.hasSubmodule.target->includes(m') and m'.hasCondition.target->includes(c)) and

b) FailureEffect
self.allInstances->select(self.functionalFMEA->size =1)->forall(e l
self.bElementAEO.build_element->exists(c l
self.functionalFMEA.bEmodelAEO.build_element->exists(l
self.functionalFMEA.bEmodelAEO.build_element.encapsulates.target->exists(p l
self.functionalFMEA.bEmodelAEO.build_element.encapsulates.target.hasCondition.target->exists(c l
self.functionalFMEA.bEmodelAEO.build_element.encapsulates.target->exists(p' l
e.functionalFMEA.subject_module = be.module_name and
be.encapsulates.target->includes(p) and
p.function_name =
e.flightPhaseFailureDescription.functionFailureModeDescription.functionFailureDescription.fnc_id and
p.hasCondition.target->includes(c') and
c'.condition_label = e.flightPhaseFailureDescription.functionFailureModeDescription.fnc_fail_mode and
(be.hasCondition.target->includes(c) or p.hasCondition.target->includes(c) or
be.encapsulates.target->includes(p') and p'.hasCondition.target->includes(c)) and

5.3.3.2.3 O-Telos Implementation of FMEA Base Classes
The following O-Telos code implements FMEA meta-model base class elements.

Definition of Structure Elements
Component in StructureElement,
SimpleClass is A String,
AerospaceEngineeringObject end
ComponentType in StructureElement,
SimpleClass is A String,
AerospaceEngineeringObject end
FailureModeRate in StructureElement,
SimpleClass is A Real,
AerospaceEngineeringObject with
constraint
fmr_in_range: $ forall f/
FailureModeRate (f <= 1)$
end
FailureMode in StructureElement,
SimpleClass is A String,
AerospaceEngineeringObject end
Structuring Safety Assessment and Product Management Artifacts

5.3.4 Relationship to the Traceability Dimensions

The case study in subsection 6.3 demonstrates traceability between development and assessment artifacts; in particular it shows how components from a Circuit Diagram trace to corresponding entries in FMEA tables. Such relationships constitute intra-micro vertical traceability; intra-micro in the sense...
that the Circuit Diagram and FMEA table both concern the same system and level of decomposition and vertical since traceability is between artifacts of different types (development and assessment).
Moreover, effects of each failure from the FMEA table may trace to basic events of fault trees; where for instance both artifacts form part of System Safety Assessment, such relationships are located on the (intra-micro) horizontal axis, as are relationships between piece-part and functional FMEAs.

5.3.5 Summary
Failure Modes and Effects Analysis has been widely used throughout the aerospace sector for nigh on forty years. Accordingly, its inclusion within the MATrA traceability framework allows us to demonstrate support for an inductive safety technique whose results are recorded in tabular form (as opposed to the hitherto featured graphical, prose or program code structures).

A novel meta-model was proposed capturing elements for both functional and piece-part FMEA formats as prescribed in the ARP 4761 guidelines for safety assessment of civil airborne systems and equipment. A set of mapping rules over the model for verifying table entries against the Product Data Synthesis was also introduced.

Section 6.3 features a contiguous case study demonstrating use of the FMEA structure (in conjunction with Fault Tree Analysis and Circuit Diagram structures), whilst a full evaluation is again provided in Chapter Seven.
5.4 Programme Evaluation & Review Technique Structure

5.4.1 Introduction

In this section we introduce our final notation meta-model. The proposed structure captures the graphical syntax of the Programme Evaluation & Review Technique which represents inter-relationships between the timing of events and project activities in the form of a network.

5.4.2 Motivation

The Programme Evaluation & Review Technique (PERT) aids in the management, planning and control of projects. It was devised by the United States Navy during the 1950s in an attempt to cut project lead times and is credited with subsequently saving two years in development of the Polaris missile system. PERT graphically demonstrates inter-relationship between the various tasks making up a project, clearly identifying its 'critical' parts. According to Lucey (1992), the technique is most beneficial and so normally applied to projects that are large, complex and subject to various time and cost constraints - characteristics common to projects in the aerospace industry. Moreover, potential usage within that particular sector is life-cycle wide, ranging from the management of requirements and design activities, to scheduling of aircraft maintenance tasks.

Aside from its relevance to our domain of interest, we are also motivated in featuring PERT by a desire to cover all categories of notation and technique used by aerospace practitioners (development and assessment having being covered in Chapter Four and previous sections of Chapter Five respectively).

5.4.2.1 Programme Evaluation & Review Technique Overview

Two types of element are fundamental to all PERT networks; activities (or 'basic activities') - denoted as arcs (and labeled with a shortened description, or alpha-numeric code) - are tasks which take time and resources to perform. The head of an arc indicates the point at which a task ends and the tail, where it begins. We note that arcs are normally drawn from left to right and that they imply precedence only; i.e., length is not proportional to duration.

The other main constituent of PERT networks are events - denoted as circles or nodes (and progressively labeled with integer sequence numbers, together with an optional descriptor) - which mark the point in time an activity or activities start or finish (i.e., they represent the achievement of a specific stage of a project).

In addition to basic activities and events, networks can also include the notion of 'dummy activities' - denoted as broken or dotted arcs. These consume neither time nor resources, but rather show logical dependencies between activities, either to preserve precedence (as is the case with examples featured in this thesis), or to prevent the breach of rules for drawing networks (Lucey, ibid.). Of these rules, the principal structural constraints are as follows:

- A network should have a single point of entry or start event and single point of exit or finish event.
Every activity must have one preceding or 'tail' event and one succeeding or 'head' event; many activities may share the same head event or the same tail event, although no activity can share both the same tail and head event. Dummy activities are used to preserve this constraint.

Cycles or series of activities leading back to the same event are not permissible since a founding tenet of networks is that they indicate the progression of activities onwards in time.

As shown in figure 5.6, a PERT network is the combination of activities, dummy activities and events in logical sequence according to the constraints outlined above:

![PERT Network Diagram](image)

**Figure 5.6 - 'Example of a Basic PERT Network'**

The logical sequence for the network in figure 5.6 is summarised in table 5.5. Note, a dummy activity (3-6) is used because of the preceding activity requirements of activity L; if activities E and H were not specified to precede this activity, then the dummy would not be necessary.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Preceding Activity</th>
<th>Activity</th>
<th>Preceding Activity</th>
<th>Activity</th>
<th>Preceding Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-</td>
<td>F</td>
<td>C</td>
<td>L</td>
<td>E, H, G, K</td>
</tr>
<tr>
<td>B</td>
<td>-</td>
<td>G</td>
<td>C</td>
<td>M</td>
<td>E, H</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>H</td>
<td>C</td>
<td>N</td>
<td>L, M</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>J</td>
<td>B, D</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E</td>
<td>A</td>
<td>K</td>
<td>F, J</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table 5.5 - 'Activities & Relationships for Basic PERT Network'**

Once the network logic has been established, activity durations can then be added enabling the overall completion time for a project to be established and its *critical path* calculated; i.e., the shortest time in which a project can be completed based on the chain of activities with longest duration times. This may or may not run through dummies.
In order to assess total project time, it is necessary to specify the Earliest and Latest Start Times (EST/LST) for each activity. The EST is the earliest point at which a succeeding activity can start; for a head event this is calculated by adding onto the EST of the tail event, its linking activity duration. Where two or more routes feed into an event, the longest route time is taken. This practice is repeated from event zero at time point zero, onwards to the end of the network (termed a forward pass).

The critical path is established by calculating LSTs for each activity. In other words, the latest time a preceding activity can finish without increasing the duration. Calculating Latest Start Times involves starting at the finish event and working backwards through the network, deducting each activity duration from the previously calculated LST (termed a backward pass). Where two or more activities lead from an event, then the lowest number is taken. Activities on the critical path are normally indicated by two small traverse lines placed across their arcs (note also that ESTs and LSTs for each head and tail event on the critical path are identical). These refinements are illustrated in figure 5.7.

In addition to enforcing structural constraints and determining the critical path, tools supporting the drawing of PERT networks often allow annotation of activities with the type and quantity of each resource they consume. Alternatively, this information may be shown separately as a table.

5.4.3 Tracing Programme Evaluation & Review Technique Networks in MATrA:

A PERT Model

In this subsection we propose a meta-model for the representation of PERT networks, allowing their integration into the MATrA traceability framework.

5.4.3.1 Concepts

Our meta-model for the Programme Evaluation & Review Technique captures the graphical syntax described in 5.4.2.1. Hence it contains elements representing activities (both dummy and basic) and events (including earliest and latest start times); activities may also be annotated with the resources they consume.

A selection of OCL constraints expressing the semantics of the rules stated (informally) in 5.4.2.1 are also specified. These govern both the drawing of networks, verification of earliest and latest start-times and derivation of the critical path.
5.4.3.2 PERT Meta-model Definitions

This subsection introduces the UML class diagram representing our PERT meta-model (5.4.3.2.1), together with OCL constraints (5.4.3.2.2) and O-Telos implementation of base classes (5.4.3.2.3).

5.4.3.2.1 PERT Meta-model

Figures 5.8 depicts elements and associations of the Programme Evaluation & Review Technique meta-model. Its core class, ProgrammeEvaluationReviewTechnique with subject module (subject_module), model name (model_name) and other configuration attributes instantiates the ProductManagementStructure metaclass.

Figure 5.8 - 'Programme Evaluation & Review Technique Structure : Elements and Associations'

A ProgrammeEvaluationReviewTechnique structure is defined as an aggregation of zero-or-more dummy activities (DummyActivity), one-or-more basic activities (BasicActivity) and two-or-more events (Event). Both DummyActivity and BasicActivity are specialisations of the abstract activity (Activity) class which for ease of manipulation, factors the common head event (head_event) and tail event (tail_event) aggregation associations into a general supertype. Basic activities have attributes for description and duration, as well as a boolean critical path (critical_path) indicator; they may also be described in terms of resources (Resource), with resource_type (resource_type) and amount attributes.

The Event class contains sequence number (sequence_no) and description, together with earliest start time (EST) and latest start time (LST) attributes; a reflexive association (transitive_event) allows for recording
of transitive closure events and exists to prevent cycles in networks using an appropriate constraint (as shown in the following subsection).

5.4.3.2 OCL Constraints

The following constraints are expressed over the UML meta-model described above and based on the informal restrictions stated in 5.4.2.1:

1. A network must have only one entry point.

   Programme Evaluation Review Technique invariant
   self.allInstances->forall(p | (p.nwk_dummy.tail_event->union(p.nwk_basic.tail_event)) -
   (p.nwk_dummy.head_event->union(p.nwk_basic.head_event))->size = 1)

2. A network must have only one exit point.

   Programme Evaluation Review Technique invariant
   self.allInstances->forall(p | (p.nwk_dummy.head_event->union(p.nwk_basic.head_event)) -
   (p.nwk_dummy.tail_event->union(p.nwk_basic.tail_event))->size = 1)

3. No two network activities share the same head and tail events.

   Activity invariant
   self.allInstances->forall(a1, a2 | not(a1.tail_event = a2.tail_event and
   a1.head_event = a2.head_event and a1 <> a2))

4. A network must not contain any 'cycles'.

   Event rule for determining transitive closure of events
   self.allInstances->forall(e1, e2 | self.activity->exists(a1 | a1.tail_event = e1 and a1.head_event = e2) or
   self.allInstances->exists(e3 | self.activity->exists(a2 | a2.tail_event = e1 and a2.head_event = e3 and
   e3.transitive_event->includes(e2)))) implies e1.transitive_event->includes(e2)

   constraint to prevent cycles within networks

   Event invariant
   self.allInstances->forall(e | not(e.transitive_event->includes(e)))

5. Earliest Start Times (EST) are correct:

   a) where two events are connected by an Activity, the head event EST is calculated by adding onto the
      EST of the tail event, the duration of the Activity.

   Event invariant
   self.allInstances->forall(e->reject(self.activity.tail_event->not exists(e' | e.activity.tail_event->includes(e')))) |
   self.activity->select(self.activity.oclType = BasicActivity)->exists(a | a.head_event->includes(e) and e.EST = (a.tail_event.EST + a.duration))
b) where two or more routes arrive at an Event, the longest route time is taken.

**Event Invariant**
```oclm
self.allInstances->forall(e | self.activity->select(self.activity.oclType = BasicActivity)->forall(a | not (a.head_event->includes(e) and e.EST < (a.tail_event.EST + a.duration))))
```

6. Latest Start Times (LST) are correct:

a) where two events are connected by an Activity, the tail event LST is calculated by subtracting from the LST of the head event, the duration of the Activity.

**Event Invariant**
```oclm
self.allInstances->forall(e | reject(self.activity.head_event->not exists(e' | e.activity.head_event->includes(e')))) | self.activity->select(self.activity.oclType = BasicActivity)->exists(a | a.tail_event->includes(e) and e.LST = (a.head_event.LST - a.duration)))
```

b) where the tails of two or more activities join an Event, the shortest route time is taken.

**Event Invariant**
```oclm
self.allInstances->forall(e | self.activity->select(self.activity.oclType = BasicActivity)->forall(a | not (a.tail_event->includes(e) and e.LST > (a.head_event.LST - a.duration))))
```

7. Rule to identify critical path activities (based on ESTs and LSTs verified by rules 5 and 6).

**BasicActivity**
```oclm
self.allInstances->forall(a | a.tail_event.EST = a.tail_event.LST and a.head_event.EST = a.head_event.LST implies a.critical_path = True)
```

Note: we do not specify a constraint verifying the subject_module attribute of the ProgrammeEvaluationReviewTechnique class against the PDS as similar rules have been stated for previous models.

5.4.3.2.3 O-Telos Implementation of PERT Base Classes

The following O-Telos code implements the PERT meta-model base class elements.
5.4.3.3 PERT Worked Example

To illustrate the PERT meta-model, we revisit an example introduced in subsection 5.4.2.1 (figure 5.6). This network has been revised - as shown in figure 5.9 - with the addition of timing properties and the critical path (activities A, C, G, L, N with a duration of 29 days).

![PERT Network - Worked Example](image-url)

Figure 5.9 - 'PERT Network - Worked Example'

O-Telos instantiation of the PERT meta-model capturing information contained in figure 5.9 (together with additional data on resource usage) is as follows:
Structuring Safety Assessment and Product Management Artifacts

```
est
  EST : 17
  lst
  LST : 17
end

Event5 in Event, Token with sequence_no
  sequenceNo : 5
  est
  EST : 19
  lst
  LST : 22
end

Event6 in Event, Token with sequence_no
  sequenceNo : 6
  est
  EST : 23
  lst
  LST : 23
end

Event7 in Event, Token with sequence_no
  sequenceNo : 7
  est
  EST : 25
  lst
  LST : 25
end

Event8 in Event, Token with sequence_no
  sequenceNo : 8
  est
  EST : 29
  lst
  LST : 29
end

ActivityA in BasicActivity, Token with description
  Description : “A”
  duration
    Duration : 9
  critical_path
    criticalPath : True
tail_event
  tailEvent : Event0
head_event
  headEvent : Event0
activity_resource
  activityResource : ActivityAResource
end

ActivityB in BasicActivity, Token with description
  Description : “B”
  duration
    Duration : 3
  critical_path
    criticalPath : False
tail_event
  tailEvent : Event0
head_event
  headEvent : Event2
activity_resource
  activityResource : ActivityBResource
end

ActivityC in BasicActivity, Token with description
  Description : “C”
  duration
    Duration : 8
  critical_path
    criticalPath : True
tail_event
  tailEvent : Event1
```
ActivityJ in BasicActivity, Token with
description
  Description : "J"
duration
  Duration : 4
critical_path
    criticalPath : False
tail_event
  tailEvent : Event2
head_event
  headEvent : Event5
activity_resource
  activityResource : ActivityJResource
end
ActivityK in BasicActivity, Token with
description
  Description : "K"
duration
  Duration : 1
critical_path
    criticalPath : False
tail_event
  tailEvent : Event5
head_event
  headEvent : Event6
activity_resource
  activityResource : ActivityKResource
end
ActivityL in BasicActivity, Token with
description
  Description : "L"
duration
  Duration : 2
critical_path
    criticalPath : True
tail_event
  tailEvent : Event6
head_event
  headEvent : Event7
activity_resource
  activityResource : ActivityLResource
end
ActivityM in BasicActivity, Token with
description
  Description : "M"
duration
  Duration : 3
critical_path
    criticalPath : False
tail_event
  tailEvent : Event3
head_event
  headEvent : Event7
activity_resource
  activityResource : ActivityMResource
end
ActivityN in BasicActivity, Token with
description
  Description : "N"
duration
  Duration : 4
critical_path
    criticalPath : True
tail_event
  tailEvent : Event7
head_event
  headEvent : Event8
activity_resource
  activityResource : ActivityNResource
end
DummyActivity3to6 in DummyActivity,
Token with
tail_event
  tailEvent : Event3
head_event
  headEvent : Event6
end
ActivityAResource in Resource, Token
  with
resource_type
    resourceType : "labour"
amount
  Amount : 20
end
ActivityBResource in Resource, Token
  with
resource_type
    resourceType : "labour"
amount
  Amount : 10
end
ActivityCResource in Resource, Token
  with
resource_type
    resourceType : "labour"
amount
  Amount : 10
end
ActivityDResource in Resource, Token
  with
resource_type
    resourceType : "labour"
amount
  Amount : 20
end
ActivityEResource in Resource, Token
  with
resource_type
    resourceType : "labour"
amount
  Amount : 20
end
ActivityFResource in Resource, Token
  with
resource_type
    resourceType : "labour"
amount
  Amount : 5
end
ActivityGResource in Resource, Token
  with
resource_type
    resourceType : "labour"
amount
  Amount : 5
end
ActivityHResource in Resource, Token
  with
resource_type
    resourceType : "labour"
amount
  Amount : 5
end
ActivityJResource in Resource, Token
  with
resource_type
    resourceType : "labour"
amount
  Amount : 5
end
ActivityKResource in Resource, Token
  with
resource_type
    resourceType : "labour"
amount
  Amount : 5
end
5.4.4 Summary

The Programme Evaluation & Review Technique is widely used throughout the aerospace industry and indeed all sectors where means are required to manage large-scale projects. On that basis, we chose it as our representative product management technique for the MATrA traceability framework.

A novel meta-model capturing the graphical syntax of PERT networks was proposed, together with a selection of well-formedness constraints; a partial O-Telos implementation of the model was then populated to demonstrate a worked example.

Again, Chapter Seven includes an evaluation of the PERT structure.
5.5 The MATrA Configuration Model

5.5.1 Introduction

So far, while considering a range of traceability structures (including meta-models of notations, plus the PDS to maintain consistency) we have disregarded means of managing their evolution. Therefore in this subsection we propose the MATrA Configuration Model (MCM), a structure supporting traceability across both version types discussed in Chapter One - i.e., revisions and variants. We also demonstrate integration of MCM with existing work on the representation and expression of argumentation. This is based on the wildly held belief that version management, together with maintenance of rationale are the main change management tools at an engineer's disposal. Aspects of the model are illustrated using a worked example from the aerospace domain.

5.5.2 Motivation

Chapter One (subsection 1.2) identified the ability to trace across versions of artifacts as a major topic of interest among aerospace practitioners; recall our conceptualisation of traceability according to four axes - horizontal, vertical, revision and variant - such that the first three provide dimensions for a cube recording links between project artifacts, while the fourth relates artifacts across different projects (i.e., cubes) and across product families within the same project. The terms revision and variant merely capture different versioning intent; revisions replace one another whereas variants coexist.

Reader understanding of the significance of version traceability to aerospace practitioners will profit from a brief insight into the industry itself. We couch this discussion (which considers revisions and variants, along with the related topic of alternatives) in the context of civil aviation as it provides the focus for our worked example in subsection 5.5.3.3.

- Revisions

Passenger aircraft are among the most complex engineering projects to be undertaken by private corporations. They must be both economical and dependable; safe and reliable of course, but also available and maintainable. Put simply, airlines do not earn money from grounded aircraft! The cost of developing systems with multiple redundancies and sophisticated in-built diagnostics that help realise these objectives mean manufacturers continually strive to extend the lifetime of an aircraft by keeping it competitive. This is accomplished by evolving the basic design to take account of technological advances in aerodynamics, materials, avionics, etc. For example, the 747 has undergone four major refinements since its maiden flight in 1969, the latest revision, the 747-400, being released in 1989 as a replacement for its predecessor the 747-300.

- Variants

The kinds of aircraft demanded by major carriers can be broadly defined in terms of two macro-level parameters - namely passenger load (approximately 100 to 500) and range (essentially targeting short, extended, long and ultra long routes). Accordingly, the dominant manufacturing philosophy has been to engineer 'families' of aircraft (based on common components) aligned to these demands, from which
variants of models within a given family may be derived over time to fill a particular market niche. For instance, apart from military or 'one-off' modifications (e.g., to carry the space shuttle), various 747 revisions have spawned longer range, increased take-off-weight, high density shuttle and swing nosed freighter variants of what is effectively the same basic design. Moreover, to maximise resources (and share artifacts between projects), manufacturers will often proceed simultaneously with development of two or more derivatives - as is the case with the Airbus A340-500 and A340-600.

From these examples, we can discern two distinct forms of variant which differ according to the time dimension; serial development where artifacts from one or more existing projects are applied to a new project - the goal being to 'maximise artifact reuse' (be it whole or partial), and parallel development whereby projects proceed concurrently - the goal being to 'minimise artifact redundancy'.

- **Alternatives**

  We regard support for tracing alternatives as part of the overall versions topic area. During development, different definitions of modules are often proposed in response to a decision problem (commonly referred to as option space exploration), while some justification or rationale must normally be given for the eventual choice. This continues to be an important research strand within DCSC, but is not considered further here. Instead, we consolidate MATrA with an existing technique for tracing alternatives using a global status property (incorporated into AerospaceTraceabilityEntity) - see Appendix B, Part 1. Riddle & Saeed (2000) have shown this to be a more effective way of structuring options than Klein's approach (subsection 3.2.3, figure 3.4) based on associations (has-option, is-best-option-for, etc.) Readers are also referred to Appendix B, Part 2 which includes an Argumentation Structure (again influenced by Riddle & Saeed, ibid.) allowing rationale to be appended to any MATrA element (as shown by the example in 5.5.3.3).

5.5.2.1 **Configuration Management Overview**

Configuration describes the control of change in evolving systems. In this subsection we briefly investigate some key concepts from configuration management literature that have shaped the MCM.

Configuration management theory distinguishes between the notions of product space and version space. These represent descriptions of the target system (which in MATrA constitutes both the PDS and Workspace) and the organisation of versions of these descriptions respectively. A version space can be said to capture the state of an evolving product (and constituent artifacts of that product) at different stages of maturity (i.e., along the time dimension).

The difference between two versions of a managed artifact is called a delta (also abbreviated to the Delta symbol). For instance, a delta between versions \( v_1 \) and \( v_2 \) of an artifact can be defined in terms of properties specific to both \( v_1 \) and \( v_2 \); i.e., \( \Delta (v_1, v_2) = (v_1 \setminus v_2) \cup (v_2 \setminus v_1) \). This is termed a symmetric delta. In cases where an artifact undergoes major change with many successive versions created, then the common properties may become progressively smaller and smaller (Conradi & Westfechtel, 1998).
The evolution history (audit trail) of versioned artifacts can be represented in terms of a graph (typically a sequence) comprising a set of versions connected by relationships of a single type. These are termed successor relationships such that \( v_2 \text{ succeeds } v_1 \) means that \( v_2 \) has been derived from \( v_1 \).

Different product versions can be produced by combining different constituent artifacts and different versions of constituents into what is termed a configuration. The degree of flexibility permitted in producing such combinations depends on the version model underlying the configuration process. Conradi & Westfechtel (1998) distinguish three such models (figure 5.10) classified by their selection order, specifically: i) product first - the product structure is selected first, followed by the versions of each module (such that each configuration has the same structure); ii) version first - the product version is selected first which uniquely determines module versions (allowing different versions to be structured differently, though in the event of change, new versions must be created for all 'parents' of the affected module - this is known as version proliferation); and iii) the intertwined model - components and versions are selected in alternating order (so preventing version proliferation).

![Figure 5.10 - 'Product First (a), Version First (b) and Intertwined Models (c)'](image-url)

---

The MATrA Configuration Model

The evolution history (audit trail) of versioned artifacts can be represented in terms of a graph (typically a sequence) comprising a set of versions connected by relationships of a single type. These are termed successor relationships such that \( v_2 \text{ succeeds } v_1 \) means that \( v_2 \) has been derived from \( v_1 \).

Different product versions can be produced by combining different constituent artifacts and different versions of constituents into what is termed a configuration. The degree of flexibility permitted in producing such combinations depends on the version model underlying the configuration process. Conradi & Westfechtel (1998) distinguish three such models (figure 5.10) classified by their selection order, specifically: i) product first - the product structure is selected first, followed by the versions of each module (such that each configuration has the same structure); ii) version first - the product version is selected first which uniquely determines module versions (allowing different versions to be structured differently, though in the event of change, new versions must be created for all 'parents' of the affected module - this is known as version proliferation); and iii) the intertwined model - components and versions are selected in alternating order (so preventing version proliferation).

![Figure 5.10 - 'Product First (a), Version First (b) and Intertwined Models (c)'](image-url)

---

248
This section merely touches on a few core issues regarding configuration management and then only briefly to provide background for the MCM. The notion of baselines, releases and promotions, as well as alternative forms of delta and version model are beyond the scope of this thesis. Interested readers are instead referred to Conradi & Westfechtel (1998) or else, the IEEE guidelines on configuration management (IEEE STD 1042, 1987). With the concepts outlined above in mind, we now introduce the MATrA Configuration Model.

5.5.3 Across Revisions and Variants in MATrA: A Configuration Model for Tracing Evolutionary Development

As previously indicated, the MATrA product space consists of two parts; a tool independent view (Product Data Synthesis) and the tool dependent view (Workspace). For configuration management it is necessary to introduce constructs to permit handling of revisions and variants for both views.

5.5.3.1 Concepts

MCM encapsulates the notion of product and version space (from 5.5.2.1) by integrating features necessary to represent the latter in the context of existing features (supporting the former) from the traceability framework introduced in subsection 3.3 - specifically, BuildElement and BuildAssociation (PDS), and AerospaceEngineeringObject and AerospaceEngineeringAssociation (Workspace).

The 'version-oriented' features to be introduced in this subsection are drawn from general configuration management theory and are motivated by a number of requirements. For the PDS, these requirements are support for:- r1) identifying and structuring revisions and variants (for which we adopt an intertwined modelling approach); r2) impact change analysis; r3) revision audit trails; r4) definition of configurations; and r5) configuration deltas (i.e., differences between configurations). Similarly, for the Workspace our requirements are support for:- r1) revision audit trails; and r2) meta-model deltas (i.e., differences between meta-models). We make the observation in stating these requirements that the Workspace presents only partial models of the emerging system, whereas the PDS provides a complete model and hence specifies all constituent elements (and their relationships) at given points in time; impact analysis and hence change is therefore driven from the PDS. This explains why there are fewer requirements for the Workspace.

5.5.3.2 MATrA Configuration Model Definitions

We now introduce the class diagram for our MATrA Configuration Model (5.5.3.2.1), together with OCL constraints and rules over elements of the model (5.5.3.2.2) and O-Telos implementation of its base classes (5.5.3.2.3).

5.5.3.2.1 MATrA Configuration Model

Figure 5.11 shows elements and relationships of the MATrA Configuration Model. Essentially, it divides into two zones (denoted by 'swimlanes'): constructs (and their subtypes) for representation and version management of the Product Data Synthesis - namely build object (BuildObject), build
dependency (BuildDependency), Configuration and configuration delta (ConfigurationDelta) - and those for representation and version management of the traceability Workspace - namely aerospace engineering object (AerospaceEngineeringObject), aerospace engineering association (AerospaceEngineeringAssociation) and aerospace engineering delta (AerospaceEngineeringDelta). The zones are connected by subtypes of aerospace link entity (AerospaceLinkEntity), BEelementAEO and BEmodelAEO (not shown) which are as described in subsection 3.3.6.4.4.

![Diagram of MATrA Configuration Model](image)

**Figure 5.11 - ‘MATrA Configuration Model (MCM)’**

### i. Version Management for the Product Data Synthesis

From figure 5.11, it can be seen that the core element in the ‘PDS zone’ is BuildObject, an abstract class subsuming Version (with property version_name of type String) and build element (BuildElement) subtypes. It was introduced as the result of a modelling decision to represent all PDS relationships - build association (BuildAssociation), configuration association (ConfigurationAssociation), Impacts, Succeeds and configuration build association (ConfigurationBuildAssociation) - using the abstract BuildDependency class. BuildDependency greatly simplifies the MCM by generalising the behaviour of each of these subtypes through common associations for source (bd_source) and target (bd_target) elements. However, since bd_source and bd_target can either be of type Version or of type BuildElement depending on the BuildDependency type to which they apply - for instance, the source of a ConfigurationAssociation must be of type BuildElement and the target of type Version - so BuildObject generalises the behaviour of BuildElement and Version. Appropriate invariants (see subsection 5.5.3.2.2(i.1)) over BuildDependency subtypes ensure type restrictions on bd_source and bd_target are maintained.
We now consider the support provided by specific elements of the MATrA Configuration Model in meeting version management requirements for the PDS introduced in subsection 5.5.3.1.

**R1 Elements supporting an intertwined approach to the identification and structuring of revisions and variants**

To support the identification of revisions and variants and to enable them to be structured using an intertwined approach, MCM uses the above-mentioned Version and BuildElement classes along with means for their association - specifically subtypes of BuildAssociation and ConfigurationAssociation.

BuildAssociation differs from the original definition in subsection 3.3.5 inasmuch as subtype sources are now mostly of type Version, while targets are strictly of type BuildElement (as before). This is because BuildAssociation is being used in conjunction with HasRevision and HasVariant (with property variant) - both subtypes of the abstract ConfigurationAssociation class - to express the underlying intertwined model structure (essentially alternating version and product elements). The exception is HasSpecification whose source - the Property BuildElement - is considered a primitive and therefore not versioned. Recall, source and target types of ConfigurationAssociation are confined to BuildElement and Version respectively.

**R2 Elements supporting impact change analysis**

Dependencies between build elements are expressed as Impacts build associations. This was influenced by the Dependency (Riddle & Saeed, 1998) and Impact (Saeed et al., 1995) Structures referred to in subsection 2.2.1.3, both of which capture interactions among system elements. While strictly beyond the scope of version management, inclusion of such a feature within MATrA is motivated by a need to support impact analysis as an integral part of the evolutionary process for revisions and variants.

**R3 Elements supporting revision audit trails**

Once a change has been initiated and the new element inserted, an instance of the Succeeds build association can be used to relate the revised instance to its predecessor, thereby capturing (for audit trail purposes) the temporal ordering (as described in 5.5.2.1).

**R4 Elements supporting definition of configurations**

Definition of configurations is supported by the Configuration class, a (manually selected) collection of build objects. A Configuration has at its root a single Version element (indicated by the root_node rolename in figure 5.11). From the root_node it is necessary to project the actual product structure implied by BuildObject selections. This is the motivation for the (abstract) ConfigurationBuildAssociation class whose subtypes (not shown), including has-configuration-submodule (HasConfigurationSubmodule) and consumes-configuration XIO (ConsumesConfigurationXIO), etc., correspond to conventional build associations. These are used to express transitive associations between versioned product elements; for example, if ModuleA (MA) version_1 (MAv_1) has-submodule ModuleB (MB) and MB has-version version_2.

---

12 Note, we have not specified this as an OCL invariant since the revised HasSpecification class (as with all specialisations of BuildAssociation) is not shown in figure 5.11.
13 Note that Impacts associations are inherently pessimistic in that they simply indicate a degree of dependency among related elements; practitioners must still investigate tolerance to change based on engineering judgement.
such that a Configuration CI includes MAv; and MBv;, then a HasConfigurationSubmodule association between these two elements is derived for CI using the rule stated in 5.5.3.2.2(i.2). A Configuration may also incorporate conventional BuildElement and BuildAssociation subtypes (the latter derivable through the rule in 5.5.3.2.2(i.3)), typically instances of Property and Specification (as primitives) and accordingly, HasProperty and HasSpecification. We note also that instances of the Condition build element (mainly ‘simple’, non-versioned states and events), along with build associations used in conjunction can also be included as part of a Configuration (a point illustrated by our worked example in 5.5.3.3). Again, invariants (not stated) may be used to enforce restrictions.

**R5 Elements for creating configuration deltas**

A configuration delta (ConfigurationDelta) represents the symmetric difference between build objects in two configurations (rolenames configuration_1 and configuration_2). Its content can also be derived using the appropriate rule from 5.5.3.2.2 (ii.1).

**R1 Elements supporting revision audit trails**

The version history of a TraceabilityStructure is recorded using the Succeeds-AEO (SucceedsAEO) association which again orders successive instances as per 5.5.2.1 (and hence serves the same purpose as Succeeds for PDS elements). We also introduce the Added To (AddedTo) and Removes associations; respectively, this pairing relate elements of a new model to a predecessor in which those elements were not present, and relate the new model to elements of its predecessor not present in the revision. Both AddedTo and Removes may be derived using rules in 5.5.3.2.2 (specifically, iii.2 and iii.3).

**R2 Elements supporting meta-model deltas**

The revision audit trail associations in r1 (above), further enable derivation of elements of the AerospaceEngineeringDelta class (populated using the rule in 5.5.3.2.2(ii.3)) which records the symmetric difference, in terms of structure elements (rolename delta_aeo), for two Workspace meta-models (rolenames model_1 and model_2).

---

14 To simplify figure 5.11, the AEE subtype has been omitted. The constraints expressed in subsection 5.5.3.2.2(iii.4) restrict from_entity and to_entity ends of the AerospaceEngineeringAssociation class.
15 Note, a corresponding addition to the «Artifact» meta-class (not shown) from 3.3.3 is required.
5.5.3.2.2 OCL Constraints

As indicated above, we state a number of constraints and rules over the MCM. These are as follows:

I. Build Dependency

1. The following constraints ensure correct instantiation of source and/or target elements for BuildDependency subtypes.

   - The target of a BuildAssociation is of type BuildElement.

   BuildAssociation Invariant
   self.allInstances->forall(b | b.bd_target.oclIsKindOf(BuildElement))

   - The source of a ConfigurationAssociation is of type BuildElement and the target of type Version.

   ConfigurationAssociation Invariant
   self.allInstances->forall(c | c.bd_source.oclIsKindOf(BuildElement) and c.bd_target.oclType = Version)

   - The target of a ConfigurationBuildAssociation is of type Version.

   ConfigurationBuildAssociation Invariant
   self.allInstances->forall(b | b.bd_target.oclType = Version)

2. This rule derives population of the ConfigurationBuildAssociation class.

   Configuration
   self.allInstances->forall(c | self.build_object->forall(b1, b2 | self.build_object.buildDependency->select(self.build_object.buildDependency.oclIsKindOf(BuildAssociation))->forall(b | self.config_build_association->exists(a | c.build_object->includes(b1) and c.build_object->includes(b2) and b.bd_source->includes(b1) and (b.bd_target.hasRevision.bd_target->includes(b2) or b.bd_target.hasVariant.bd_target->includes(b2)) and (b2.bd_source.isEmpty and b2.bd_target.isEmpty))) implies c.config_build_association->includes(a) and a.bd_source->includes(b1) and a.bd_target->includes(b2))

   Note: an additional rule (not shown) ensures instances of ConfigurationBuildAssociation (variable 'a' in the above) correspond to instances of BuildAssociation (variable 'b' in the above) from which they are derived; e.g., b.oclType = HasSubmodule implies a.oclType = HasConfigurationSubmodule.

3. This rule derives population of the BuildAssociation class.

   Configuration
   self.allInstances->forall(c | self.build_object->forall(b1, b2 | self.build_object.buildDependency->select(self.build_object.buildDependency.oclIsKindOf(BuildAssociation))->forall(b | c.build_object->includes(b1) and c.build_object->includes(b2) and b.bd_source->includes(b1) and b.bd_target->includes(b2) and b2.bd_hasRevision.isEmpty and b2.bd_hasVariant.isEmpty))) implies c.build_association->includes(b)

II. Configuration Delta and Aerospace Engineering Delta

1. The following rule derives population of ConfigurationDelta elements.
ConfigurationDelta invariant
self.allInstances->forall(d)
self.configuration_1->forall(c_1 | self.configuration_2->forall(c_2 |
d.configuration_1->includes(c_1) and d.configuration_2->includes(c_2) )) implies
d.build_object->includesAll((c_1.build_object - c_2.buildObject)->union(c_2.buildObject - c_1.buildObject))

2. Constraint to ensure that AEOs specified as subjects of an AerospaceEngineeringDelta (over rolenames model_1 and model_2) are forms of TraceabilityStructure.

AerospaceEngineeringDelta invariant
self.allInstances->forall(a | a.model_1.oclType.oclType.oclIsKindOf(TraceabilityStructure) and
a.model_2.oclType.oclType.oclIsKindOf(TraceabilityStructure))

3. Rule to derive population of AerospaceEngineeringDelta elements.

AerospaceEngineeringDelta invariant
self.allInstances->forall(d | self.model_1->forall(m_1 | self.model_2->forall(m_2 | self.model_2.removes.to_entity->union(self.model_1.addedTo.from_entity)->forall(c |
s.model_1->includes(m_1) and
s.model_2->includes(m_2) and
d.model_1->includes(c) and
d.model_2->includes(c)))
implies
d.delta_aeo->includes(c)

Aerospace Engineering Association
1. Rule to populate composed_of associations of the AerospaceEngineeringObject class for use in deriving AddedTo and Removes associations between AEOs.

AerospaceEngineeringObject
self.allInstances->forall(a_1, a_2 | self.associationEnds->forall(e | a_1.associationEnds->includes(e) and a_1.e->includes(a_2))
implies
a_1.composed_of->includes(a_2)

2. Rule to derive AddedTo associations.

AerospaceEngineeringObject
self.allInstances->select(self.allInstances.oclType.oclType.oclIsKindOf(TraceabilityStructure))->forall(m_1 | self.succeedsAEO.to_entity->exists(m_2 | self.addedTo->exists(d |
self.composed_of->forall(c_1 | self.succeedsAEO.to_entity.composed_of->not exists(c_2 | self.composed_of->forall(c | self.succeedsAEO.to_entity.composed_of->not exists(c | self.composed_of.attributes->forall(a |
self.composed_of.attributes->forall(a' |
self.composed_of.associationEnds->forall(e | m_1.succeedsAEO.to_entity->includes(m_2) and
m_1.composed_of->includes(c_1) and m_2.composed_of->includes(c_2) and

ii. Aerospace Engineering Association
Structuring Safety Assessment and Product Management Artifacts

c1.oclType = c2.oclType and
c1.attributes->includes(a) and c2.attributes->includesAll(c1.attributes) and
c1.a = c2.a and
c1.associationEnds->includes(e) and c2.associationEnds->includesAll(c1.associationEnds) and
c1.e->includes(c') and c2.e->includes(c") and c'.oclType = c".oclType and
c'.attributes->includes(a') and c".attributes->includesAll(c'.attributes) and
c'.a' = c".a'))))))))))
implies
d.from_entity->includes(c1) and d.to_entity->includes(m2)

3. Rule to derive Removes associations.

AerospaceEngineeringObject
self.allInstances->select(self.allInstances.oclType.oclType.oclIsKindOf(TraceabilityStructure))->forall(m1 l
self.succeedsAEO.to_entity->exists(m2 l
self.removes->exists(r l
self.composed_of->not exists(c1 l
self.succeedsAEO.to_entity.composed_of->forall (c2 l
self.composed_of->not exists(c' l
self.succeedsAEO.to_entity.composed_of->forall (c" l
self.succeedsAEO.to_entity.composed_of.of_attributes->forall(a l
self.succeedsAEO.to_entity.composed_of.of_attributes->forall(a' l
self.succeedsAEO.to_entity.composed_of.of associationEnds->forall(e l
m1.succeedsAEO.to_entity->includes(m2) and
m1.composed_of->includes(c1) and m2.composed_of->includes(c2) and
c1.oclType = c2.oclType and
c2.attributes->includes(a) and c1.attributes->includesAll(c2.attributes) and
c1.a = c2.a and
c2.associationEnds->includes(e) and c1.associationEnds->includesAll(c2.associationEnds) and
c2.e->includes(c") and c1.e->includes(c') and c'.oclType = c".oclType and
c".attributes->includes(a') and c'.attributes->includesAll(c".attributes) and
c".a' = c'.a'))))))))))
implies
r.from_entity->includes(m1) and r.to_entity->includes(c2)

We note that rules to populate AddedTo or Removes associations are based on the tenet that ‘equivalent’
elements appearing in separate models are in fact represented by different objects. This results from the
assumed mapping process between CASE tools and the MATrA Workspace. Thus we cannot simply
consider whether two objects are the same, but instead must determine their equivalence - i.e., whether
they are of the same type, have the same attributes with the same values and are connected to the same
set of ‘equivalent’ elements. We return to this issue in the section on future work (7.4.10).

4. The following ensure correct instantiation of source and target elements for the SucceedsAEO,
   AddedTo and Removes AerospaceEngineeringAssociation subtypes.

SucceedsAEO Invariant
self.allInstances->forall(s l
s.from_entity.oclType.oclType.oclIsKindOf(TraceabilityStructure) and
s.to_entity.oclType.oclType.oclIsKindOf(TraceabilityStructure))

AddedTo Invariant
self.allInstances->forall(a l
a.from_entity.oclType.oclType = StructureElement and
The MATrA Configuration Model

a.to_entity.oclType.oclType.oclIsKindOf(TraceabilityStructure))

Removes invariant
  self.allInstances->forall(r1
    r. from_entity.oclType.oclType.oclIsKindOf(TraceabilityStructure) and
    r.to_entity.oclType.oclType = StructureElement)

We reiterate that the above are merely a selection of constraints and rules expressing possible behaviour of the MCM. Further areas to consider include invariants relating to an element’s status\textsuperscript{16}, of which the following provides a flavour. Specifically, it imposes a restriction that the source of a Succeeds association must have an "active" status, whilst the status of its target must be "abandoned"\textsuperscript{17}.

Succeeds Invariant
  self.allInstances->forall(s | s.bd_source.status = "active" and s.bd_target.status = "abandoned")

5.5.3.2.3 O-Telos Implementation of MATrA Configuration Model Base Classes

The following O-Telos code implements MCM base classes as per figure 5.11, along with extensions to the Framework Model in Chapter Three (see Appendix B, Part 1 for the 'equivalent' UML extensions).

Aerospace Management Entity - MCM
dsubtypes
  AerospaceConfigurationEntity in SimpleClass isA AerospaceManagementEntity
  with constraint
    abstract_ACE: $ forall t/Token, s/SimpleClass
      (t in s) ==> not (t in AerospaceConfigurationEntity)$ end

Aerospace Configuration Entity subtypes
  AerospaceEngineeringDelta in SimpleClass isA AerospaceConfigurationEntity
  with attribute
    model_1 : AerospaceEngineeringObject;
    model_2 : AerospaceEngineeringObject;
    delta_aeo : AerospaceEngineeringObject
  end
  ConfigurationDelta in SimpleClass isA AerospaceConfigurationEntity
  with attribute
    configuration_1 : Configuration;
    configuration_2 : Configuration;
    build_object : BuildObject
  end
  Configuration in SimpleClass isA AerospaceConfigurationEntity
  with attribute
    root_node : Version;
    build_association : BuildAssociation;
    config_build_association : ConfigurationBuildAssociation;
    build_object : BuildObject
  end

Version in SimpleClass isA AerospaceConfigurationEntity, BuildObject
  with attribute
    version_name : String
  end

Build Object (abstract supertype of Version and BuildElement)
  BuildObject in SimpleClass
  with constraint
    abstract_BO: $ forall t/Token, s/SimpleClass
      (t in s) ==> not (t in BuildObject)$ end

Aerospace Build Entity - MCM subtypes
  BuildElement in SimpleClass isA AerospaceBuildEntity, BuildObject
  with constraint
    abstract_BE: $ forall t/Token, s/SimpleClass
      (t in s) ==> not (t in BuildElement)$ end
  BuildDependency in SimpleClass isA AerospaceBuildEntity
  with attribute
    bd_source : BuildObject;
    bd_target : BuildObject
    constraint
      abstract_BD: $ forall t/Token, s/SimpleClass
        (t in s) ==> not (t in

\textsuperscript{16} Recall status is a property of AerospaceTraceabilityEntity and so part of every MATrA framework class (see Appendix B, Part 1).

\textsuperscript{17} As we do not consider option space exploration and the different status' an element may have, all revisions are assumed to be 'active' and their predecessor, 'abandoned'.
5.5.3.3 MCM Worked Example: Tracing Revisions for the Airbus A320-100/A320-200 Flight Control System

To illustrate key concepts of the MCM, we now present a detailed worked example. It should be noted that whereas the meta-models introduced previously represented established notations (or domain-specific variants thereof), this section has proposed a novel structure which as such, has no recognised syntax. Therefore to describe elements of the model in a uniform and "canonical" formalism, a graph-based notation similar to DRCS (subsection 3.2.3) is adopted.
The example demonstrates support provided by the MATrA Configuration Model for tracing revisions in the context of Flight Control Systems for the Airbus A320-100 and A320-200 aircraft. It considers the interplay between product space and version space, as well as management of Aerospace Engineering Objects.

### 5.5.3.3.1 Background

Designed to carry 150 passengers over short-to-medium routes, A320 was Airbus' first narrowbody jetliner. However, only twenty-one of the initial -100 (pronounced 'dash 100') were built, before production switched to the -200 revision. Effects of this evolution on the Flight Control System serve as the focus of our example.

Flight Control Systems provide functions for take-off and landing, as well as in-flight manoeuvres such as banking and climb. These are commanded through a number of primary and secondary control surfaces that dictate air-flow over the wings, body and tail-plane. Our example concerns the ‘slats’, secondary control surfaces preventing stall located on the wings leading edge. Readers are referred to Mair & Birdsall (1992) for more information on aircraft control surfaces.

A320 was the first aircraft to adopt a completely ‘fly-by-wire’ approach. On conventional aircraft, mechanical linkages transmit pilot orders to actuators, which in turn move the control surfaces. Though power applied to each surface is supplied from hydro-mechanical servo-amplifiers, their precise movements are co-ordinated solely by the flight crew. In contrast, A320 pilots provide instructions to flight control computers which are interpreted and used to generate actuator commands. This enables control law constraints to be applied to the aircraft to maintain ‘flight envelope protection’, thereby preventing certain pilot commanded actions - e.g. slat retraction at low speed - which may induce hazardous conditions (for instance, engine stall).

The FCS architecture comprises three computer types: ELlevator and Aileron Computers (ELACs), Spoiler and Elevator Computers (SECs) and Slat and Flap Control Computers (SFCCs). Redundancy, is essential given reliance on the system and manoeuvring difficulties in the event of total loss. In this scenario, we concentrate on the dual SFCCs; both divide into two channels Slat and Flap, with each channel further sub-divided into two lanes. Readers are referred to Favre (1994) or to Pearson & Rowlands (1996) for more information on A320 Flight Control Systems.

### 5.5.3.3.2 Scenario

This scenario concerns revisions to the A320-100 towards production of A320-200 - an extended range and increased take-off weight derivative. The MATrA Configuration Model is used to support modification of -100 SFCCs to accommodate (hypothetical) parameter variances stemming from the revised range and take-off weight requirements. In particular, we examine a function that prevents slat retraction by overriding pilot commands under specified conditions - the aim being to avert hazardous stall conditions. Our scenario comprises a number of 'episodes' which progress the example from
definition of the A320-100 base aircraft, to creation of change deltas between this and the A320-200.

The following serves as a route-map identifying the purpose of each episode, along with their relevance to the MCM requirements stated in subsection 5.5.3.1:

1. **Episode 1** (partially) describes the PDS (using an intertwined representation) for the A320-100 base aircraft, including definition of the FCS and its subsystems. We also show an audit trail example for successive instances of one element; exhibits PDS requirements R1 and R3 (and provides context for Episodes 4 and 5).

2. **Episode 2** (partially) describes the Workspace for the A320-100 using a Statechart meta-model fragment for a key function of the SFCCs (together with Aerospace Link Entities associating appropriate parts of the meta-model with corresponding PDS elements); provides context for Episodes 6 and 8.

3. **Episode 3** demonstrates a partial configuration for the A320-100; exhibits PDS requirement R4.

4. **Episode 4** demonstrates a partial impact change analysis over properties of the A320-100; exhibits PDS requirement R2.

5. **Episode 5** is included for completeness and describes updating of the PDS towards definition of the A320-200 derivative aircraft in response to findings from the impact change analysis; also provides a context for Episode 7.

6. **Episode 6** updates the Statechart meta-model from Episode 2 for the A320-200 revision and inserts appropriate audit trail associations; exhibits Workspace requirement R1.

7. **Episode 7** demonstrates a partial configuration delta between A320-100 and -200 elements; exhibits PDS requirement R5.

8. **Episode 8** demonstrates an aerospace engineering delta for the A320-100 and -200 meta-model fragments featured in Episode 6; exhibits Workspace requirement R2.

---

**Episode 1 - Describing the A320-100 Base Aircraft**

We begin by populating the MCM with elements describing the A320-100. Firstly, a 'container' class for the A320 project (A320 Project) as a whole is created\(^{16}\), together with a Product Data Synthesis (A320 PDS) for this particular project. The PDS includes our base aircraft A320-100 (of type Version), a revision of the generic A320 (BuildElement of type Module) for which a number of properties (BuildElement(s) of type Property) are defined, as shown in figure 5.12. Specifically, Gross Wing Area, Landing Distance, Wing Loading, Fuel Capacity, Maximum Takeoff Weight, Maximum Lift Coefficient, Payload, Range, Touch Down Speed, Approach Speed and Stalling Speed\(^{19}\).

\(^{16}\) This will contain all A320 revisions, together with variants including A321 and A319 (as indicated by the A321-100 'stub'). Variants are not considered further in this example as their application in terms of Configurations and Configuration Deltas, etc. is the same as that for revisions (except of course that multiple variants can co-exist).\(^{19}\) Interested readers are referred to Mair & Birdsall (1992) for the precise meaning of these terms.
Figure 5.12 - ‘A320-100 Product Data Synthesis Fragment - Properties’

Figure 5.13 - ‘A320-100 Product Data Synthesis Fragment - Architecture’
At this point, our concern is purely to provide a context for the scenario and in particular, a set of properties with which to demonstrate use of Impacts associations in assessing change (Episode 4), and a set of architectural elements (to be introduced below) for illustrating modifications to the base aircraft as a result of this analysis (Episode 5). However, to give an early indication of how an audit trail of multiple revisions is represented using the MATrA Configuration Model, Maximum Takeoff Weight is shown as an emergent property captured through a succession of value specifications.

Figure 5.13 extends our base aircraft description. It shows a (partially versioned\(^{20}\)) vertical path through the FCS architecture, including twin fault tolerant SEC (SEC1 and SEC2), ELAC (ELAC1 and ELAC2) and SFCC (SFCC1 and SFCC2) submodules. SFCC1 is further decomposed to show the Slat and Flap channels (Slat Channel and Flap Channel) for this particular computer. The former is likewise decomposed to lanes 1 and 2 (SCLane1 and SCLane2); for lane 1, we show the Inhibit Slat Retraction function and its two subfunctions ISR-Low Speed and ISR-High Angle of Attack. In addition, conditions (states) that this function may assume, namely disabled (Disabled), not engaged (Not-Engaged), speed baulk (Speed-Baulk) and alpha lock (Alpha-Lock) are also identified. Note the alternating Version/BuildElement (featuring Module, Function and Condition subtypes) pattern that characterises an intertwined approach.

\[\text{Figure 5.13} - \text{Decomposition of Inhibit Slat Retraction at High Angles of Attack (A320-100)}\]

Figure 5.14 further refines this decomposition through the introduction of sub-conditions for Alpha-Lock - namely Alpha-Lock Wait (Alpha-Lock Wait), Engaged Alpha-Lock (Engaged Alpha-Lock) and Release Alpha-Lock (Release Alpha-Lock). The ISR-High Angle of Attack sub-function consumes a flow of type Corrected Angle of Attack (CAoA) produced by external sensor devices (not shown). A transition from Engaged Alpha-Lock to Release Alpha-Lock occurs when the CAoA data flow value is less than 7.1 degrees.

\(^{20}\) Again, multiple revision elements are not shown to reduce complexity of the diagram.

261
(CAoA < 7.1 degrees); this event is 'carried' in the data flow\textsuperscript{21}.

Similarly, figure 5.15 introduces sub-conditions for Speed-Baulk - namely Speed Baulk Wait (Speed-Baulk Wait), Engaged Speed Baulk (Engaged Speed-Baulk) and Release Speed Baulk (Release Speed-Baulk). The ISR-Low Speed sub-function consumes a flow of type Computed Air Speed (CAS), again produced by external sensors. A transition from Engaged Speed-Baulk to Release Speed-Baulk occurs when CAS exceeds one hundred and fifty four knots (CAS > 154 knots).

Figure 5.15 - 'Decomposition of Inhibit Slat Retraction at Low Speed (A320-100)'

O-Telos objects capturing (partial) information from figures 5.12 through 5.15 are as follows:-

\textbf{Instantiation of elements in figure 5.12}

\begin{verbatim}
A320 in AerospaceEngineeringProject, Token with
   project_title : "A320 Project"
   has_pds       : A320PDS
end

A320PDS in ProductDataSynthesis, Token with
   build_entity
      buildEntity1 : AirbusA320;
      buildEntity2 : GrossWingArea;
      buildEntity3 : LandingDistance;
      buildEntity4 : WingLoading;
      buildEntity5 : FuelCapacity;
      buildEntity6 :
end

MaximumLiftCoefficient;
   buildEntity7 : Payload;
   buildEntity8 : Range;
   buildEntity9 : TouchDownSpeed;
   buildEntity10 : ApproachSpeed;
   buildEntity11 : StallingSpeed;
   buildEntity12 : MTOW

AirbusA320 in Module, Token with
   moduleName : "A320"
   bd_source
end

A320HasVer100 in HasRevision, Token with
   bd_source
end
\end{verbatim}

\textsuperscript{21} Note the Condition build element can be either versioned or treated as a primitive as needs dictate. As a rule of thumb, conditions representing hierarchical states composed of sub-states are versioned, (e.g., Alpha Lock), whereas simple states and events (as with specifications) are not (e.g., CAoA < 7.1 degrees) - see figures 5.14 and 5.15.
Structuring Safety Assessment and Product Management Artifacts

bdSource : AirbusA320
bd_target : Dash100
end

Dash100 in Version, Token with
version_name : "A320-100"
end

Dash100HasPropertyGrossWingArea in
HasProperty, Token with
bd_source
  bdSource : Dash100
bd_target
  bdTarget : GrossWingArea
end

GrossWingArea in Property, Token with
  propertyName : "Gross Wing Area"
end

Dash100HasPropertyMTOW in HasProperty,
Token with
bd_source
  bdSource : Dash100
bd_target
  bdTarget : MTOW
end

MTOW in Property, Token with
  propertyName : "Maximum Takeoff Weight"
end

-- see App. B, Pt. 3 for other properties

MTOWHasSpecificationSpec1 in
HasSpecification, Token with
bd_source
  bdSource : MTOW
bd_target
  bdTarget : MTOWSpec1
end

MTOWSpec1 in Specification, Token with
  valueSpecification : "75 000 kg"
end

MTOWHasSpecificationSpec2 in
HasSpecification, Token with
bd_source
  bdSource : MTOW
bd_target
  bdTarget : MTOWSpec2
end

MTOWSpec2 in Specification, Token with
  valueSpecification : "75 500 kg"
end

MTOWHasSpecificationSpec3 in
HasSpecification, Token with
bd_source
  bdSource : MTOW
bd_target
  bdTarget : MTOWSpec3
end

MTOWSpec3 in Specification, Token with
  valueSpecification : "76 000 kg"
end

MTOWSpecSucceeds2 in Succeeds, Token with
bd_source
  bdSource : MTOWSpec3
bd_target
  bdTarget : MTOWSpec2
end

Instantiation of elements in figure 5.13

FlightControlSystem in Module, Token with
module_name
  moduleName : "FCS"
end

FCSHasVerDash100FCS in HasRevision, Token with
bd_source
  bdSource : FlightControlSystem
bd_target
  bdTarget : Dash100FCS
end

Dash100FCS in Version, Token with
version_name
  versionName : "-100 FCS"
end

Dash100FCSHasFTsubmoduleSEC1 in
HasFTSubmodule, Token with
bd_source
  bdSource : Dash100FCS
bd_target
  bdTarget : SEC1
end

SEC1 in Module, Token with
module_name
  moduleName : "SEC1"
end

-- see App. B, Pt. 3 for SEC2 objects

Dash100FCSHasFTsubmoduleELAC1 in
HasFTSubmodule, Token with
bd_source
  bdSource : Dash100FCS
bd_target
  bdTarget : ELAC1
end

ELAC1 in Module, Token with
module_name
  moduleName : "ELAC1"
end

-- see App. B, Pt. 3 for ELAC2 objects

Dash100FCSHasFTsubmoduleSFCC1 in
HasFTSubmodule, Token with
bd_source
  bdSource : Dash100FCS
bd_target
  bdTarget : SFCC1
end

SFCC1 in Module, Token with
module_name
  moduleName : "SFCC1"
end

-- see App. B, Pt. 3 for SFCC2 objects

SFCC1HasVerDash100SFCC1 in HasRevision,
Token with
bd_source
  bdSource : SFCC1
bd_target
  bdTarget : SFCC1
end

263
bdTarget : Dash100SFCC1
end
Dash100SFCC1 in Version, Token with versionName : "-100 SFCC1"
end

-- see App.B, Pt.3 for Flap Channel objects
Dash100SFCC1HasSubmoduleSlatChannel in HasSubmodule, Token with bdSource : Dash100SFCC1
bdTarget : SlatChannel
end
SlatChannel in Module, Token with moduleName : "Slat Channel"
end
SlatChannelHasVerSlatChanneli in HasRevision, Token with bdSource : SlatChannel
bdTarget : SlatChanneli
end
SlatChanneli in Version, Token with versionName : "Slat Channel-i"
end
SlatChanneliHasFTsubmoduleSCLane1 in HasFTSubmodule, Token with bdSource : SlatChannel
bdTarget : SlatChanneli
end
SlatChanneliLanel in Module, Token with moduleName : "SCLanel"
end
-- see App.B, Pt.3 for Slat Channel Lane 2 objects
SlatChanneliLanelHasVerSlatChanneliLaneli in HasRevision, Token with bdSource : SlatChanneli
bdTarget : SlatChanneliLaneli
end
SlatChanneliLaneli in Version, Token with versionName : "SCLanel-i"
end
SlatChanneliLaneliEncapsulatesInhibitSlatRetraction in Encapsulates, Token with bdSource : SlatChanneli
bdTarget : InhibitSlatRetraction
end
InhibitSlatRetraction in Function, Token with functionName : "Inhibit Slat Retraction"
end

InhibitSlatRetractionHasVerInhibitSlatRetractioni in HasRevision, Token with bdSource : InhibitSlatRetraction
bdTarget : InhibitSlatRetractioni
end
InhibitSlatRetractioni in Version, Token with versionName : "Inhibit Slat Retraction-i"
end

-- see App.B, Pt.3 for 'Disabled' and 'Not-Engaged' condition objects
InhibitSlatRetractioniHasConditionSpeedBaulk in HasCondition, Token with bdSource : InhibitSlatRetractioni
bdTarget : SpeedBaulk
end
SpeedBaulk in Condition, Token with conditionLabel : "Speed-Baulk"
end
InhibitSlatRetractioniHasConditionAlphaLock in HasCondition, Token with bdSource : InhibitSlatRetractioni
bdTarget : AlphaLock
end
AlphaLock in Condition, Token with conditionLabel : "Alpha-Lock"
end
InhibitSlatRetractioniHasSubfunctionISRLowSpeed in HasSubfunction, Token with bdSource : InhibitSlatRetractioni
bdTarget : ISRLowSpeed
end
ISRLowSpeed in Function, Token with functionName : "ISR-Low Speed"
end
InhibitSlatRetractioniHasSubfunctionISRHighAoA in HasSubfunction, Token with bdSource : InhibitSlatRetractioni
bdTarget : ISRHighAoA
end
ISRHighAoA in Function, Token with functionName : "ISR-High Angle of Attack"
end

Instantiation of elements in figure 5.14
Structuring Safety Assessment and Product Management Artifacts

The following episode introduces artifacts from the traceability Workspace of our featured project. Again, the intention is to provide further contextual information - this time in the form of Aerospace Engineering Objects - for 'versioning' in subsequent episodes.
Episode 2 - Modelling the Behaviour of A320-100 SFCCs Using Statecharts

Practitioners will typically model behaviour of Inhibit Slat Retraction as a Statechart (Harel, 1988). In a MATrA context, this implies a meta-model exists capable of receiving data mapped from an appropriate tool. Statecharts are not a feature of our work here as project NATURE (Pohl, 1996) produced a meta-model for the OMT dynamic model (an extension of Statecharts), and because a Statechart meta-model for UML also exists in the public domain. Instead, we use a simplified version of the latter (Appendix B, Part 4) - based on version 1.3 of the UML Specification (Rational Software Corporation, 1997a) - to demonstrate their representation in this example.

The Inhibit Slat Retraction function has been modelled with ten states (conditions) as previously described in figures 5.14 and 5.15; for reader orientation the complete Statechart is shown in figure 5.16. Essentially, we are interested in events triggering a transition from Engaged Alpha-Lock to Release-Alpha Lock and from Engaged Speed-Baulk to Release Speed-Baulk; recall from the PDS, these occur under conditions CAoA < 7.1 degrees and CAS > 154 knots respectively. However as we later demonstrate, both are required to change as an indirect result of increased range and take-off-weight requirements for the dash 200 revision. This is shown for the former by analysis of Impacts associations in Episode 4.

Mapping the Statechart in figure 5.16, from the tool in which it was developed into the MATrA Workspace establishes BElementAEO and BMmodelAEO associations between the PDS and corresponding objects of the StateMachine meta-model. We demonstrate this at the logical level (i.e., as it appears to MATrA users) in figure 5.17, with the Statechart on the left and Product Data Synthesis on the right22.

---

Figure 5.16 - 'Statechart Representation of Inhibit Slat Retraction'

22 To maintain readability, PDS revisions are suppressed in figure 5.17.
O-Telos code capturing information contained in figure 5.17 appears below. Specifically, we present instantiation of the partial StateMachine meta-model and Aerospace Link Entity (BEmodelAEO and BEElementAEO) classes.

**Definition of State Machine Elements**
(subset)

InhibitSlatRetractionStateMachine in StateMachine, Token with subject

- Subject: "Inhibit Slat Retraction"
- State : AlphaLockState

end

AlphaLockState in SubMachineState, Token with

- state_name
  - stateName "Alpha-Lock"
- sub_machine
  - subMachine : AlphaLockStateMachine

end

AlphaLockStateMachine in StateMachine, Token with

- _State1 : EngagedAlphaLockState;
- _State2 : ReleaseAlphaLockState
- transition
  - _Transition : EngagedToReleaseCAoA

end

EngagedAlphaLockState in State, Token with

- state_name
  - stateName "Engaged Alpha-Lock"
end

ReleaseAlphaLockState in State, Token with

- state_name
  - stateName "Release Alpha-Lock"
end

EngagedToReleaseCAoA in Transition, Token with

- transSource : EngagedAlphaLock
- trigger
  - trans_trigger :
    EngagedToReleaseTriggerCAoA

end

EngagedToReleaseTriggerCAoA in Event, Token with

- event-description
  - eventDescription : "CAoA < 7.1 degrees"

end

**Definition of Aerospace Link Entity Elements**
(subset)

InhibitSlatRetraction in BEmodelAEO, Token with

- build_element : InhibitSlatRetraction
- aerospace_engineering_object : InhibitSlatRetractionStateMachine

end

EngagedToRelease in BEElementAEO, Token with

- build_element : AlphaLock
- aerospace_engineering_object : EngagedAlphaLockState

end

EngagedToRelease in BEElementAEO, Token with

- build_element : AlphaLock
- aerospace_engineering_object : ReleaseAlphaLockState

end

EngagedToRelease in BEElementAEO, Token with

- build_element : AlphaLock
- aerospace_engineering_object : EngagedAlphaLockState

end

267
Next we describe a configuration constructed from versions of build elements featured in the models for Episode 1.

**Episode 3 - Creating a Configuration for A320-100**

To create a configuration (instantiating the Configuration class) for A320-100, it is necessary to identify constituent elements from the versioned PDS data-set. While we purposely limit inclusion of multiple version elements for presentation purposes, a configuration can still be created to provide a flavour of what this practice involves (figure 5.18). Note the example features a representative subset of elements from the structures in Episode 1. Note also the omission of ‘spine’ associations (rolename build_object), and that all configuration build associations are assumed to be populated by the rule in 5.5.3.2.2(i.2).
In the following, we provide O-Telos code capturing information contained in the above figure (5.18).
Episode 4 - Impact Analysis of A320-100

At this point we shift emphasis to the consideration of impacts dependencies between elements introduced in Episode 1. As indicated in subsection 5.5.3.2.1, this is necessary to assess the need for modifications across a populated Product Data Synthesis - either to facilitate evolutionary change, or to assist in the development of product variants.

The intention is for the dependencies themselves to be inserted as development proceeds and so this episode is purely concerned with their application. Specifically, how changes in Range and Maximum-Takeoff Weight impact on various other aircraft performance properties and ultimately on the triggering of transitions from Engaged Alpha-Lock to Release-Alpha Lock and from Engaged Speed-Baulk to Release Speed-Baulk.
Structuring Safety Assessment and Product Management Artifacts

Figure 5.19 - 'Impacts Associations Showing Dependency Propagation'

It can be seen from figure 5.19 that Range influences Fuel Capacity which in turn has a bearing on the Maximum Takeoff Weight; this affects Wing Loading and hence Stalling Speed such that the latter impacts on our condition triggering the transition from Engaged Speed-Baulk to Release Speed-Baulk. Recall that the featured approach is pessimistic in that it simply alerts engineers to the fact that one build element is in some way influenced by another\(^\text{23}\). And whilst the element in question may exhibit some degree of 'tolerance' to change, this must be affirmed by investigation. Here we assume a change to the condition in question is necessary and demonstrate how it is handled within MATRa in later episodes. A similar investigation would of course be necessary to establish whether the condition triggering transitions from Engaged Alpha-Lock to Release Alpha-Lock also needs to be changed.

We now present an O-Telos example of the Impacts association class from figure 5.19, with source Maximum Takeoff Weight and target Wing Loading:

```
MtowImpactsWL in Impacts, Token with
  bd_source bdSource : MTOW
  bd_target bdTarget : WingLoading
end
```

The remaining episodes in our scenario relate to structuring of changes arising from analysis of the existing A320-100 aircraft towards production of its successor, the A320-200.

\(^\text{23}\) Note: a rule could specified to determine transitive dependencies between Impacts associations in order to support analysis.
Episode 5 - Describing the A320-200 Derivative

Let us now assume development of the A320-100 is complete and that engineers are in the process of modifying the existing design in order to produce an extended range and increased takeoff weight derivative, the A320-200. In Episode 4 we described how tracing across Impacts associations within the Product Data Synthesis has revealed that range affects fuel-capacity, thereby increasing the weight of the aircraft and that this impacts transitively on its safe operational envelope. In particular, conditions triggering release of the Inhibit Slat Retraction sub-functions ISR-Low Speed (shown) and ISR-High Angle of Attack (not shown, but based on the same principles of investigation).

Therefore having identified the need for change and established new event conditions using various analyses (again not shown) - the CAS threshold is modified to > 171 knots from 154, while CAoA becomes < 7.6 degrees from 7.1- engineers are in a position to express this information within the Product Data Synthesis. Firstly, a new Version object for the A320-200 aircraft (A320-200) is introduced, followed by new events (build elements) for the CAS and CAoA parameter conditions (figure 5.20).

Here, the changes are relatively small. But while at a low level of decomposition, the fact an intertwined model underpins our approach means original elements representing behaviour, as well as the functional and physical architecture, are largely retained. We simply introduce appropriate revisions to InputOutput and Condition elements and create a new configuration. In contrast, with a version first approach, new versions of all ‘parent’ build elements (direct and transitive) would need to be created.

Figure 5.20 - ‘Revising the A320-100 PDS (Including Partial Configuration Selection)’
Episode 6 - Updating Models of Behaviour for A320-200

While developing the revised aircraft, engineers must also update the Workspace. Therefore this episode considers traceability between structures capturing information from the existing and revised Statechart models. At a coarse granularity, this manifests as SucceedsAEO associations between successive models and at a finer level, as Removes and AddedTo associations (as described in 5.5.3.2.1).

In the context of our featured example, the original conditions triggering transitions from Engaged
Alpha-Lock to Release Alpha-Lock and Engaged Speed-Baulk to Release Speed-Baulk have been removed (i.e., are not present) in the new model, while their replacements (which of course were not present in the original) have been added. This concept is represented (again) at the logical level in figure 5.21:-

O-Telos code implementing the associations featured in figure 5.21 appears below:-

```o-telos
RemovesCAoA in Removes, Token with
  from_entity
    fromEntity : InhibitSlatRetractionStateMachineNew
  to_entity
    toEntity : EngagedToReleaseTriggerCAoA
end

AddedToCAoA in AddedTo, Token with
  from_entity
    fromEntity : EngagedToReleaseTriggerCAoANew
  to_entity
    toEntity : InhibitSlatRetractionStateMachine
end

RemovesCAS in Removes, Token with
  from_entity
    fromEntity : InhibitSlatRetractionStateMachineNew
  to_entity
    toEntity : EngagedToReleaseTriggerCAS
end

AddedToCAS in AddedTo, Token with
  from_entity
    fromEntity : EngagedToReleaseTriggerCASNew
  to_entity
    toEntity : InhibitSlatRetractionStateMachine
end

ISRSMSucceedsAEO in SucceedsAEO, Token with
  from_entity
    fromEntity : InhibitSlatRetractionStateMachineNew
  to_entity
    toEntity : InhibitSlatRetractionStateMachine
end

We also instantiate the Argumentation Structure (Appendix B, Part 2) with elements representing rationale attached to the 'ISRSMSucceedsAEO' association (from figure 5.21). In O-Telos, this is expressed as follows:-

ISRSMRaisesIssue in RaisesIssue, Token with
  at_source
    atSource : ISRSMSucceedsAEO
  issue_target
    issueTarget : ISRSMIssue
end

ISRSMIssue in Issue, Token with
  issue_text
    issueText : "Why replace existing model?"
end

ISRSMAssertionResolvesIssue in ResolvesIssue, Token with
  assertion_source
    assertionSource: ISRSMAssertion
  issue_target
    issueTarget : ISRSMIssue
end

ISRSMAssertion in Assertion, Token with
  assertion_text
    assertionText : "Change to CAoA and CAS"
end
```
The final two episodes consider support for change deltas, both within the Product Data Synthesis (Episode 7) and the traceability Workspace (Episode 8).

**Episode 7 - Configuration Delta for A320-100/A320-200**

As indicated in subsection 5.5.2.1, a configuration delta provides engineers with a summary of the symmetric difference between two configurations. In Episode 3, we demonstrated a fragment of a configuration for the A320-100 aircraft; further configurations would be taken in developing its successor the A320-200. The O-Telos example below provides a partial configuration delta (instantiating ConfigurationDelta) based on the configuration from Episode 3 and an assumed configuration (based on hypothetical elements, and those selected in Episode 5) for the A320-200. Recall that derived population of ConfigurationDelta is achieved by appropriate implementation of the rule in 5.5.3.2.2(ii.1).

```
A32010OA320200ConfigurationDelta in ConfigurationDelta, Token with
collection_1 : A320100SampleConfiguration
collection_2 : A320200SampleConfiguration
build_object
  buildObject1 : Dash100;
  buildObject2 : Dash200;
  buildObject3 : CAS154KtsEvent;
  buildObject4 : CAoA7Pt1Event;
  buildObject5 : CAS171KtsEvent;
  buildObject6 : CAoA7Pt6Event;
  buildObject7 : CAoAI;
  buildObject8 : CAoAII;
  buildObject9 : CASI;
  buildObject10 : CASII
end
```

**Episode 8 - Aerospace Engineering Delta for Behavioural Models**

Finally, we consider an aerospace engineering delta (instantiating AerospaceEngineeringDelta) expressing the symmetric difference between elements of our featured Statechart models derived (through appropriate implementation of the rule in 5.5.3.2.2(ii.3)) from the AddedTo and Removes associations in Episode 6. Again, only the O-Telos representation is provided.

```
ISRStateMachineAEODelta in AerospaceEngineeringDelta, Token with
model_1 : InhibitSlatRetractionStateMachine
model_2 : InhibitSlatRetractionStateMachineNew
delta_aeo
  deltaAE01 : EngagedToReleaseTriggerCAoA;
  deltaAE02 : EngagedToReleaseTriggerCAS;
  deltaAE03 : EngagedToReleaseTriggerCAoANew;
  deltaAE04 : EngagedToReleaseTriggerCASNew
end
```

**5.5.4 Summary**

Traceability of versions (subsuming revisions and variants) is essential to the aerospace industry as it supports continued evolution of existing aircraft, allowing manufacturers to keep pace with new technologies and changing markets.
This section proposed a novel structure for managing versions, and in particular revisions to both the Product Data Synthesis and traceability Workspace. The model encapsulates key concepts from configuration management literature, notably the intertwined versioning approach, as well as support for structuring configurations and change deltas.

Aspects of the model were demonstrated using a lengthy worked example, while a full evaluation again appears in Chapter Seven.
5.6 Chapter Summary

This chapter has presented novel meta-models (traceability structures) for well-defined and flexible notations used in safety assessment and product management throughout the aerospace industry; specifically, structures for Fault Tree Analysis and Failure Modes and Effects Analysis, together with a further structure based on the Programme Evaluation & Review Technique. Once more, we assume these are populated via the tool2matra transfer mechanism and where appropriate, verified for consistency against the Product Data Synthesis.

As in the previous chapter, we expressed key syntactic elements of each notation as a UML Class Diagram (using the System Engineering Notation Meta-model as a common basis), with well-formedness, usage and other constraints stated in OCL. Again the base classes were implemented in O-Telos (using ConceptBase) to provide a ‘flavour’ of automation, while consideration of the PERT structure also included a small worked example. Collectively these structures provide representative coverage of the assessment and product management Workspace viewpoints to complement the development perspective from Chapter Four.

The chapter concluded by demonstrating support for another aspect of product management, namely traceability of revisions and variants. In doing so we proposed the MATrA Configuration Model which is based on an established structuring technique from Configuration Management literature (the intertwined model), but which also incorporates means to represent change deltas and versioning related dependencies. Aspects of MCM were illustrated by a worked example.

In the following chapter we present two case studies based on material supplied by aerospace practitioners. The first of these exhibits the User Centred Requirements Structure (from Chapter Four) and the second, the Fault Tree and Failure Mode and Effects Analysis structures. Both case studies also feature the MATrA Natural Language Structure.
6 Tracing Development and Assessment Artifacts

6.1 Introduction

This chapter presents two case studies that demonstrate proof of concept for the User Centred Requirements Structure (from section 4.3) and the Fault Tree and Failure Modes and Effects Analysis structures (from sections 5.2 and 5.3).

Reader attention is drawn to the fact that featured artifacts are interleaved with O-Telos instantiations of the appropriate structures. Every effort has been made to maintain flow, with large amounts of code appearing in the appendices (C, Part 1 and D). However, traceability is essentially a practical topic and so representative examples are exhibited within the text. In view of these considerations, two paths are proposed which readers may follow depending on their interests, as shown in table 6.1.

<table>
<thead>
<tr>
<th>section</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2</td>
<td>Case Study I: A Hypothetical Mission Planning System for the Hawk 100 and 200 Series Aircraft</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Mission Planning System Overview</td>
</tr>
<tr>
<td>6.2.2</td>
<td>BASE Control Software Requirements and MArT Analysis</td>
</tr>
<tr>
<td>6.2.2.1</td>
<td>Use Case View</td>
</tr>
<tr>
<td>6.2.2.2</td>
<td>Interaction View</td>
</tr>
<tr>
<td>6.2.2.2.1</td>
<td>Erase Cartridge - Normal Path</td>
</tr>
<tr>
<td>6.2.2.2.2</td>
<td>Erase Cartridge - No Cartridge Present</td>
</tr>
<tr>
<td>6.2.2.2.3</td>
<td>Erase Cartridge - No Data on Cartridge</td>
</tr>
<tr>
<td>6.2.2.4</td>
<td>Erase Cartridge - Pilot Chooses Not to Erase Data</td>
</tr>
<tr>
<td>6.2.2.5</td>
<td>Retrieve from Cartridge - Normal Path (Timing Fragment)</td>
</tr>
<tr>
<td>6.2.2.6</td>
<td>Choose Mission and Aircraft (Event Group Fragment)</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Trace Relations</td>
</tr>
<tr>
<td>6.2.3.1</td>
<td>Instantiation of Trace Relations</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Summary</td>
</tr>
<tr>
<td>6.3</td>
<td>Case Study II: A Brake System Control Unit for a Wheel Braking System of a Hypothetical Aircraft</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Scope of Case Study</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Overview of S18 Wheel Braking System and Brake System Control Unit</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Preliminary System Safety Assessment - Brake System Control Unit</td>
</tr>
<tr>
<td>6.3.3.1</td>
<td>Background on BSCU Design</td>
</tr>
<tr>
<td>6.3.3.2</td>
<td>Fault Tree Analysis - Preliminary</td>
</tr>
<tr>
<td>6.3.3.2.1</td>
<td>Instantiation of Fault Tree Analysis Meta-model - Preliminary Fault Tree</td>
</tr>
<tr>
<td>6.3.4</td>
<td>System Safety Assessment - Brake System Control Unit</td>
</tr>
<tr>
<td>6.3.4.1</td>
<td>Background on BSCU Power Supply Design</td>
</tr>
<tr>
<td>6.3.4.2</td>
<td>Failure Modes and Effects Analysis</td>
</tr>
<tr>
<td>6.3.4.2.1</td>
<td>Functional Failure Modes and Effects Analysis</td>
</tr>
<tr>
<td>6.3.4.2.2</td>
<td>Instantiation of Functional Failure Modes and Effects Analysis Meta-model</td>
</tr>
<tr>
<td>6.3.4.2.3</td>
<td>Piece-Part Failure Modes and Effects Analysis</td>
</tr>
<tr>
<td>6.3.4.2.4</td>
<td>Instantiation of Piece-Part Failure Modes and Effects Analysis Meta-model</td>
</tr>
<tr>
<td>6.3.4.3</td>
<td>Fault Tree Analysis - Updated</td>
</tr>
<tr>
<td>6.3.4.3.1</td>
<td>Instantiation of Fault Tree Analysis Meta-model - Updated Fault Tree</td>
</tr>
</tbody>
</table>
6.3.5 Trace Relations

6.3.5.1 Instantiation of Trace Relations

6.3.6 Summary

6.4 Chapter Summary

Table 6.1 - ‘Chapter 6: Suggested Reading Paths’

The Application-Engineering path (i.e., omitting the shaded subsections in table 6.1) should be taken by those interested in traceability from a user perspective. Alternatively, readers concerned with tool development, and hence implementation of the featured structures may wish to concentrate on the Method-Engineering path (i.e., focusing on the O-Telos subsections shaded in table 6.1).

6.2 Case Study I: A Hypothetical Mission Planning System for the Hawk 100 and 200 Series Aircraft

In this section, we present a case study based on a Mission Planning System (MPS) for the Hawk 100 and 200 series of trainer/lead-in-fighter aircraft produced by BAE SYSTEMS (BAe, 1999). Its purpose is to demonstrate features of the User Centred Requirements Structure (subsection 4.3) and to show support for traceability across development artifacts within the (intra-micro) horizontal dimension.

6.2.1 Mission Planning System Overview

The Mission Planning System (MPS) supports mission planning and rehearsal and the transfer of data (stored on a Data Transfer Cartridge - DTC) to and from the aircraft environment. Its components include a Data Transfer Module (DTM) which provides a receptacle for, and facilitates supporting, the loading and unloading of data to and from the DTC; DTCs are inserted into the aircraft for the transfer and retrieval of data from aircraft avionics systems (BAe, ibid.)

MPS features four software modules, as shown in figure 6.1 - the BASE Control Software (BCS), Data Transfer Module Interface Software (DTMIS), Modular Mission Support Software (MMSS) and Theatre Creation Software (TCS). In this case study, we concentrate on the BASE Control Software.
6.2.2  BASE Control Software Requirements and MATrA Representation

The BASE Control Software provides an integrated mechanism for downloading missions to a DTC, for retrieving data from a DTC and for the display and editing of data stored on a DTC. In this subsection we introduce a subset of user centred requirements for the BASE Control Software expressed as Use Case Diagrams, Scenarios and Message Sequence Charts.

6.2.2.1  Use Case View

Hypothetical ‘transactions’ available to a pilot through the BCS shall include the ability to:- i) erase a cartridge; ii) retrieve (data) from a cartridge; and iii) choose a mission and aircraft (to work on). We now systematically consider details of these transactions, the service(s) to which they belong, their combination (through includes and extends), pre and post conditions and alternative paths.

I. Erase Cartridge

**Overview**

The purpose of this use case is to erase all current data from a Cartridge. Before proceeding with the erase process, checks are made to confirm that a Cartridge is present in the DTM and that the Cartridge contains data.

**Services**

Cartridge Administration

**Includes**

Check for Cartridge, Check Cartridge for Data, Erase Data from Cartridge (see figure 6.2).

**Post-Conditions**

The data on the Cartridge has been erased (assumes Normal Path scenario).

**Scenarios**

Normal Path, No Cartridge Present, No Data on Cartridge, Pilot Chooses Not to Erase Cartridge.

As indicated, constituent use cases of Erase Cartridge (i.e., combined through «includes» associations) are Check for Cartridge, Check Cartridge for Data and Erase Data from Cartridge. These may be described as follows:

**Check for Cartridge**

**Overview**

This is a utility transaction ‘called’ from other high-level use cases. Its purpose is to determine the presence or otherwise of a Cartridge in the hardware before processing operations (e.g., erase, load, etc.) are attempted.

**Post-Conditions**

The MPS has been informed that there is a Cartridge present in the hardware (assumes Normal Path scenario).

**Scenarios**

Normal Path, No Cartridge (Loaded In the Hardware).

**Check Cartridge for Data**

**Overview**

A utility transaction ‘called’ from other high-level use cases in order to determine whether there is data on the Cartridge before processing operations (such as erase) are attempted.

**Pre-Conditions**

There is a Cartridge present in the hardware.

**Post-Conditions**

The MPS has been informed whether or not there is data loaded on the Cartridge (assumes Normal Path scenario for Check for Cartridge).

**Scenarios**

Normal Path, No Data Present (on Cartridge).

---

1 In doing so, we remain faithful to the content of the original case study document.
Tracing Development & Assessment Artifacts

Erase Data from Cartridge

Overview
This utility transaction 'called' from high-level use cases deletes data stored on a Cartridge.

Pre-Conditions
The Pilot has confirmed that he wishes to erase data from the Cartridge.
The Cartridge has to contain data.

Post-Conditions
Data has been erased from the Cartridge.

Scenarios
Normal Path.

A graphical use case model (based on the notation introduced in 4.3.2.1) depicting the combination of these utility use cases towards fulfilment of the Erase Cartridge transaction is shown in figure 6.2.

Figure 6.2 - 'Erase Cartridge Use Case Diagram'

ii. Retrieve from Cartridge

Overview
This use case is used to retrieve Cartridge data and store it on the MPS.

Services
Mission Administration.

Includes
Check for Cartridge, Check Cartridge for Data, Retrieve Data from Cartridge, Display Cartridge Data (see figure 6.3).

Post-Conditions
The data on the Cartridge has been copied onto the MPS.
The data has been displayed to the Pilot (assumes Normal Path scenario).

Scenarios
Normal Path, No Cartridge Present, No Data on Cartridge, Pilot Retrieves a Subset of Data.

Retrieve from Cartridge is further 'extended' by Select Data to Retrieve, described as follows:-

Select Data to Retrieve

Overview
Allows the Pilot to define the data to be retrieved from the Cartridge.

Extends
Retrieve from Cartridge (see figure 6.3).

Post-Conditions
The data types to be retrieved have been defined (assumes Normal Path).

Scenarios
Normal Path, Pilot Chooses Not To Define Data Types To Retrieve.

The 'included' use cases for Retrieve from Cartridge not featured previously - namely Retrieve Data from Cartridge and Display Cartridge Data - may be described as follows:-
Case Study I: A Hypothetical Mission Planning System for the Hawk 100 and 200 Series Aircraft

**Retrieve Data from Cartridge**

**Overview**
The purpose of this use case is to retrieve data that is on the Cartridge and store it on the MPS.

**Pre-Conditions**
A Cartridge containing data is present in the hardware.

**Post-Conditions**
The data on the Cartridge has been copied onto the MPS (assumes Normal Path scenario).

**Scenarios**
Normal Path, Data Type Requested Not Stored on the Cartridge.

**Display Cartridge Data**

**Overview**
The purpose of this use case is to allow the Pilot to view data on the Cartridge.

**Pre-Conditions**
Data has been retrieved from the Cartridge and stored on the MPS.

**Post-Conditions**
The data that is currently loaded onto the Cartridge is displayed for the Pilot (assumes Normal Path scenario).

**Scenarios**
Normal Path, System Has No Data Retrieved From the Cartridge Stored.

Display Cartridge Data is extended by Print Data, which is described as follows:

**Print Data**

**Overview**
The purpose of this use case is to print data items of a particular type.

**Extends**
Display Cartridge Data (see figure 6.3).

**Pre-Conditions**
The data to be printed is being displayed.

**Post-Conditions**
The data item has been printed by the MPS Printer.

**Scenarios**
Normal Path.

Again, a graphical use case model showing combination of these use cases towards fulfilment of the Retrieve from Cartridge transaction is shown in figure 6.3.

![Retrieve From Cartridge Use Case Diagram](image-url)

Figure 6.3 - ‘Retrieve From Cartridge Use Case Diagram’
iii. Choose Mission and Aircraft

Overview
The purpose of this use case is to allow the Pilot to select Mission and Aircraft Data to work on.

Services
Mission Administration.

Includes
Retrieve from Mission Plan (see figure 6.4).

Pre-Conditions
A database must be attached to the MPS.

Post-Conditions
A mission and aircraft have been selected. Data for the selected Mission Plan has been loaded onto the MPS.

Scenarios
New Mission From Open Missions, New Mission From All Missions, No Data for Selected Mission In Mission Plan, Pilot Does Not Initially Select an Aircraft or Mission.

We describe the ‘included’ use case for Choose Mission and Aircraft - Retrieve from Mission Plan - as follows:-

Retrieve from Mission Plan

Overview
A utility use case to retrieve Mission Plan data and store it on the Mission Planning System.

Pre-Conditions
A mission and aircraft have been defined.

Post-Conditions
The data on the Mission Plan for the selected mission and aircraft has been copied onto the MPS (assumes Normal Path).

Scenarios
Normal Path, No data for selected mission In Mission Plan.

Scenario Notes
The events involved in transferring data to the Mission Planning System are iterated according to the number of different types of data stored in the Mission Plan (see also subsection 6.2.2.2.6).

Figure 6.4 shows how Choose Mission and Aircraft and Retrieve from Mission Plan combine.

Figure 6.4 shows how Choose Mission and Aircraft and Retrieve from Mission Plan combine.

![Choose Mission and Aircraft Use Case Diagram](image)

Figure 6.4 - ‘Choose Mission and Aircraft Use Case Diagram’

A summary of services (according to the concepts discussed in subsection 4.3.3.1) to which the Erase Cartridge, Retrieve from Cartridge and Choose Mission and Aircraft use cases contribute appears in figure 6.5.
Case Study I: A Hypothetical Mission Planning System for the Hawk 100 and 200 Series Aircraft

Figure 6.5 - ‘Hawk MPS : Services’

6.2.2.1.1 Instantiation of Use Case View

In this subsection, we instantiate O-Telos classes from the Use Case View of the User Centred Requirements Structure introduced in section 4.3, with a subset of requirements data for the MPS of the Hawk aircraft introduced above. This will allow us to demonstrate support for the representation and traceability of use case artifacts within MATrA. Note, the complete O-Telos code for these artifacts appears in Appendix C, Part One.

Instantiation of UseCaseModel Elements

- Actor Instances
  Pilot in Actor, Token with
  actor_name
  actorName : "Pilot"
  end
  Cartridge in Actor, Token with
  actor_name
  actorName : "Cartridge"
  end
  MPSPrinter in Actor, Token with
  actor_name
  actorName : "MPS Printer"
  end
  MissionPlan in Actor, Token with
  actor_name
  actorName : "Mission Plan"
  end

- Text Nodes (for specifying Pre and Post Conditions)
  CartridgeNode in ModuleNode, Token with
  module_name
  moduleName : "Cartridge"
  end
  MissionPlanningSystemNode in ModuleNode, Token with
  module_name
  moduleName : "MPS"
  end

- Use Case Instances
  Erase Cartridge
  EraseCartridge in UseCase, Token with
  use_case_name
  useCaseName : "Erase Cartridge"
  sub_case
  subCase : False
  post_condition
  postCondition1 : ECPost1NLS
  end
  Post Condition for Erase Cartridge
  ECPost1NLS in
  MatraNaturalLanguageStructure, Token with
  mlns_composite
  mlnsComposite1 : ECPostC1
  mlns_plain_text
  mlnsPlainText1 : ECPostPT1;
  mlnsPlainText2 : ECPostPT2
  mlns_module
  mlnsModule1 : CartridgeNode
  end
  ECPostC1 in MatraNLSComposite, Token with
  preceding_fragment
  precedingFragment : ECPostPT1
  subject_node
  subjectNode : CartridgeNode
  following_fragment
  followingFragment : ECPostPT2
  end
  ECPostPT1 in PlainTextNode, Token with
  mlns_text
Tracing Development & Assessment Artifacts

Check for Cartridge

Check for Cartridge in UseCase, Token with
use_case_name
useCaseName : "Check for Cartridge"
sub_case
subCase : True
post_condition
postCondition1 : CfCPost1NLS
nnlsPlainText : "The data on the ECPostPT2 in PlainTextNode, Token with
nnlsText : " has been erased."
end

Post Condition for Check for Cartridge

CfCPost1NLS in MatraNaturalLanguageStructure, Token with
nnls_composite
nnlsComposite1 : CfCPostC1
nnlsComposite2 : CfCPostC2
nnlsPlainText
nnlsPlainText1 : CfCPostPT1;
nnnlsPlainText2 : CfCPostPT2;
nnlsPlainText3 : CfCPostPT3
nnls_module
nnlsModule1 : MissionPlanningSystemNode;
nnnlsModule2 : CartridgeNode
end

CfCPostC1 in MatraNLSComposite, Token with
precondition
precondition1 : CfCPost1NLS
subject_node
subjectNode : MissionPlanningSystemNode
following_fragment
followingFragment : CfCPostC2
end

CfCPostPT1 in PlainTextNode, Token with
nnls_text
nnlsText : "The data on the end

CfCPostC2 in MatraNLSComposite, Token with
precondition
precondition1 : CfCPost1NLS
subject_node
subjectNode : MissionPlanningSystemNode
following_fragment
followingFragment : CfCPostC3
end

CfCPostPT2 in PlainTextNode, Token with
nnls_text
nnlsText : " has been informed that there is a end

CfCPostPT3 in PlainTextNode, Token with
nnls_text
nnlsText : " present in the hardware."
end

Check Cartridge for Data

CheckCartridge for Data in UseCase, Token with
use_case_name
useCaseName : "Check Cartridge for Data"
sub_case
subCase : True
pre-condition
preCondition1 : CCfDPre1NLS
post_condition
postCondition1 : CCfDPost1NLS
nnlsPlainText : "There is a end

Post Condition for Check Cartridge for Data

CCfDPost1NLS in MatraNaturalLanguageStructure, Token with
nnls_composite
nnlsComposite1 : CCfDPostC1
nnlsPlainText
nnlsPlainText1 : CCfDPostPT1;
nnnlsPlainText2 : CCfDPostPT2
nnls_module
nnlsModule1 : CartridgeNode
end

CCfDPostC1 in MatraNLSComposite, Token with
precondition
precondition1 : CCfDPre1NLS
subject_node
subjectNode : CartridgeNode
following_fragment
followingFragment : CCfDPostPT2
end

CCfDPostPT1 in PlainTextNode, Token with
nnls_text
nnlsText : "There is a end

CCfDPostPT2 in PlainTextNode, Token with
nnls_text
nnlsText : " present in the hardware."
end

Post Condition for Check Cartridge for Data

CCfDPost1NLS in MatraNaturalLanguageStructure, Token with
nnls_composite
nnlsComposite1 : CCfDPostC1
nnlsPlainText
nnlsPlainText1 : CCfDPostPT1;
nnnlsPlainText2 : CCfDPostPT2
nnls_module
nnlsModule1 : MissionPlanningSystemNode;
nnnlsModule2 : CartridgeNode
end

CCfDPostC1 in MatraNLSComposite, Token with
precondition
precondition1 : CCfDPost1NLS
subject_node
subjectNode : MissionPlanningSystemNode
following_fragment
followingFragment : CCfDPostC2
end

CCfDPostPT1 in PlainTextNode, Token with
nnls_text
nnlsText : "The end
Case Study I: A Hypothetical Mission Planning System for the Hawk 100 and 200 Series Aircraft

CCfDPostC2 in MatraNLSComposite, Token with preceding_fragment: CCfDPostPT2
subject_node: CartridgeNode
following_fragment: CCfDPostPT3
end

CCfDPrePT2 in PlainTextNode, Token with mnls_text: has been informed whether or not there is data loaded on the CartridgeNode in ModuleNode, Token with moduleName: `Cartridge'
end

EDfCPre1PT2 in MatraNLSComposite, Token with preceding_fragment: EDfCPre1PT2
subject_node: CartridgeNode
following_fragment: EDfCPre1PT3
end

EDfCPre1PT3 in MatraNLSComposite, Token with preceding_fragment: EDfCPre1PT2
subject_node: CartridgeNode
following_fragment: EDfCPre1PT3
end

Post Condition for Erase Data from Cartridge

EDfCPost1NLS in MatraNaturalLanguageStructure, Token with mnls_composite: EDfCPostC1
mnls_plain_text: EDfCPostPT2
mnls_module: CartridgeNode
end
Tracing Development & Assessment Artifacts

Choose Mission and Aircraft

ChooseMissionandAircraft in UseCase, Token with
  use_case_name
  useCaseName : "Choose Mission and Aircraft"
  sub_case
  subCase : False
  pre_condition
  preCondition1 : CMAPre1NLS
  post_condition
  postCondition1 : CMAPost1NLS;
  postCondition2 : CMAPost2NLS
end

Retrieve from Mission Plan

RetrievefromMissionPlan in UseCase, Token with
  use_case_name
  useCaseName : "Retrieve from Mission Plan"
  sub_case
  subCase : True
  pre_condition
  preCondition1 : RfMFPre1NLS
  post_condition
  postCondition1 : RfMFPPost1NLS
end

- Interaction Instances

    PilotEraseCartridge in Interaction, Token with
    interactor_1
       interactor1 : Pilot
    interactor_2
       interactor2 : EraseCartridge
end

    CartridgeCheckforCartridge in Interaction, Token with
    interactor_1
       interactor1 : Cartridge
    interactor_2
       interactor2 : CheckforCartridge
end

    CartridgeCheckCartridgeforData in Interaction, Token with
    interactor_1
       interactor1 : Cartridge
    interactor_2
       interactor2 : CheckCartridgeforData
end

    CartridgeEraseDatafromCartridge in Interaction, Token with
    interactor_1
       interactor1 : Cartridge
    interactor_2
       interactor2 : EraseDatafromCartridge
end

    PilotRetrievefromCartridge in Interaction, Token with
    interactor_1
       interactor1 : Pilot
    interactor_2
       interactor2 : RetrievefromCartridge
end
CartridgeRetrieveDatafromCartridge in Interaction, Token with interactor_1
   interactor1 : Cartridge
   instantiator_2
   interactor2 : RetrieveDatafromCartridge
end

PilotSelectDatatoRetrieve in Interaction, Token with interactor_1
   interactor1 : Pilot
   instantiator_2
   interactor2 : SelectDatatoRetrieve
end

MPSPrinterPrintData in Interaction, Token with interactor_1
   interactor1 : MPSPrinter
   instantiator_2
   interactor2 : PrintData
end

PilotChooseMissionandAircraft in Interaction, Token with interactor_1
   interactor1 : Pilot
   instantiator_2
   interactor2 : ChooseMissionandAircraft
end

MissionPlanRetrievefromMissionPlan in Interaction, Token with interactor_1
   interactor1 : MissionPlan
   instantiator_2
   interactor2 : RetrievefromMissionPlan
end

MissionPlanChooseMissionandAircraft in Interaction, Token with interactor_1
   interactor1 : MissionPlan
   instantiator_2
   interactor2 : ChooseMissionandAircraft
end

* Includes Instances

EraseCartridgeCheckforCartridge in
Includes, Token with
   includes_base
   includesBase : EraseCartridge
   instantiator_include
   includesInclude : CheckforCartridge
end

EraseCartridgeCheckCartridgeforData in
Includes, Token with
   includes_base
   includesBase : EraseCartridge
   instantiator_include
   includesInclude : CheckCartridgeforData
end

EraseCartridgeEraseDatafromCartridge in
Includes, Token with
   includes_base
   includesBase : EraseCartridge
   instantiator_include
   includesInclude : EraseDatafromCartridge
end

RetrievefromCartridgeCheckforCartridge in
Includes, Token with
   includes_base
   includesBase : RetrievefromCartridge
   instantiator_include
   includesInclude : CheckforCartridge
end

RetrievefromCartridgeCheckCartridgeforData in
Includes, Token with
   includes_base
   includesBase : RetrievefromCartridge
   instantiator_include
   includesInclude : CheckCartridgeforData
end

RetrievefromCartridgeRetrieveDatafromCartridge in
Includes, Token with
   includes_base
   includesBase : RetrievefromCartridge
   instantiator_include
   includesInclude : CheckCartridgeforData
end

RetrievefromCartridgeDisplayCartridgeData in
Includes, Token with
   includes_base
   includesBase : RetrievefromCartridge
   instantiator_include
   includesInclude : DisplayCartridgeData
end

ChooseMissionandAircraftRetrievefromMissionPlan in
Includes, Token with
   includes_base
   includesBase : ChooseMissionandAircraft
   instantiator_include
   includesInclude : RetrievefromMissionPlan
end

• Extends Instances

PrintDataDisplayCartridgeData in Extends, Token with
   extends_base
   extendsBase : PrintData
   instantiator_extend
   extendsExtend : DisplayCartridgeData
end

SelectDatatoRetrieveRetrievefromCartridge in
Extends, Token with
   extends_base
   extendsBase : SelectDatatoRetrieve
   instantiator_extend
   extendsExtend : RetrievefromCartridge
end

Erase Cartridge Use Case Model

EraseCartridgeModel in UseCaseModel,
Token with model_name
   modelName : "Erase Cartridge Model"
   ucm_comments
   ucmComments : EraseCartridgeComments
   ucm_use_case
   ucmUseCase1 : EraseCartridge;
   ucmUseCase2 : CheckforCartridge;
   ucmUseCase3 : CheckCartridgeforData;
   ucmUseCase4 : EraseDatafromCartridge
   ucm_actor
   ucmActor1 : Pilot;
   ucmActor2 : Cartridge
To show support for the representation and traceability of scenarios within MATrA, we present the UCRS Interaction View for the Erase Cartridge use case from 6.2.2.1(i). Featured paths include Normal Path, No Cartridge, No Data on Cartridge and Pilot Chooses Not To Erase the Data. In each case, we introduce logical (i.e., user) level representations in both textual and Message Sequence Chart forms (sections 6.2.2.2.1 through 6.2.2.2.4), along with their 'equivalent' O-Telos implementation. We further include fragments of scenarios (sections 6.2.2.2.5 and 6.2.2.2.6) for the Retrieve from Cartridge (Normal Path) and Choose Mission and Aircraft (New Mission from Open Missions) use cases to demonstrate features of UCRS not present in the above examples (the full scenarios appear in Appendix C, Part Two). Note, constituent (i.e. included) use cases of Erase Cartridge, such as Check for Cartridge and Check Cartridge for Data are modelled separately, with their O-Telos representations appearing in Appendix C, Part One.
6.2.2.2.1 Erase Cartridge - Normal Path

2. C[IR] The [MPS] asks the [Cartridge] whether there is a cartridge present in the hardware?
3. C[IP] The [Cartridge] responds to the [MPS] that there is a cartridge present in the hardware.
4. C[IP] The [MPS] asks the [Cartridge] whether there is data on the cartridge?
5. C[IP] The [Cartridge] responds to the [MPS] that there is data on the cartridge.
6. C[IP] The [MPS] informs the [Pilot] that there is data on the cartridge.
7. C[SR] The [Pilot] indicates to the [MPS] that it is OK to proceed with Cartridge Erase.
8. C[IP] The [MPS] tells the [Cartridge] to erase the current data.
9. C[SP] The [Cartridge] indicates to the [MPS] that all current data has been deleted.
10. C[SP] The [MPS] indicates to the [Pilot] that all current data has been deleted.

Figure 6.6 - 'MSC: Erase Cartridge - Normal Path'
Instantiation of Interaction View

Instantiation of Featured Instance Definitions

Instantiation of Erase Cartridge Interaction Model

Instantiation of Erase Cartridge Normal Path (Partial)
Case Study I: A Hypothetical Mission Planning System for the Hawk 100 and 200 Series Aircraft

ProceedCartridgeEraseRequestSSN in SequenceNumber, Token with sequence_no
sequenceNo : 7 end

DataErasedNotificationSSN in SequenceNumber, Token with sequence_no
sequenceNo : 10 end

DataErasedAcknowledgementSSN in SequenceNumber, Token with sequence_no
sequenceNo : 11 end

MainScreenDisplay_1_NormalPathSSN in SequenceNumber, Token with sequence_no
sequenceNo : 12 end

CheckForCartridgeRequest_EC_NP_ISN in SequenceNumber, Token with sequence_no
sequenceNo : 2 end

ConfirmCartridgePresent_EC_NP_ISN in SequenceNumber, Token with sequence_no
sequenceNo : 3 end

CheckCartridgeforDataRequest_EC-NP_ISN in SequenceNumber, Token with sequence_no
sequenceNo : 4 end

ConfirmCartridgeDataPresent_EC-NP_ISN in SequenceNumber, Token with sequence_no
sequenceNo : 5 end

EraseCurrentDataRequest_EC_NP_ISN in SequenceNumber, Token with sequence_no
sequenceNo : 8 end

CurrentDataErasedNotification_EC_NP_ISN in SequenceNumber, Token with sequence_no
sequenceNo : 9 end

EraseCartridge_NormalPathTSV in TsnScenarioViewpoint, Token with tsv_tsn_comm
  tsvnComm1 : EraseCartridgeRequestTextDescription;
  tsvnComm2 : ProceedCartridgeEraseRequestTextDescription;
  tsvnComm3 : ConfirmDataPresentOnCartridgeTextDescription;
  tsvnComm4 : DataErasedNotificationTextDescription;
  tsvnComm5 : DataErasedAcknowledgementTextDescription;
  tsvnComm6 : MainScreenDisplay_1TextDescription end

EraseCartridge_NormalPathMSV in MscScenarioViewpoint, Token with
  msv_msc_comm
    msvMsgComm1 : EraseCartridgeRequestMSCDescription;
    msvMsgComm2 : ConfirmDataPresentOnCartridgeMSCDescription;
    msvMsgComm3 : ProceedCartridgeEraseRequestMSCDescription;
    msvMsgComm4 : DataErasedNotificationMSCDescription;
    msvMsgComm5 : DataErasedAcknowledgementMSCDescription;
    mscMsgComm6 : MainScreenDisplay_1MSCDescription end

Erase Cartridge Request Event Instantiation

EraseCartridgeRequest in CommunicationEvent, Token with interaction_type
  interaction_type = "SR"
sequence_no
sequenceNo1 :
EraseCartridgeRequest_NormalPathSSN;
sequenceNo2 :
EraseCartridgeRequest_NoCartridgeSSN;
sequenceNo3 :
EraseCartridgeRequest_NoDataOnCartridgeSSN;
sequenceNo4 :
EraseCartridgeRequest_PilotChoosesNotToEraseDataSSN
  tsn_communication_event
  tsnCommunicationEvent :
    tsnCommunicationEvent :
      msc_communication_event
      mscCommunicationEvent :
        EraseCartridgeRequestMSCDescription end

EraseCartridgeRequestEventText in
  ScenarioEventNaturalLanguageStructure, Token with
  mnls_composite
    mnlsComposite1 : ECREC1;
    mnlsComposite2 : ECREC2;
    mnlsComposite3 : ECREC3
  mnls_plain_text
    mnlsPlainText1 : ECREPT1;
    mnlsPlainText2 : ECREPT2;
    mnlsPlainText3 : ECREPT3;
    mnlsPlainText4 : ECREPT4
tsn_sender_node
  tsnSenderId : PilotInstance
  tsn_receiver_node
  tsnReceiverNodeId : MPSInstance
tsn_message_node
  tsnMessageNode :
    EraseCartridgeRequest end

ECREC1 in MatraNLSComposite, Token with preceding_fragment
  precedingFragment : ECREPT1 subject_node
  subjectNode : PilotInstance following_fragment
  followingFragment : ECREC2 end

ECREC2 in MatraNLSComposite, Token with...

292
Confirm Data Present on Cartridge Event Instantiation

ConfirmDataPresentonCartridge in CommunicationEvent, Token with interaction_type
interactionType : "IP"
sequence_no
sequenceNo1 : ConfirmDataPresentonCartridge_NormalPathsSN;
sequenceNo2 : ConfirmDataPresentonCartridge_PilotChoose
sNotToEraseDataSN
follows_from
followsFrom : ConfirmCartridgeDataPresent
tsn_communication_event
tsnCommunicationEvent :
ConfirmDataPresentonCartridgeTextDescription
msc_communication_event
mscCommunicationEvent :
ConfirmDataPresentonCartridgeMSCDescription end

ConfirmDataPresentonCartridgeTextDescription in TsnCommunication, Token with communication_description
communicationDescription :
ConfirmDataPresentonCartridgeEventText end

ConfirmDataPresentonCartridgeEventText in ScenarioEventNaturalLanguageStructure, Token with

mnlS_composite
mnlSComposite1 : CDPOCEC1;
mnlSComposite2 : CDPOCEC2;
mnlSComposite3 : CDPOCEC3
mnlS_plain_text
mnlSPlainText1 : CDPOCEPT1;
mnlSPlainText2 : CDPOCEPT2;
mnlSPlainText3 : CDPOCEPT3;
mnlSPlainText4 : CDPOCEPT4
tsn_sender_node
tsnSenderNode : MPSInstance
tsn_receiver_node
tsnReceiverNode : PilotInstance
tsn_message_node
tsnMessageNode :
CartridgeDataFoundNotification end

CDPOCEC1 in MatraNLSComposite, Token with preceding Fragment
precedingFragment : CDPOCEPT1
subject_node
subjectNode : MPSInstance
following_fragment
followingFragment : CDPOCEC2 end

CDPOCEC2 in MatraNLSComposite, Token with preceding Fragment
precedingFragment : CDPOCEPT2
subject_node
subjectNode : PilotInstance
following_fragment
followingFragment : CDPOCEC3 end

CDPOCEC3 in MatraNLSComposite, Token with preceding Fragment
precedingFragment : CDPOCEPT3
subject_node
subjectNode : CartridgeDataFoundNotification
following_fragment
followingFragment : CDPOCEPT4 end

EraseCartridgeRequest in Message, Token with
message_name
messageName : "Erase Cartridge Request"
tsn_msg_parameter
tsnMsgParameter :
EraseCartridgeRequestTsnParameter
msc_msg_parameter
mscMsgParameter :
EraseCartridgeRequestMscParameter
end

EraseCartridgeRequestTsnParameter in MessageDescription, Token with msg_parameter
msgParameter : "Erase Cartridge option"
end

EraseCartridgeRequestMscParameter in MessageDescription, Token with msg_parameter
msgParameter : "Erase Cartridge"
end

EraseCartridgeRequestMSCDescription in MacCommunication, Token with link_name
linkName : "unspecified"
synchronisation
_synchronisation : = "sim"
frequency
_frequency : = "aperiodic"
delayed
_Delayed : False
msc_sender_instance
mscSenderInstance : PilotInstance
msc_receiver_instance
mscReceiverInstance : MPSInstance
msc_message
mscMessage : EraseCartridgeRequest
end
Case Study I: A Hypothetical Mission Planning System for the Hawk 100 and 200 Series Aircraft

CDPCEP1 in PlainTextNode, Token with
\text{mnls_text : "The "}
end

CDPCEP2 in PlainTextNode, Token with
\text{mnls_text : "indicates to the "}
end

CDPCEP3 in PlainTextNode, Token with
\text{mnls_text : "that "}
end

CDPCEP4 in PlainTextNode, Token with
\text{mnls_text : "."}
end

CartridgeDataFoundNotification in Message, Token with
message_name : "Cartridge Data Found Notification"
\text{tsn_msg_parameter : CartridgeDataFoundNotificationTsnParameter}
\text{msg_parameter : "there is data on the cartridge"}
end

CartridgeDataFoundNotificationTsnParameter in MessageDescription, Token with
\text{msg_parameter : "Confirm Data on Cartridge"}
end

ConfirmDataPresentonCartridge in MscCommunication, Token with
\text{link_name : "unspecified"}
synchronisation : "sim"
\text{frequency : "aperiodic"
delayed : False}
\text{msc_sender_instance : MPSInstance}
\text{msc_receiver_instance : PilotInstance}
\text{msc_message : CartridgeDataFoundNotification}
end

Proceed Cartridge Erase Request Event Instantiation

ProceedCartridgeEraseRequest in CommunicationEvent, Token with
\text{interaction_type : "SR"}
sequence_no : ProceedCartridgeEraseRequestSSN
follows_from : ConfirmDataPresentonCartridge
tsn_communication_event : ProceedCartridgeEraseRequestTextDescription
\text{msc_communication_event : ProceedCartridgeEraseRequestMSCDescription}
end

ProceedCartridgeEraseRequestTextDescription in TsnCommunication, Token with
\text{communication_description : ProceedCartridgeEraseRequestEventText}
end

ProceedCartridgeEraseRequestEventText in ScenarioEventNaturalLanguageStructure, Token with
\text{msl_composite : ProceedCartridgeEraseRequestEventTextComposite}
\text{tsn_sender_node : PilotInstance}
\text{tsn_receiver_node : MPSInstance}
\text{tsn_message_node : ProceedEraseCartridge}
end

PCEREC1 in MatraNLSComposite, Token with
\text{preceeding_fragment : PCEREPT1}
\text{subject_node : PilotInstance}
\text{following_fragment : PCEREPT2}
end

PCEREPT1 in PlainTextNode, Token with
\text{mnls_text : "The "}
end

PCEREPT2 in PlainTextNode, Token with
\text{mnls_text : "indicates to the "}
end

PCEREPT3 in PlainTextNode, Token with
\text{mnls_text : "that it is "}
end
Tracing Development & Assessment Artifacts

PCEREPT4 in PlainTextNode, Token with

ProceedEraseCartridge in Message, Token with

ProceedEraseCartridgeTsnParameter in MessageDescription, Token with

ProceedEraseCartridgeMscParameter in MessageDescription, Token with

ProceedCartridgeEraseRequestMSCDescription in MscCommunication, Token with

Data Erased Notification Event Instantiation

DataErasedNotification in CommunicationEvent, Token with

DataErasedNotificationTsnParameter in MessageDescription, Token with

DataErasedNotificationMSCDescription in MscCommunication, Message with

CartridgeErasedNotification in Message, Token with

CartridgeErasedNotificationTsnParameter in MessageDescription, Token with

CartridgeErasedNotificationMSCDescription in MscCommunication, Message with

DENEC1 in MatraNLSCComposite, Token with

DENEC2 in MatraNLSCComposite, Token with

DENEC3 in MatraNLSCComposite, Token with

DENEP1 in PlainTextNode, Token with

DENEP2 in PlainTextNode, Token with

DENEP3 in PlainTextNode, Token with

DENEP4 in PlainTextNode, Token with

CartridgeErasedNotification in Message, Token with
Case Study 1: A Hypothetical Mission Planning System for the Hawk 100 and 200 Series Aircraft

```
msg_parameter
msgParameter : "all current data has
been deleted"
end

CartridgeErasedNotificationMSCParameter
in MessageDescription, Token with
msg_parameter
  msgParameter : "Confirm All Data
Deleted"
end

DataErasedNotificationMSCDescription in
MscCommunication, Token with
  link_name
    linkName : "unspecified"
  synchronisation
    _Synchronisation : = "sim"
  frequency
    _Frequency : "aperiodic"
  delayed
    _Delayed : False
msc_sender_instance
  mscSenderInstance : MPSInstance
msc_receiver_instance
  mscReceiverInstance : PilotInstance
msc_message
  mscMessage : CartridgeErasedNotification
end

Data Erased Acknowledgement Event
Instantiation

DataErasedAcknowledgement in
CommunicationEvent, Token with
  interaction_type
    interactionType : "IP"
  sequence_no
    sequenceNo1 :
      DataErasedAcknowledgementSSN
        follows_from :
          followsFrom : DataErasedNotification
            tsn_communication_event
              tsnCommunicationEvent :
                DataErasedAcknowledgementTextDescription
                  msc_communication_event
                    DataErasedAcknowledgementMSCDescription
end

DataErasedAcknowledgementTextDescription in
TsnCommunication, Token with
  communication_description
    communicationDescription :
      DataErasedAcknowledgementEventText
end

DataErasedAcknowledgementEventText in
ScenarioEventNaturalLanguageStructure, Token with
  mnlS_composite
    mnlSComposite1 : DEAEC1;
    mnlSComposite2 : DEAEC2;
    mnlSComposite3 : DEAEC3
  mnlS_plain_text
    mnlSPlainText1 : DEAEP1;
    mnlSPlainText2 : DEAEP2;
    mnlSPlainText3 : DEAEP3;
    mnlSPlainText4 : DEAEP4
tsn_sender_node
  tsnSenderNode : PilotInstance
tsn_receiver_node
  tsnReceiverNode : MPSInstance
tsn_message_node
  tsnMessageNode : AcknowledgeErase
end

DEAEC1 in MatraNLSCComposite, Token with
  preceding_fragment
    DEAEC1
subject_node
  subjectNode : PilotInstance
following_fragment
  followingFragment : DEAEC2
end

DEAEC2 in MatraNLSCComposite, Token with
  preceding_fragment
    DEAEC2
subject_node
  subjectNode : AcknowledgeErase
following_fragment
  followingFragment : DEAEC3
end

DEAEC3 in MatraNLSCComposite, Token with
  preceding_fragment
    DEAEC3
subject_node
  subjectNode : MPSInstance
following_fragment
  followingFragment : DEAEC4
end

DEAEC4 in MatraNLSCComposite, Token with
  preceding_fragment
    DEAEC4
subject_node
  subjectNode : AcknowledgeErase
following_fragment
  followingFragment : DEAEC5
end

AcknowledgeErase in Message, Token with
  message_name
    messageName : "Acknowledge Erase"
tsn_msg_parameter
  tsnMsgParameter :
    AcknowledgeEraseTsnParameter
msc_msg_parameter
  mscMsgParameter :
    AcknowledgeEraseMscParameter
end

AcknowledgeEraseTsnParameter in
MessageDescription, Token with
  msg-parameter
    msgParameter : acknowledgement
end

AcknowledgeEraseMscParameter in
MessageDescription, Token with
  msg-parameter
    msgParameter : 'OK: Acknowledge
Erase'
end
```

mscSenderInstance : PilotInstance
msc_receiver_instance
mscReceiverInstance : MPSInstance
msc_message : AcknowledgeErase
end

Main Screen Display (from Erase Cartridge) Event Instantiation

MainScreenDisplay_l in CommunicationEvent, Token with interaction_type interactionType : "IP"
sequenceNo1 :
MainScreenDisplay_l_NormalPathSSN;
sequenceNo2 :
MainScreenDisplay_l_NoCartridgeSSN;
sequenceNo3 :
MainScreenDisplay_l_NoDataOnCartridgeSSN;
sequenceNo4 :
MainScreenDisplay_l_PilotChoosesNotToEraseDataSSN
follows_from
followsFrom1 :
DataErasedAcknowledgement;
followsFrom2 :
NoCartridgeAcknowledgement;
followsFrom3 :
NoCartridgeDataPresentAcknowledgement;
followsFrom4 :
CancelEraseRequest
tsn_communication_event
MainScreenDisplay_lTextDescription
msc_communication_event
MainScreenDisplay_lMSCDescription
end

MainScreenDisplay_lTextDescription in TanCommunication, Token with communication_description communicationDescription : MainScreenDisplay_lEventText
end

MainScreenDisplay_lEventText in ScenarioEventNaturalLanguageStructure, Token with
mlns_composite
mlnsComposite1 : MSD_1EC1;
mlnsComposite2 : MSD_1EC2;
mlnsComposite3 : MSD_1EC3
mlns_plain_text
mlnsPlainText1 : MSD_1EPT1;
mlnsPlainText2 : MSD_1EPT2;
mlnsPlainText3 : MSD_1EPT3;
mlnsPlainText4 : MSD_1EPT4
tsn_sender_node
tsnSenderNode : MPSInstance
tsn_receiver_node
tsnReceiverNode : PilotInstance
tsn_message_node
tsnMessageNode : MainScreen_1
end

MSD_1EC1 in MatraNLSComposite, Token with preceding_fragment
precedingFragment : MSD_1EPT1
subject_node
subjectNode : MPSInstance
following_fragment
followingFragment : MSD_1EC2
end

MSD_1EC2 in MatraNLSComposite, Token with preceding_fragment
precedingFragment : MSD_1EPT2
subject_node
subjectNode : PilotInstance
following_fragment
followingFragment : MSD_1EC3
end

MSD_1EC3 in MatraNLSComposite, Token with preceding_fragment
precedingFragment : MSD_1EPT3
subject_node
subjectNode : MainScreen_1
following_fragment
followingFragment : MSD_1EPT4
end

MSD_1EPT1 in PlainTextNode, Token with mlns_text
mlnsText : "The "
end

MSD_1EPT2 in PlainTextNode, Token with mlns_text
mlnsText : " displays to the "
end

MSD_1EPT3 in PlainTextNode, Token with mlns_text
mlnsText : " the "."
end

MainScreen_1 in Message, Token with
message_name
messageName : "Main Screen"
end

MainScreenDisplay_lMSCDescription in MscCommunication, Token with
link_name
linkName : "unspecified"
synchronisation
_syncronisation : = "sin"
frequency
_frequency : "aperiodic"
delayed
_Delayed : False
msc_sender_instance
mscSenderInstance : MPSInstance
msc_receiver_instance
mscReceiverInstance : PilotInstance
msc_message
mscMessage : MainScreen_1
end
6.2.2.2 Erase Cartridge - No Cartridge Present


2. C[IR] The [IR MPS] asks the [IR Cartridge] whether there [IR is a cartridge present in the hardware? Request Confirm Cartridge].

3. C[IP] The [IP Cartridge] responds to the [IP MPS] that [IP there is no cartridge present in the hardware; No Data].

4. C[IP] The [IP MPS] informs the [IP Pilot] that there is [IP no cartridge is present in the hardware; No Cartridge].


---

The diagram above illustrates the sequence of events in the Erase Cartridge - No Cartridge scenario. The scenario includes the following events:

- EraseCartridgeRequest
- NoCartridgePresentNotification
- NoCartridgeAcknowledgement
- MainScreenDisplay

The sequence of events is as follows:

1. EraseCartridgeRequest
2. NoCartridgePresentNotification
3. NoCartridgeAcknowledgement
4. MainScreenDisplay

---

Instantiation of Erase Cartridge: No Cartridge Present

EraseCartridge_NoCartridge in Scenario, Token with scenarioTitle : "Erase Cartridge - No Cartridge Present" is_exception : True scenario_event : EraseCartridgeRequest; scenarioEvent2 : NoCartridgePresentNotification; scenarioEvent3 : NoCartridgeAcknowledgement; scenarioEvent4 : MainScreenDisplay; includes_scenario : includesScenario1 : CheckforCartridge_NoCartridge included_event : includedEvent1 : CheckforCartridgeRequest; includedEvent2 : NoCartridgePresent scn_seq_no : scnSeqNo1 : EraseCartridgeRequest_NoCartridgeSSN; scnSeqNo2 : NoCartridgePresentNotificationSSN; scnSeqNo3 : NoCartridgeAcknowledgementSSN; scnSeqNo4 : MainScreenDisplay_1_NoCartridgeSSN scnIncludedSeqNo : scnIncludedSeqNo1 : CheckforCartridgeRequest_EC_NC_SS; scnIncludedSeqNo2 : NoCartridgePresent_EC_NC_SS tsn_viewpoint : tsnViewpoint : EraseCartridge_NoCartridgeTSV msc_viewpoint : mscViewpoint : EraseCartridge_NoCartridgeMSV end

---

Again, 'included_event' may be instantiated using a variation of the rule in App. A, Pt. 2 (vii. c); see also App. C, Pt. 1 for O-Telos implementation of included_event classes

EraseCartridgeRequest_NoCartridgeSSN in SequenceNumber, Token with sequence_no : sequenceNo : 1 end NoCartridgePresentNotificationSSN in
Tracing Development & Assessment Artifacts

SequenceNumber, Token with
sequence_no
sequenceNo : 4
end

NoCartridgeAcknowledgementSSN in
SequenceNumber, Token with
sequence_no
sequenceNo : 5
end

MainScreenDisplay_1_NoCartridgeSSN in
SequenceNumber, Token with
sequence_no
sequenceNo : 6
end

CheckforCartridgeRequest_EC_NC_ISN in
SequenceNumber, Token with
sequence_no
sequenceNo : 2
end

NoCartridgePresent_EC_NC_ISN in
SequenceNumber, Token with
sequence_no
sequenceNo : 3
end

EraseCartridge_NoCartridgeTSV in
TsnScenarioViewpoint, Token with
tsn_tsn_comm
tsnComm1 :
EraseCartridgeRequestTextDescription;

tsnComm2 :
NoCartridgePresentNotificationTextDescription;

tsnComm3 :
NoCartridgeAcknowledgementTextDescription;

MainScreenDisplay_1TextDescription
end

EraseCartridge_NoCartridgeMSV in
MscScenarioViewpoint, Token with
msv_msc_comm
msvComm1 :
EraseCartridgeRequestMSCDescription;

msvComm2 :
NoCartridgePresentNotificationMSCDescription;

msvComm3 :
NoCartridgeAcknowledgementMSCDescription;

MainScreenDisplay_1MSCDescription
end

No Cartridge Present Notification Event
Instantiation

NoCartridgePresentNotification in
CommunicationEvent, Token with
interaction_type
interactionType : "IP"
sequence_no
sequenceNo1 :
NoCartridgePresentNotificationSSN
follows_from :
NoCartridgePresent

tsn_communication_event

NoCartridgePresentNotificationTextDescription

msc_communication_event

NoCartridgePresentNotificationMSCDescription
end

NoCartridgePresentNotificationTextDescription in TsnCommunication, Token with
communication_description
communicationDescription :
NoCartridgePresentNotificationEventText
end

NoCartridgePresentNotificationEventText in
ScenarioEventNaturalLanguageStructure, Token with
mls_composite

mlsComposite1 : NCPNEC1;

mlsComposite2 : NCPNEC2;

mlsComposite3 : NCPNEC3

mlsPlain_text

mlsPlainText1 : NCPNEPT1;

mlsPlainText2 : NCPNEPT2;

mlsPlainText3 : NCPNEPT3;

mlsPlainText4 : NCPNEPT4
tsn_sender_node
tsnSenderNode : MPSInstance
tsn_receiver_node
tsnReceiverNode : PilotInstance
tsn_message_node
tsnMessageNode :
NoCartridgeNotification
end

NCPNEC1 in MatraNLSComposite, Token with
preceding_fragment

precedingFragment : NCPNEPT1

subject_node

subjectNode : MPSInstance

following_fragment

followingFragment : NCPNEC2
end

NCPNEC2 in MatraNLSComposite, Token with
preceding_fragment

precedingFragment : NCPNEPT2

subject_node

subjectNode : PilotInstance

following_fragment

followingFragment : NCPNEC3
end

NCPNEC3 in MatraNLSComposite, Token with
preceding_fragment

precedingFragment : NCPNEPT4

subject_node

subjectNode : NoCartridgeNotification

following_fragment

followingFragment : NCPNEC4
end

NCPNEPT1 in PlainTextNode, Token with
mls_text

mlsText : "The"
end

NCPNEPT2 in PlainTextNode, Token with
mls_text

mlsText : "informs the"
end

NCPNEPT3 in PlainTextNode, Token with
mls_text

mlsText : "that there is"
end

NCPNEPT4 in PlainTextNode, Token with
mls_text

mlsText : ""
end

NoCartridgeNotification in Message, Token with
message_name

messageName : "No Cartridge Notification"
Case Study 1: A Hypothetical Mission Planning System for the Hawk 100 and 200 Series Aircraft

No Cartridge Notification TsnParameter in MessageDescription, Token with msg parameter
  msgParameter : "no cartridge present in the hardware" end

No Cartridge Notification MscParameter in MessageDescription, Token with msg-parameter
  msgParameter : "Report No Cartridge" end

No Cartridge Present Notification MscDescription in MscCommunication, Token with link name
  linkName : "unspecified"
  _Synchronisation : = "sim"
  _Frequency : "aperiodic"
  _Delayed : False
  msc_sender_instance
    mscSenderInstance : MPSInstance
  msc_receiver_instance
    mscReceiverInstance : PilotInstance
  msc_message
    mscMessage : NoCartridgeNotification
end

No Cartridge Acknowledgement Event Instantiation

NoCartridgeAcknowledgement in CommunicationEvent, Token with interaction_type
  interactionType : "IP"
  sequence no
  sequenceNo1 : NoCartridgeAcknowledgementSSN follows_from
  followsFrom : NoCartridgePresentNotification
  tsn_communicationEvent
    tsnCommunicationEvent : NoCartridgeAcknowledgementTextDescription
    mscCommunicationEvent
      mscCommunicationEvent : NoCartridgeAcknowledgementMSCDescription
end

NoCartridgeAcknowledgementTextDescription in TsnCommunication, Token with
  communication_description
    communicationDescription : NoCartridgeAcknowledgementEventText
end

NoCartridgeAcknowledgementEventText in ScenarioEventNaturalLanguageStructure, Token with
  mnlComposite
    mnlComposite1 : NCAEC1;
    mnlComposite2 : NCAEC2;
    mnlComposite3 : NCAEC3
  mnlPlainText
    mnlPlainText1 : NCAEPT1;
    mnlPlainText2 : NCAEPT2;
    mnlPlainText3 : NCAEPT3;
    mnlPlainText4 : NCAEPT4
  tsn_sender_node
    tsnSenderNode : PilotInstance
end

NoCartridgeAcknowledgementMscParameter in MessageDescription, Token with
  msg parameter
  msgParameter : "Report No Cartridge" end

NCAEC1 in MatraNLSComposite, Token with
  preceding_fragment : NCAEPT1
  subject_node
    subjectNode : MPSInstance
  following_fragment : NCAEPT2
end

NCAEC2 in MatraNLSComposite, Token with
  preceding_fragment : NCAEPT3
  subject_node
    subjectNode : MPSInstance
  following_fragment : NCAEPT4
end

NCAEPT1 in PlainTextNode, Token with
  mnl_text
    mnlText : "The "
end

NCAEPT2 in PlainTextNode, Token with
  mnl_text
    mnlText : " sends an "
end

NCAEPT3 in PlainTextNode, Token with
  mnl_text
    mnlText : " to the "
end

NCAEPT4 in PlainTextNode, Token with
  mnl_text
    mnlText : "."
end

AcknowledgeNoCartridge in Message, Token with
  message_name
    messageName : "Acknowledge No Cartridge"
  tsn_msg_parameter
    tsnMsgParameter : AcknowledgeNoCartridgeTsnParameter
  msc_msg_parameter
    mscMsgParameter : AcknowledgeNoCartridgeMscParameter
end

AcknowledgeNoCartridge in MessageDescription, Token with
  msg parameter
  msgParameter : "acknowledgement" end

AcknowledgeNoCartridge in MessageDescription, Token with
  msg parameter
  msgParameter : "OK: Acknowledge No Cartridge" end

NoCartridgeAcknowledgementEventText in ScenarioEventNaturalLanguageStructure, Token with
  mnlComposite
    mnlComposite1 : NCAEC1;
    mnlComposite2 : NCAEC2;
    mnlComposite3 : NCAEC3
  mnlPlainText
    mnlPlainText1 : NCAEPT1;
    mnlPlainText2 : NCAEPT2;
    mnlPlainText3 : NCAEPT3;
    mnlPlainText4 : NCAEPT4
  tsn_sender_node
    tsnSenderNode : PilotInstance
end
6.2.2.2.3 Erase Cartridge - No Data on Cartridge

1. **C(SR)** The [69 Pilot] selects the [scenario : Erase Cartridge Request] on the [scenario : MPS Main Screen].
2. **C(IR)** The [scenario : User] asks the [scenario : Pilot] whether there is a cartridge present in the hardware? [scenario : Request Confirm Cartridge].
3. **C(IP)** The [scenario : Pilot] responds to the [scenario : User] that there is a cartridge present in the hardware [scenario : Confirm Cartridge].
4. **C(IR)** The [scenario : User] asks the [scenario : Pilot] whether there is data on the cartridge? [scenario : Request Confirm Cartridge Data].
5. **C(IP)** The [scenario : Pilot] responds to the [scenario : User] that there is no data on the cartridge [scenario : No Cartridge Data].
6. **C(IP)** The [scenario : Pilot] informs the [scenario : User] that there is no data on the cartridge [scenario : No Cartridge Data Notification].
8. **C(IP)** The [scenario : User] displays to the [scenario : Pilot] the [scenario : Main Screen].

Instantiation of Erase Cartridge: No Data on Cartridge

EraseCartridge_NoDataOnCartridge in Scenario, Token with
scenario_title : "Erase Cartridge - No Data on Cartridge"
is_exception : True
scenario_event
scenarioEvent1 : EraseCartridgeRequest;
scenarioEvent2 : NoCartridgeDataPresentNotification;
scenarioEvent3 : NoCartridgeDataPresentAcknowledgement;
scenarioEvent4 : MainScreenDisplay_1
.includes_scenario
.includesScenario1:
  .CheckCartridge forCartridge_NormalPath;
  .includesScenario2:
  .CheckCartridge forCartridge_NoData
.includes_event
  .includedEvent1:
  .CheckCartridge forCartridgeRequest;
  .includedEvent2:
  .ConfirmCartridge forCartridgePresent;
  .includedEvent3:
  .CheckCartridge forCartridgeRequest;
.includes_event
  .includedEvent4:
  .NoCartridgeDataPresent

scn_seq_no
scnSeqNo1:
  .EraseCartridge forCartridgeRequest_NoDataOnCartridgeSSN;
scnSeqNo2:
  .NoCartridgeDataPresentNotificationSSN;
scnSeqNo3:
  .NoCartridgeDataPresentAcknowledgementSSN;
MainScreenDisplay_1_NoDataOnCartridgeSSN
scn_included_seq_no
scnIncludedSeqNo1:
  .CheckCartridge forCartridgeRequest_EC_ND_ISN;
scnIncludedSeqNo2:
  .ConfirmCartridge forCartridgeRequest_EC_ND_ISN;
scnIncludedSeqNo3:
  .CheckCartridge forCartridgeRequest_EC_ND_ISN;
scnIncludedSeqNo4:
  .NoCartridgeDataPresent_EC_ND_ISN

tsn_viewpoint
EraseCartridge_NoDataOnCartridgeTSV
msc_viewpoint
EraseCartridge_NoDataOnCartridgeMSV

-- again 'included_event' may be instantiated using a variation of the rule in App. A, Pt2 (vi. c); see also App. C, Pt. 1 for O-Telos implementation of included_event classes

EraseCartridge forCartridgeRequest_NoDataOnCartridgeSSN in SequenceNumber, Token with sequence_no
  sequenceNo : 1
end

NoCartridgeDataPresentNotificationSSN in SequenceNumber, Token with sequence_no
  sequenceNo : 6
end

NoCartridgeDataPresentAcknowledgementSSN in SequenceNumber, Token with sequence_no
  sequenceNo : 7
end

MainScreenDisplay_1_NoDataOnCartridgeSSN in SequenceNumber, Token with sequence_no
  sequenceNo : 8
end

CheckCartridge forCartridgeRequest_EC_ND_ISN in SequenceNumber, Token with sequence_no
  sequenceNo : 2
end

ConfirmCartridge forCartridgeRequest_EC_ND_ISN in SequenceNumber, Token with sequence_no
  sequenceNo : 3
end

NoCartridgeDataPresent_EC_ND_ISN in SequenceNumber, Token with sequence_no
  sequenceNo : 4
end

NoCartridgeDataPresent_EC_ND_ISN in SequenceNumber, Token with sequence_no
  sequenceNo : 5
end

EraseCartridge_NoDataOnCartridgeTSV in TsnScenarioViewpoint, Token with
tsv_tsn_comm
  tsvTsnComm1 :
  .EraseCartridgeRequestTextDescription;
  tsvTsnComm2 :
  .NoCartridgeDataPresentNotificationTextDescription;
  tsvTsnComm3 :
  .NoCartridgeDataPresentAcknowledgementTextDescription;
  tsvTsnComm4 :
  .MainScreenDisplay_1TextDescription
end

No Cartridge Data Present Notification Event Instantiation
NoCartridgeDataPresentNotification in CommunicationEvent, Token with interaction_type interactionType : "IP"
  sequence_no
  sequenceNo : 1
end

NoCartridgeDataPresentNotificationSSN in SequenceNumber, Token with
  followsFrom : NoCartridgeDataPresentNotificationSSN
  sequenceNo : 6
end

NoCartridgeDataPresentAcknowledgementSSN in SequenceNumber, Token with
  sequenceNo : 7
end

MainScreenDisplay_1_NoDataOnCartridgeSSN in SequenceNumber, Token with
  sequenceNo : 8
end

CheckCartridge forCartridgeRequest_EC_ND_ISN in SequenceNumber, Token with
  sequenceNo : 2
end

ConfirmCartridge forCartridgeRequest_EC_ND_ISN in SequenceNumber, Token with
  sequenceNo : 3
end

NoCartridgeDataPresentNotificationEventTextDescription in CommunicationEvent, Token with
  communication_Description : NoCartridgeDataPresentNotificationEventTextDescription
end

NoCartridgeDataPresentNotificationEventTextDescription in ScenarioEventNaturalLanguageStructure, Token with
  mnis_composite
    mnisComposite1 : NCDPNEC1;
Tracing Development & Assessment Artifacts

mnlsComposite2 : NCDPNEC2;
mnlsComposite3 : NCDPNEC3

mnlsPlainText
    mnlsPlainText1 : NCDPNEPT1;
mnlsPlainText2 : NCDPNEPT2;
mnlsPlainText3 : NCDPNEPT3;
mnlsPlainText4 : NCDPNEPT4

NCDPNEC1 in MatraNLSComposite, Token with preceding_fragment
    precedingFragment : NCDPNEPT1

subject_node
    subjectNode : MPSInstance

following_fragment
    followingFragment : NCDPNEC2

NCDPNEC2 in MatraNLSComposite, Token with preceding_fragment
    precedingFragment : NCDPNEPT2

subject_node
    subjectNode : PilotInstance

following_fragment
    followingFragment : NCDPNEC3

NCDPNEC3 in MatraNLSComposite, Token with preceding_fragment
    precedingFragment : NCDPNEPT3

subject_node
    subjectNode : PilotInstance

following_fragment
    followingFragment : NCDPNEC3

NCDPNEPT1 in PlainTextNode, Token with
    mnls_text
    mnlsText : "The "

end

NCDPNEPT2 in PlainTextNode, Token with
    mnls_text
    mnlsText : " informs the "

end

NCDPNEPT3 in PlainTextNode, Token with
    mnls_text
    mnlsText : " that there is "

end

NCDPNEPT4 in PlainTextNode, Token with
    mnls_text
    mnlsText : " ."

end

NoCartridgeDataNotification in Message, Token with
    message_name
    messageName : "No Cartridge Data Notification"

    tsn_msgParameter
    tsnMsgParameter : VNoCartridgeDataNotificationTsnParameter

    msgParameter : VNoCartridgeDataNotificationMscParameter

end

NoCartridgeDataNotification in MessageDescription, Token with
    msgParameter
    msgParameter : "Report No Data on Cartridge"

end

NoCartridgeDataPresentNotification in MscCommunication, Token with
    link_name
    linkName : "unspecified"

end

NoCartridgeDataPresentNotification in MessageDescription, Token with
    interaction_type
    interactionType : "IP"

end

NoCartridgeDataPresentNotification in MessageDescription, Token with
    communication_name
    communicationName : "unspecified"

end

NoCartridgeDataPresentAcknowledgement in CommunicationEvent, Token with
    interaction_type
    interactionType : "IP"

end

NoCartridgeDataPresentAcknowledgement in CommunicationEvent, Token with
    communication_name
    communicationName : "unspecified"

end

NoCartridgeDataPresentAcknowledgement in CommunicationEvent, Token with
    communication_name
    communicationName : "unspecified"

end

NoCartridgeDataPresentAcknowledgement in CommunicationEvent, Token with
    communication_name
    communicationName : "unspecified"

end

NoCartridgeDataPresentAcknowledgement in CommunicationEvent, Token with
    communication_name
    communicationName : "unspecified"

end

NoCartridgeDataPresentAcknowledgement in CommunicationEvent, Token with
    communication_name
    communicationName : "unspecified"

end

NoCartridgeDataPresentAcknowledgement in CommunicationEvent, Token with
    communication_name
    communicationName : "unspecified"

end

NoCartridgeDataPresentAcknowledgement in CommunicationEvent, Token with
    communication_name
    communicationName : "unspecified"

end

NoCartridgeDataPresentAcknowledgement in CommunicationEvent, Token with
    communication_name
    communicationName : "unspecified"

end
6.2.2.2.4 Erase Cartridge - Pilot Chooses Not to Erase Data


2. C[IR] The [es MPS] asks the [es Cartridge] whether there [es is a cartridge present in the hardware? : Request Confirm Cartridge].

3. C[IP] The [es Cartridge] responds to the [es MPS] that [es there is a cartridge present in the hardware : Confirm Cartridge].

4. C[IR] The [es MPS] asks the [es Cartridge] whether there [es is data on the cartridge? : Request Confirm Cartridge Data].

5. C[IP] The [es Cartridge] responds to the [es MPS] that [es there is data on the cartridge : Confirm Cartridge Data].

6. C[IP] The [es MPS] informs the [es Pilot] that [es there is data on the cartridge : Cartridge Data Found Notification].

7. C[IP] The [es Pilot] indicates to the [es MPS] that he wishes to [es cancel the Erase Cartridge request : Cancel Erase Cartridge].

Instantiation of Erase Cartridge: Pilot Chooses Not to Erase Data

EraseCartridge_PilotChoosesNotToEraseData in Scenario, Token with
scenario_title = "Erase Cartridge - Pilot Chooses Not to Erase Data"
is_exception = True
scenario_event1 = EraseCartridgeRequest;
scenario_event2 = ConfirmDataPresentOnCartridge;
scenario_event3 = CancelEraseRequest;
scenario_event4 = MainScreenDisplay_1
includes_scenario = includes_scenario1
  includes_scenario1 = CheckCartridgeRequest_NormalPath;
  includes_scenario2 = CheckCartridgeForDataRequest_NormalPath
includes_event1 = EraseCartridgeRequest;
includes_event2 = ConfirmCartridgePresent;
includes_event3 = CheckCartridgeForDataRequest;
includes_event4 = ConfirmCartridgeDataPresent

scn_seq_no EraseCartridgeRequest_PilotChoosesNotToEraseDataSSN;
scnSeqNo1 = EraseCartridgeRequest_PilotChoosesNotToEraseDataSSN;
scnSeqNo2 = ConfirmCartridgeForDataRequest_PilotChoosesNotToEraseDataSSN;
scnSeqNo3 = CancelEraseRequestSSN;
scnSeqNo4 = MainScreenDisplay_1_PilotChoosesNotToEraseDataSSN

scn_included_seq_no
scnIncludedSeqNo1 = CheckCartridgeRequest_EC_PCNTED_ISN;
scnIncludedSeqNo2 = ConfirmCartridgePresent_EC_PCNTED_ISN;
scnIncludedSeqNo3 = CheckCartridgeForDataRequest_EC_PCNTED_ISN;
scnIncludedSeqNo4 = ConfirmCartridgeDataPresent_EC_PCNTED_ISN

tsn_viewpoint
  tsnViewpoint = EraseCartridge_PilotChoosesNotToEraseDataTSV
msc_viewpoint
  mscViewpoint = EraseCartridge_PilotChoosesNotToEraseDataMSV
d -- again 'included_event' may be instantiated using a variation of the
  rule in App.A, Pt.2 (vii.c); see also
  App.C, Pt.1 for 0-Telos implementation of included_event classes
EraseCartridgeRequest_PilotChoosesNotToEraseDataSSN in SequenceNumber, Token with
  sequence_no : 1
end
ConfirmCartridgePresentOnCartridge_PilotChoosesNotToEraseDataSSN in SequenceNumber, Token with
  sequence_no : 6
end
CancelEraseRequestSSN in SequenceNumber, Token with
  sequence_no
Case Study I: A Hypothetical Mission Planning System for the Hawk 100 and 200 Series Aircraft

sequenceNo: 7
end

MainScreenDisplay_1_PilotChoosesNotToEraseDataSSN in SequenceNumber, Token with sequence_no; sequenceNo: 8
end

CheckCartridgeRequest_EC_PCNTED_ISN in SequenceNumber, Token with sequence_no; sequenceNo: 2
end

ConfirmCartridgePresent_EC_PCNTED_ISN in SequenceNumber, Token with sequence_no; sequenceNo: 3
end

CheckCartridgeRequest_EC_PCNTED_ISN in SequenceNumber, Token with sequence_no; sequenceNo: 4
end

ConfirmCartridgeDataPresent_EC_PCNTED_ISN in SequenceNumber, Token with sequence_no; sequenceNo: 5
end

EraseCartridge_PilotChoosesNotToEraseDataTSV in TsnScenarioViewpoint, Token with tsv_tsn_comm:

EraseCartridgeRequestTextDescription: tsvTsnComm1:
EraseCartridgeRequestMSCDescription:
EraseCartridgeRequestTextDescription: tsvTsnComm2:
ConfirmDataPresentonCartridgeTextDescription:
ConfirmDataPresentonCartridgeMSCDescription:
CancelEraseRequestTextDescription:
CancelEraseRequestMSCDescription:
MainScreenDisplay_1TextDescription end

EraseCartridge_PilotChoosesNotToEraseDataMSV in MscScenarioViewpoint, Token with msv_msc_comm:

EraseCartridgeRequestMSCDescription:
EraseCartridgeRequestMSCDescription:
EraseCartridgeRequestMSCDescription:
CancelEraseRequestMSCDescription:
CancelEraseRequestMSCDescription:
MainScreenDisplay_1MSCDescription end

Cancel Erase Request Event Instantiation

CancelEraseRequest in CommunicationEvent, Token with interaction_type: "IP":
sequence_no: sequenceNo: CancelEraseRequestSSN follows_from:
followsFrom:
ConfirmDataPresentonCartridge
CancelEraseRequestTextDescription:
CancelEraseRequestMSCDescription end

CancelEraseRequestTextDescription in TsnCommunication, Token with communication_description:
CancelEraseRequestEventText end

CancelEraseRequestEventText in ScenarioEventNaturalLanguageStructure, Token with
mlns_composite:
mlnsComposite1: CEREC1;
mlnsComposite2: CEREC2;
mlnsComposite3: CEREC3
mlns_plain_text:
mlnsPlainText1: CEREPT1;
mlnsPlainText2: CEREPT2;
mlnsPlainText3: CEREPT3;
mlnsPlainText4: CEREPT4
tsn_sender_node:
tsnSenderNode: Pilotlnstance
tsn_receiver_node:
tsnReceiverNode: MPSInstance
tsn_message_node:
tsnMessageNode:
CancelEraseCartridge end

CEREC1 in MatraNLSComposite, Token with preceding_fragment:
subject_node:
subjectNode: PilotInstance
following_fragment:
followingFragment: CEREC2 end

CEREC2 in MatraNLSComposite, Token with preceding_fragment:
subject_node:
subjectNode: MPSInstance
following_fragment:
followingFragment: CEREC3 end

CEREC3 in MatraNLSComposite, Token with preceding_fragment:
subject_node:
subjectNode: CancelEraseCartridge
following_fragment:
followingFragment: CEREP4 end

CEREPT1 in PlainTextNode, Token with mlns_text:
mmlsText: "The " end

CEREPT2 in PlainTextNode, Token with mlns_text:
mmlsText: " indicates to the " end

CEREPT3 in PlainTextNode, Token with mlns_text:
mmlsText: " that he wishes to " end

CEREPT4 in PlainTextNode, Token with mlns_text:
mmlsText: "." end

CancelEraseCartridge in Message, Token with message_name:
messageName: "Cancel Erase Cartridge"
tsnt_msg_parameter
6.2.2.2.5 Retrieve from Cartridge - Normal Path (Timing Fragment)

From Retrieve from Cartridge (Normal Path), we consider the following additional scenario fragment to demonstrate instantiation of timer-set, time-out and internal action events (the full logical level representation appears in Appendix C, Part Two).

11. \(T\) The \(\text{[Reset set MPS]}\) sets the \(\text{[Start DataTimer]}\) to \(\text{[Duration 10 seconds]}\).

12. \(\text{C}[SP]\) The \(\text{[MPS]}\) informs the \(\text{[Pilot]}\) that \(\text{[msg_type_1 data has been retrieved - Type-1 Retrieved Notification]}\).

13. \(T\) The \(\text{[Stop-on-timeout MPS]}\) \(\text{[Timer DataTimer]}\) times-out.

14. \(A\) The \(\text{[Pilot MPS]}\) \(\text{[msg stores type_1 data - Store Type_1 Data]}\).

---

**Figure 6.10 - ‘Retrieve from Cartridge - Normal Path (Timing Fragment)’**
Case Study I: A Hypothetical Mission Planning System for the Hawk 100 and 200 Series Aircraft

`SDLPT3; SDLPT4` -

 Initialize Timer Set Node:

\[tsn\text{_timer\_set\_node}\]

`tsn\text{_timer\_instance\_node}` -

`DataTimer` -

\[tsn\text{_timer\_duration}\]

\[SetDataTimer1Duration\]

\[SDLPT1\text{ in MatraNLSComposite, Token with
preceding\_fragment}\]

\[subject\_node\]

\[subject\_node\]

\[SetDataTimer1Duration\]

\[SDLPT2\text{ in MatraNLSComposite, Token with
preceding\_fragment}\]

\[subject\_node\]

\[subject\_node\]

\[SetDataTimer1Duration\]

\[SDLPT3\text{ in MatraNLSComposite, Token with
preceding\_fragment}\]

\[subject\_node\]

\[subject\_node\]

\[SetDataTimer1Duration\]

\[SDLPT4\text{ in MatraNLSComposite, Token with
preceding\_fragment}\]

\[subject\_node\]

\[subject\_node\]

\[SetDataTimer1Duration\]

\[DataTimer\text{ in Timer, Token with
timer\_name}\]

\[timer\_name\]

\[timer\_duration\]

\[SetDataTimer1Duration\]

\[SetDataTimer1Duration\text{ in TimerDuration,
Token with
duration}\]

\[duration\]

\[SetDataTimer1Duration\]

\[DataTimer\text{ in Timer, Token with
msc\_timer\_set\_instance}\]

\[msc\_timer\_set\_instance\]

\[msc\_timer\_instance\]

\[msc\_timer\_instance\]

\[msc\_timer\_duration\]

\[SetDataTimer1Duration\]

\[RTINEC1\text{ in MatraNLSComposite, Token with
preceding\_fragment}\]

\[subject\_node\]

\[subject\_node\]

\[SetDataTimer1Duration\]

\[RTINEC2\text{ in MatraNLSComposite, Token with
preceding\_fragment}\]

\[subject\_node\]

\[subject\_node\]

\[SetDataTimer1Duration\]

\[RTINEC3\text{ in MatraNLSComposite, Token with
preceding\_fragment}\]

\[subject\_node\]

\[subject\_node\]

\[SetDataTimer1Duration\]

\[Type\text{\_1 (Data) Retrieved Notification
(Event \# 12) Instantiation}\]

\[SDLPT3; SDLPT4\]

\[tsn\text{\_timer\_set\_node}\]

\[tsn\text{\_timer\_instance\_node}\]

\[tsn\text{\_timer\_duration}\]

\[SetDataTimer1Duration\]

\[SDLPT1\text{ in MatraNLSComposite, Token with
preceding\_fragment}\]

\[subject\_node\]

\[subject\_node\]

\[SetDataTimer1Duration\]

\[SDLPT2\text{ in MatraNLSComposite, Token with
preceding\_fragment}\]

\[subject\_node\]

\[subject\_node\]

\[SetDataTimer1Duration\]

\[SDLPT3\text{ in MatraNLSComposite, Token with
preceding\_fragment}\]

\[subject\_node\]

\[subject\_node\]

\[SetDataTimer1Duration\]

\[SDLPT4\text{ in MatraNLSComposite, Token with
preceding\_fragment}\]

\[subject\_node\]

\[subject\_node\]

\[SetDataTimer1Duration\]
Tracing Development & Assessment Artifacts

TimeoutDataTimer1EventText in ScenarioEventNaturalLanguageStructure, Token with
mls_composite
  mlsComposite1 : TODTIEC1;
  mlsComposite2 : TODTIEC2
mls_plain_text
  mlsPlainText1 : TODTIEPT1;
  mlsPlainText2 : TODTIEPT2
mls_null
  mlsNull : TODT1ENull
tsn_host_on_timeout_node
  tsnHostOnTimeoutNode : MPSInstance
tsn_timer_instance_node
  tsnTimerInstanceNode : DataTimer
end

TODTIEC1 in MatraNLSComposite, Token with preceding_fragment
  precedingFragment : TODTIEPT1
subject_node
  subjectNode : MPSInstance
following_fragment
  followingFragment : TODTIEPT2
end

TODTIEC2 in MatraNLSComposite, Token with preceding_fragment
  precedingFragment : TODTIEPT1
subject_node
  subjectNode : MPSInstance
following_fragment
  followingFragment : TODTIEPT2
end

StorageType1 in InternalActionEvent, Token with
......
  tsn_action_event
    tsnActionEvent :
      StorageType1TextDescription
        msc_action_event
          mscActionEvent :
            StorageType1MSCDescription
          msc_host_on_timeout_instance
            mscHostOnTimeoutInstance :
              MPSInstance
            msc_timer_instance
              mscTimerInstance : DataTimer
          end
      StorageType1EventText
        msc_timing_event
          mscTimingEvent :
            StorageType1TextDescriptionInTsnAction
              action_description
                actionDescription :
                  StorageType1EventText
              end
        StorageType1EventText
          mls_composite
            mlsComposite1 : STIEC1;
            mlsComposite2 : STIEC2
6.2.2.2.6 Choose Mission and Aircraft (Event Group Fragment)

Finally, from Choose Mission and Aircraft (New Mission from Open Missions), we consider the following fragment to demonstrate elements for event grouping and upper and lower bounds on iteration.

```
ITERATION: Lower Bound = 1: Upper Bound = card data types Mission Piano

13. A The [auth MPS] [act Stores the data item for the selected Mission and Aircraft: Store Selected MissionAC Data].
```

Figure 6.11 - ‘Choose Mission and Aircraft - New Mission from Open Missions (Event Group Fragment)’
Data Item Request (Event # 11) Instantiation

DataItemRequest in CommunicationEvent, Token with interaction_type : "IR"

......

DataItemRequestTextDescription in TsnCommunication, Token with
communication_description :
DataItemRequestEventText end

DataItemRequestEventText in ScenarioEventNaturalLanguageStructure, Token with
mlns_composite
mlnsComposite1 : DIREC1;
mlnsComposite2 : DIREC2;
mlnsComposite3 : DIREC3
mlns_plain_text
mlnsPlainText1 : DIREPT1;
mlnsPlainText2 : DIREPT2;
mlnsPlainText3 : DIREPT3;
mlnsPlainText4 : DIREPT4
tsn_sender_node
  tsnSenderNode : MPSInstance
tsn_receiver_node
  tsnReceiverNode : MissionPlanInstance
tsn_message_node
  tsnMessageNode : RequestDataltem end

DIREC1 in MatraNLSComposite, Token with preceding_fragment
  precedingFragment : DIREPT1
subject_node
  subjectNode : MPSInstance
following_fragment
  followingFragment : DIREC2 end

DIREC2 in MatraNLSComposite, Token with preceding_fragment
  precedingFragment : DIREPT2
subject_node
  subjectNode : MissionPlanInstance
following_fragment
  followingFragment : DIREC3 end

DIREC3 in MatraNLSComposite, Token with preceding_fragment
  precedingFragment : DIREPT3
subject_node
  subjectNode : RequestDataltem
following_fragment
  followingFragment : DIREPT4 end

DIREPT1 in PlainTextNode, Token with
  mlns_text
    mlnsText : "The "
end

DIREPT2 in PlainTextNode, Token with
  mlns_text
    mlnsText : " asks the "
end

Mission And Aircraft Data Item Provision (Event #12) Instantiation

MACDataItemProvision in CommunicationEvent, Token with interaction_type : "IP"

......

MACDataItemProvisionTextDescription in TsnCommunication, Token with
communication_description :
MACDataItemProvisionEventText end

MACDataItemProvisionEventText in ScenarioEventNaturalLanguageStructure, Token with
mlns_composite
mlnsComposite1 : MACDIPEC1;
mlnsComposite2 : MACDIPEC2;
mlnsComposite3 : MACDIPEC3
mlns_plain_text
Case Study I: A Hypothetical Mission Planning System for the Hawk 100 and 200 Series Aircraft

MACDIPEC1 in MatraNLSComposite, Token
with
preceding_fragment
precedingFragment = MACDIPEPT1
subject_node
subjectNode = MissionPlanInstance
following_fragment
followingFragment = MACDIPEC2
end

MACDIPEC2 in MatraNLSComposite, Token
with
preceding_fragment
precedingFragment = MACDIPEPT2
subject_node
subjectNode = MissionPlanInstance
following_fragment
followingFragment = MACDIPEC3
end

MACDIPEC3 in MatraNLSComposite, Token
with
preceding_fragment
precedingFragment = MACDIPEPT3
subject_node
subjectNode = MissionPlanInstance
following_fragment
followingFragment = MACDIPEC4
end

MACDIPEPT1 in PlainTextNode, Token with
mls_text
mlsText = "The "
end

MACDIPEPT2 in PlainTextNode, Token with
mls_text
mlsText = "supplies the "
end

MACDIPEPT3 in PlainTextNode, Token with
mls_text
mlsText = "to the "
end

MACDIPEPT4 in PlainTextNode, Token with
mls_text
mlsText = ".
end

Mission&ACDataItem in Message, Token with
message_name
messageName = "Mission&AC Data Item"
msg_parameter
tsnMsgParameter = Mission&ACDataItemTsnParameter
mscMsgParameter = Mission&ACDataItemMscParameter
end

Mission&ACDataItemTsnParameter in MessageDescription, Token with
msg_parameter
msgParameter = "data item for the selected Mission and Aircraft"
end

Mission&ACDataItemMscParameter in MessageDescription, Token with
msg_parameter
msgParameter = "data item for Selection"
end

MACDataItem ProvisionMSCDescription in MscCommunication, Token with
link_name
linkName = "unspecified"
synchronisation
synchronisation = "sim"
frequency
frequency = "aperiodic"
delayed
delayed = False
msc_sender_instance
mscSenderInstance = MissionPlanInstance
msc_receiver_instance
mscReceiverInstance = MPSInstance
msc_message
mscMessage = Mission&ACDataItem
end

Storage Mission and Aircraft Data (Event #13) Instantiation

StorageMAC in InternalActionEvent, Token with
......

Mission&ACDataItemTsnParameter in MessageDescription, Token with
msg_parameter
msgParameter = "data item for the selected Mission and Aircraft"
end

Mission&ACDataItemMscParameter in MessageDescription, Token with
msg_parameter
msgParameter = "data item for Selection"
end

MACDataItem ProvisionMSCDescription in MscCommunication, Token with
link_name
linkName = "unspecified"
synchronisation
synchronisation = "sim"
frequency
frequency = "aperiodic"
delayed
delayed = False
msc_sender_instance
mscSenderInstance = MissionPlanInstance
msc_receiver_instance
mscReceiverInstance = MPSInstance
msc_message
mscMessage = Mission&ACDataItem
end

Storage Mission and Aircraft Data (Event #13) Instantiation

StorageMAC in InternalActionEvent, Token with
......

Mission&ACDataItemTsnParameter in MessageDescription, Token with
msg_parameter
msgParameter = "data item for the selected Mission and Aircraft"
end

Mission&ACDataItemMscParameter in MessageDescription, Token with
msg_parameter
msgParameter = "data item for Selection"
end

MACDataItem ProvisionMSCDescription in MscCommunication, Token with
link_name
linkName = "unspecified"
synchronisation
synchronisation = "sim"
frequency
frequency = "aperiodic"
delayed
delayed = False
msc_sender_instance
mscSenderInstance = MissionPlanInstance
msc_receiver_instance
mscReceiverInstance = MPSInstance
msc_message
mscMessage = Mission&ACDataItem
end

Storage Mission and Aircraft Data (Event #13) Instantiation

StorageMAC in InternalActionEvent, Token with
......

Mission&ACDataItemTsnParameter in MessageDescription, Token with
msg_parameter
msgParameter = "data item for the selected Mission and Aircraft"
end

Mission&ACDataItemMscParameter in MessageDescription, Token with
msg_parameter
msgParameter = "data item for Selection"
end

MACDataItem ProvisionMSCDescription in MscCommunication, Token with
link_name
linkName = "unspecified"
synchronisation
synchronisation = "sim"
frequency
frequency = "aperiodic"
delayed
delayed = False
msc_sender_instance
mscSenderInstance = MissionPlanInstance
msc_receiver_instance
mscReceiverInstance = MPSInstance
msc_message
mscMessage = Mission&ACDataItem
end

Storage Mission and Aircraft Data (Event #13) Instantiation

StorageMAC in InternalActionEvent, Token with
......
subjectNode : StoreSelectedMission&ACData
following_fragment
  followingFragment : SMACEPT2
end

SMACEPT1 in PlainTextNode, Token with mnls_text
  mnlsText : "The "
end

SMACEPT2 in PlainTextNode, Token with mnls_text
  mnlsText : "."
end

StoreSelectedMission&ACData in Action, Token
  action_name
    actionName : "Store Selected Mission and Aircraft Data"
  tsn_act.parameter
    tsnActParameter
      stores the data item for the selected Mission and Aircraft"
end

StoreSelectedMission&ACData in MessageDescription, Token
  actionParameter
    "stores the data item for the selected Mission and Aircraft"
end

6.2.3 Trace Relations

This subsection will demonstrate how, given the meta-model representations in subsection 6.2.2.1.1 and in subsections 6.2.2.2.1 through 6.2.2.2.6, the MATra workspace is able to support traceability of user centred requirements artifacts within (in this case) the intra-micro horizontal dimension. In doing so, we instantiate a subset of trace relations introduced in subsection 3.3.6.3.2. These relations are illustrated at a logical level in figure 6.12 (and summarised in table 6.2).

It should be noted that the featured examples are fairly arbitrary. Indeed, as Chapter Three indicated, a thorough investigation (combining literature review and practitioner consultation) is deemed necessary to develop a comprehensive set of associations for linking different artifact types used throughout the aerospace industry; this is earmarked as a future work item in subsection 7.4.3. The investigation would need to consider a wide variety of notations and techniques as potential source and targets of these associations - including requirements, design, implementation, safety assessment and product management. Those for requirements could potentially build on the taxonomy proposed by project NATURE (Pohl, 1996) referred to in subsection 3.2.1.
Case Study I: A Hypothetical Mission Planning System for the Hawk 100 and 200 Series Aircraft

Scenario Name: Erase Cartridge - Normal Path

1. C(JR) The MPS selects the Erase Cartridge option on the MPS Main Screen.
2. C(IR) The MPS asks the Cartridge whether there is a cartridge present in the hardware.
3. C(JP) The Cartridge responds to the MPS that there is a cartridge present in the hardware.
4. C(IR) The MPS asks the Cartridge whether there is data on the cartridge.
5. C(JP) The Cartridge responds to the MPS that there is data on the Cartridge.
6. C(JP) The MPS informs the Pilot that there is data on the Cartridge.

Figure 6.12 - 'Exemplar Intra-Micro Horizontal Traceability Relations'

Table 6.2 - 'Summary of Traceability Relations' (from figure 6.12)
6.2.3.1 Instantiation of Trace Relations

We now demonstrate application of these relations by instantiating the appropriate O-Telos base classes introduced in Chapter Three (subsection 3.3.6.3.2).

Our first example populates the PathThrough relation; its source (fromEntity) is a textual scenario for ‘Erase Cartridge - Normal Path’ (EraseCartridge_NormalPathTSV) and its destination (toEntity), the Erase Cartridge (EraseCartridge) use case of the Erase Cartridge use case model (EraseCartridgeModel).

```
PathThroughBy01 in PathThrough, Token with
  from_entity
    fromEntity : EraseCartridge_NormalPathTSV
  to_entity
    toEntity : EraseCartridge
```

Example two populates IsARoleOf, in this case between the Pilot (sender instance) of a textual scenario and the cognate Actor element from the Erase Cartridge use case model (EraseCartridgeModel).

```
IsARoleOf01 in IsARoleOf, Token with
  from_entity
    fromEntity : PilotInstance
  to_entity
    toEntity : Pilot
```

Our third example populates the IllustratedBy association. This particular relationship exists between corresponding textual and sequence diagram representations of the same event.

```
IllustratedBy01 in IllustratedBy, Token with
  from_entity
    fromEntity : ConfirmDataPresentonCartridgeTextDescription
  to_entity
    toEntity : ConfirmDataPresentonCartridgeMSCDescription
```

Our final example populates the HasCommonInstance association linking two scenarios, ‘Erase Cartridge - Normal Path’ and ‘New Mission from Open Missions’, both of which include Pilot and MPS instances (one such event for the ‘New Mission from Open Missions’ scenario is highlighted in figure 6.12).

```
HasCommonInstance01 in HasCommonInstance, Token with
  from_entity
    fromEntity : EraseCartridge_NormalPath
  to_entity
    toEntity : NewMissionfromOpenMissions
```

Note the source element of HasCommonInstance01, EraseCartridge_NormalPath, comes from instantiation of the Interaction View in 6.2.2.2.1 (EraseCartridgeModel), while its target NewMissionfromOpenMissions is introduced purely to support the example.

---

2 A corresponding PathThrough association would also exist between the MSC (EraseCartridge_NormalPathMSV) representation and EraseCartridge.
3 Recall that textual and MSC scenario representations share common Instance objects such that figure 6.12 could show a similar logical-level association from the corresponding Sequence Diagram element.
6.2.4 Summary

This case study has demonstrated application of the User Centred Requirements Structure introduced in section 4.3. In doing so, a commercial specification supplied by BAE SYSTEMS was used to populate an O-Telos implementation of (UCRS) base classes. We then illustrated how constituent models and elements of the structure may be linked using various trace relations. The results of this case study will be considered as part of our overall thesis evaluation in Chapter Seven.
6.3 Case Study II: A Brake System Control Unit for a Wheel Braking System of a Hypothetical Aircraft

This section presents a case study based on extracts from a contiguous example in ARP 4761 (Appendix L). We use a subset of artifacts featured in that example describing assessment of a hypothetical aircraft - the S18 - to demonstrate aspects of the Fault Tree and Failure Modes and Effects Analysis structures introduced in sections 5.2 and 5.3. We also show how elements of these structures may be linked using trace relations, again based on the approach shown in 3.3.6.3.2.

6.3.1 Scope of Case Study

The case study concentrates on a subset of activities from the ARP 4761 assessment process and in doing so, on one particular component (item) of the S18 aircraft. Specifically, we consider identification and verification of quantitative (safety) requirements for the Brake System Control Unit (BSCU) computer, a sub-module of the S18 Wheel Braking System (WBS). Requirements identification forms part of Preliminary System Safety Assessment (PSSA) of the BSCU and verification, part of System Safety Assessment (SSA). This is placed in context by the following diagram fragment (figure 6.13); note readers are referred back to figure 1.6 in subsection 1.4.6.2.2 for the complete diagram and a description of the ARP 4761 assessment process.

![Figure 6.13 - 'ARP Assessment Process (Partial)'](image)

6.3.2 Overview of S18 Wheel Braking System and Brake System Control Unit

The Wheel Braking System is installed on the main landing gears, its purpose to decelerate aircraft on the ground (without the tyres skidding) during taxi and landing phases and in the event of rejected take-off. Braking is automatically performed on landing, or when manually activated by the pilot.
Each wheel brake assembly is operated by two independent sets of hydraulic pistons. One set operates from the Green hydraulic supply and is used in Normal Braking mode. The Alternate mode which uses the Blue hydraulic supply is on standby and is selected automatically when the Normal system fails. An Emergency braking mode is also available, although designers take the decision that all safety requirements shall be met without regard to this system. In the Normal mode, all wheels are individually braked from their own servo valves according to commands received from one of two Braking System Control Unit computers - normally BSCU1 or, if system 1 reports a failure, BSCU2.

Failure conditions relating to this function include 'Loss of all Wheel Braking' (shown by the fault tree in figure 5.1, subsection 5.2.2.1) and 'Inadvertent Wheel Braking' - the focus of this case study. Such events are established by a (Braking System) Functional Hazard Assessment (FHA), with budget probabilities allocated according to severity. These requirements provide inputs to the (Braking System) PSSA; to determine causality, catastrophic and hazardous conditions form top events of fault trees (an example of this relationship is shown at aircraft level in figure 1.5 of Chapter One). In turn, basic events of the system PSSA trees derive requirements that feed into an item level PSSA. This is the point at which we join the example from ARP 4761.

6.3.3 Preliminary System Safety Assessment - Brake System Control Unit

For the purpose of this case study, we assume a fault tree (not shown) developed as part of the Wheel Braking System PSSA has a basic event imposing the following item level requirement over the component BSCU:

- The probability of 'BSCU commands braking in absence of braking commands and causes inadvertent braking' shall be less than 2.5E-9 per flight.

Preliminary System Safety Assessment of the BSCU subsequently serves to complete safety requirements for this particular component. These include quantitative requirements, again derived using fault trees - one of which will now be used to demonstrate instantiation of 'preliminary' elements of the Fault Tree Analysis structure.

6.3.3.1 Background on BSCU Design

The BSCU design consists of two independent systems (referred to previously as BSCUI and BSCU2). Both generate necessary voltages from their respective power supplies; out of specification voltages are detected within each system by a power supply monitor (described in 6.3.4.1).

According to the design proposal considered here, each system will also contain a command and monitor channel which computes braking commands based on brake pedal inputs. The commands generated by each channel are compared, with a failure reported when a disparity is detected.

Results of the power supply monitor and comparator are then provided to a system validity monitor; a failure reported by either BSCU causes that system to disable its outputs and set the system validity
monitor to invalid. Each system validity monitor is provided to an overall BSCU validity monitor; failure of both BSCU1 and BSCU2 causes the selector valve to switch to the Alternate Braking System.

In normal operation, BSCU1 provides the braking commands. When a failure is reported (via its system validity monitor), the output of BSCU2 (if valid) is switched in to allow braking. If system 2 also fails, all BSCU outputs are disabled and the BSCU validity monitor set to invalid.

6.3.3.2 Fault Tree Analysis - Preliminary
The fault tree in figures 6.14i to 6.14iii considers inadvertent wheel braking attributable to the BSCU (with requirement 2.5E-9 per flight hour as stated in subsection 6.3.3) to determine feasibility of the design outlined above. In the interests of readability, transfer-in/transfer-out symbols (denoted by a triangle with a horizontal line from its side) are employed allowing the two main fault tree branches to be shown as separate figures.

In addition to safety requirements from the previous assessment phase, PSSA also takes into account operational considerations. Those relevant to this analysis include average flight length (5 hours), average power-up interval (100 hours), estimated aircraft life (100,000 hours) and time between V1 and VR (0.004167 hour)\(^4\). These provide exposure times and are used (together with failure rates based on design data and service experience) in calculating probabilities for basic fault tree events.

It can be seen from figure 6.14i that the tree assumes no undetected BSCU failures can lead to inadvertent braking (BSCUUNDF); such an assumption must be proved correct through Failure Modes and Effects Analysis and/or Common Mode Analysis. The other branch of the fault tree (developed in figures 6.14ii and 6.14iii) addresses combinations of monitored BSCU failures and monitor failures\(^5\).

The fault tree in 6.14ii describes analysis of the BSCU1 detectable failures causing bad data event (BSCUIDETD). In summary, this event can occur if the power supply monitor is stuck valid (BS1PSMOFV) and the power supply failure causes bad data (BSCU1PSF). An alternative path to the event is if the monitor channel always reports valid due to hardware failure (BSCUIMOFV) and command channel I/O failure causes bad data (BSCU1I/O) or, command channel CPU hardware failure causes bad data (BSCU1CPUF).

The fault tree in figure 6.14iii contains the same failure information relative to BSCU2. However, this system is only applied to the brakes upon activation of the monitor/select switch. Thus the event of the switch being in position 2 (SWITCH2) is And-ed with BSCU2 failures. Two conditions may result in switch activation, a previously detected failure of BSCU1 (BSCU1TF), or an independent monitor switch failure (SWIFAIL2).

\(^4\) V1 is the take off decision speed at which it is possible to take off safely following engine failure. It is also the speed whereby, if take off were abandoned, then the aircraft may be brought to a safe stop within the remaining runway. VR is the speed where the aircraft begins to rotate to lift off attitude and climb.

\(^5\) Note event labels in figures 6.14i-6.14iii contain the subject entity (preceded by an optional qualifying entity enclosed in square brackets) and a condition, separated by a period.
Case Study II: A Brake System Control Unit for a Wheel Braking System of a Hypothetical Aircraft

Figure 6.14i - 'BSCU Commands Braking in Absence of Brake Input Causing Inadvertent Braking - Preliminary Fault Tree (page 1)'

Figure 6.14ii - 'BSCU Commands Braking in Absence of Brake Input Causing Inadvertent Braking - Preliminary Fault Tree (page 2)'

320
6.3.3.2.1 Instantiation of Fault Tree Analysis Meta-model - Preliminary Fault Tree

In the following, we instantiate an O-Telos implementation of the Fault Tree Analysis meta-model from section 5.2 with failure behaviours shown in figure 6.14. At this stage, we instantiate the preliminary tree and preliminary event profiles only. However, subsection 6.3.4.3 revisits the meta-model with respect to the BSCU System Safety Assessment and in doing so, completes population of the updated tree and updated event profiles.

**Definition of Event ‘BSCUINADD’**

EventA in Event, Token with identifier

Identifier : "BSCUINADD"
preliminary_profile
preliminaryProfile : EventAPreliminaryProfile
end

EventAPreliminaryProfile in PreliminaryEventProfile, Token with type

_type : "top"
event_connective
eventConnective : OrGate1P
preliminary_budget
preliminaryBudget : EventAPreliminaryBudget
preliminary_label
preliminaryLabel :

---

Figure 6.14iii - ‘BSCU Commands Braking in Absence of Brake Input Causing Inadvertent Braking - Preliminary Fault Tree (page 3)’

---

EventAPreliminaryLabel end

EventAPreliminaryBudget in BudgetProbability, Token with probability

_Probability : 2.50E-09
annotation

_Annotation : EventAPreliminaryBudgetMNLS
end

EventAPreliminaryBudgetMNLS in MatraNaturalLanguageStructure, Token with

_mnls_plain_text

_mnlsPlainText1 : EventAPBPT1
end

EventAPBPT1 in PlainTextNode, Token with
Case Study II: A Brake System Control Unit for a Wheel Braking System of a Hypothetical Aircraft

Requirement is per 5 hour flight.

Gate definitions

EventA

Definition of Event 'BSCU UNDF'

EventB

Definition of Event 'BSCU DETD'

Gate definitions

EventC

Definition of Event 'BSCU DETFD'

EventD
Tracing Development & Assessment Artifacts

Definition of Event `BSCU1PSIND'

EventF in Event, Token with identifier
  _Identifier : "BSCU1PSIND"
  preliminary_profile
    preliminaryProfile : EventFPreliminaryProfile
  end

EventFPreliminaryProfile in PreliminaryEventProfile, Token with type
  _Type : "int"
  event_connective
    EventConnective : AndGate3P
  preliminary_budget
    preliminaryBudget : EventFPreliminaryBudget
  preliminary_label
    preliminaryLabel : EventFPreliminaryLabel
  end

EventFPreliminaryBudget in BudgetProbability, Token with probability
  _Probability : 1.00E-09
  end

EventFPreliminaryLabel in SimpleLabel, Token with
  simple_description
  simpleDescription : EventFSimpleEventDescription
end

EventFSimpleEventDescription in SimpleEventDescription, Token with entity
  _Entity : "Power_Supply"
  qualifying_entity
    qualifyingEntity : "BSCU1"
  condition
    _Condition : "Power Supply Failure Causes Bad Data"
end

Definition of Event `BSCU1CDIND'

EventG in Event, Token with identifier
  _Identifier : "BSCU1CDIND"
  preliminary_profile
    preliminaryProfile : EventGPreliminaryProfile
  end

EventGPreliminaryProfile in PreliminaryEventProfile, Token with type
  _Type : "int"
  event_connective
    EventConnective : AndGate2P
  preliminary_budget
    preliminaryBudget : EventGPreliminaryBudget
  preliminary_label
    preliminaryLabel : EventGPreliminaryLabel
  end

EventGPreliminaryBudget in BudgetProbability, Token with probability
  _Probability : 1.00E-09
  end

EventGPreliminaryLabel in SimpleLabel, Token with
  simple_description
  simpleDescription : EventGSimpleEventDescription
end

EventGSimpleEventDescription in SimpleEventDescription, Token with entity
  _Entity : "BSCU2"
  condition
    _Condition : "Detectable Failure Causes Bad Data"
end

Gate definitions

OrGate3P in OrGate, Token with input
  _Input1 : EventFPreliminaryProfile;
  _Input2 : EventGPreliminaryProfile
end

preliminary_label
  preliminaryLabel : EventDPreliminaryLabel
end

EventDPreliminaryBudget in BudgetProbability, Token with probability
  _Probability : 2.00E-09
  end

EventDPreliminaryLabel in SimpleLabel, Token with simple_description
  simpleDescription : EventDSimpleEventDescription
end

EventDSimpleEventDescription in SimpleEventDescription, Token with entity
  _Entity : "BSCU1"
  condition
    _Condition : "Detectable Failure Causes Bad Data"
end

Definition of Event `BSCU2DETD'

EventE in Event, Token with identifier
  _Identifier : "BSCU2DETD"
  preliminary_profile
    preliminaryProfile : EventEPreliminaryProfile
  end

EventEPreliminaryProfile in PreliminaryEventProfile, Token with type
  _Type : "int"
  event_connective
    EventConnective : AndGate3P
  preliminary_budget
    preliminaryBudget : EventEPreliminaryBudget
  preliminary_label
    preliminaryLabel : EventEPreliminaryLabel
  end

EventEPreliminaryBudget in BudgetProbability, Token with probability
  _Probability : 5.00E-10
  end

EventEPreliminaryLabel in SimpleLabel, Token with
  simple_description
  simpleDescription : EventESimpleEventDescription
end

EventESimpleEventDescription in SimpleEventDescription, Token with entity
  _Entity : "BSCU2"
  condition
    _Condition : "Detectable Failure Causes Bad Data"
end

Gate definitions

OrGate3P in OrGate, Token with input
  _Input1 : EventFPreliminaryProfile;
  _Input2 : EventGPreliminaryProfile
end

323
Case Study 11: A Brake System Control Unit for a Wheel Braking System of a Hypothetical Aircraft

EventGSimpleEventDescription in SimpleEventGDescription, Token with entity
  _Entity : "Computational_Channel"
  qualifying_entity
    qualifyingEntity : "BSCU1"
condition
  _Condition : "Failure/Error Causes
  Bad Data"
end

Gate definitions

AndGate1P in AndGate, Token with
  _Input : HIEventSetP
end

AndGate2P in AndGate, Token with
  _Input : JKEventSetP
end

HIEventSetP in EventSet, Token with event_profile
eventProfile1 :
  EventHPreliminaryProfile;
eventProfile2 :
  EventHPreliminaryProfile
end

JKEventSetP in EventSet, Token with event_profile
  eventProfile1 :
    EventJPreliminaryProfile;
  eventProfile2 :
    EventKPreliminaryProfile
end

Definition of Event 'BS1PSMOFV'

EventH in Event, Token with identifier
  _Identifier : "BS1PSMOFV"
preliminary_profile
  preliminaryProfile :
    EventHPreliminaryProfile
end

EventHPreliminaryProfile in PreliminaryEventProfile, Token with type
  _Type : "bas"
preliminary_budget
  preliminaryBudget :
    EventHPreliminaryBudget
preliminary_rate
  preliminaryRate :
    EventHPreliminaryRate
preliminary_exposure
  preliminaryExposure :
    EventHPreliminaryExposure
preliminary_label
  preliminaryLabel :
    EventHPreliminaryLabel
end

EventHPreliminaryBudget in BudgetProbability, Token with probability
  _Probability : 2.00E-02
end

EventHPreliminaryRate in Rate, Token with failure_rate
  failureRate : 2.00E-07
end

EventHPreliminaryExposure in Exposure, Token with period
  _Period : 100000.00
end

EventHPreliminaryLabel in SimpleLabel, Token with simple_description
  simpleDescription :
    EventHSimpleEventDescription
end

EventHSimpleEventDescription in SimpleEventDescription, Token with entity
  _Entity : "Power_Supply_Monitor"
  qualifying_entity
    qualifyingEntity : "BSCU1"
condition
  _Condition : "Monitor Stuck Valid"
end

Definition of Event 'BSCU1PSF'

EventI in Event, Token with identifier
  _Identifier : "BSCU1PSF"
preliminary_profile
  preliminaryProfile :
    EventIPreliminaryProfile
end

EventIPreliminaryProfile in PreliminaryEventProfile, Token with type
  _Type : "bas"
preliminary_budget
  preliminaryBudget :
    EventIPreliminaryBudget
preliminary_rate
  preliminaryRate :
    EventIPreliminaryRate
preliminary_exposure
  preliminaryExposure :
    EventIPreliminaryExposure
preliminary_label
  preliminaryLabel :
    EventIPreliminaryLabel
end

EventIPreliminaryBudget in BudgetProbability, Token with probability
  _Probability : 5.00E-08
end

EventIPreliminaryRate in Rate, Token with failure_rate
  failureRate : 1.20E-05
end

EventIPreliminaryExposure in Exposure, Token with period
  _Period : 0.004167
end

EventIPreliminaryLabel in SimpleLabel, Token with simple_description
  simpleDescription :
    EventISimpleEventDescription
end

EventISimpleEventDescription in SimpleEventDescription, Token with entity

324
Definition of Event 'BSCUIMORV'

EventJ in Event, Token with identifier
  _Identifier : "BSCUIMORV"
preliminary_profile
  preliminaryProfile : EventJPreliminaryProfile
end

EventJPreliminaryProfile in PreliminaryEventProfile, Token with type
  _Type : "int"
  event_connective
  EventConnective : OrGate4P
  preliminary_budget
  preliminaryBudget : EventJPreliminaryBudget
  preliminary_label
  preliminaryLabel : EventJPreliminaryLabel
end

EventJPreliminaryBudget in BudgetProbability, Token with probability
  _Probability : 2.00E-02
end

EventJPreliminaryLabel in SimpleLabel, Token with simple_description
  simpleDescription : EventJSimpleEventDescription
end

EventJSimpleEventDescription in SimpleEventDescription, Token with entity
  _Entity : "Monitor_Channel"
  qualifying_entity
  qualifyingEntity : "BSCU1"
  condition
  _Condition : "Always Reports Valid"
end

Definition of Event 'BSCUICDF'

EventK in Event, Token with identifier
  _Identifier : "BSCUICDF"
preliminary_profile
  preliminaryProfile : EventKPreliminaryProfile
end

EventKPreliminaryProfile in PreliminaryEventProfile, Token with type
  _Type : "int"
  event_connective
  EventConnective : OrGate5P
  preliminary_budget
  preliminaryBudget : EventKPreliminaryBudget
  preliminary_label
  preliminaryLabel : EventKPreliminaryLabel
end

EventKPreliminaryBudget in BudgetProbability, Token with probability
  _Probability : 5.00E-08
end

EventKPreliminaryLabel in SimpleLabel, Token with simple_description
  simpleDescription : EventKSimpleEventDescription
end

EventKSimpleEventDescription in SimpleEventDescription, Token with entity
  _Entity : "Command_Channel"
  qualifying_entity
  qualifyingEntity : "BSCU1"
  condition
  _Condition : "Failure/Error Causes Bad Data"
end

Gate definitions

OrGate4P in OrGate, Token with input
  _Input1 : EventLPreliminaryProfile;
  _Input2 : EventMPreliminaryProfile
end

OrGate5P in OrGate, Token with input
  _Input1 : EventNPreliminaryProfile;
  _Input2 : EventOPreliminaryProfile
end

Definition of Event 'BSCUIMOFV'

EventL in Event, Token with identifier
  _Identifier : "BSCUIMOFV"
preliminary_profile
  preliminaryProfile : EventLPreliminaryProfile
end

EventLPreliminaryProfile in PreliminaryEventProfile, Token with type
  _Type : "base"
  preliminary_budget
  preliminaryBudget : EventLPreliminaryBudget
  preliminary_rate
  preliminaryRate : EventLPreliminaryRate
  preliminary_exposure
  preliminaryExposure : EventLPreliminaryExposure
  preliminary_label
  preliminaryLabel : EventLPreliminaryLabel
end

EventLPreliminaryBudget in BudgetProbability, Token with probability
  _Probability : 2.00E-02
end

EventLPreliminaryRate in Rate, Token with failure_rate
  failureRate : 2.00E-07
end

EventLPreliminaryExposure in Exposure, Token with period
  _Period : 100000
end
Case Study 11: A Brake System Control Unit for a Wheel Braking System of a Hypothetical Aircraft

Definition of Event 'BSCU1_CMDE'

EventM in Event, Token with identifier

_Identifier : "BSCU1_CMDE"
preliminary_profile
EventM_PreliminaryProfile
end

EventM_PreliminaryProfile in PreliminaryEventProfile, Token with type
_Type : "und"
preliminary_budget
EventM_PreliminaryBudget
preliminary_label
EventM_PreliminaryLabel
end

EventM_PreliminaryBudget in BudgetProbability, Token with probability
_Probability : 0.00E+00
end

EventM_PreliminaryLabel in SimpleLabel, Token with simple_description
EventM_SimpleEventDescription
end

Definition of Event 'BSCU1_CPUBD'

EventN in Event, Token with identifier

_Identifier : "BSCU1_CPUBD"
preliminary_profile
EventN_PreliminaryProfile
end

EventN_PreliminaryProfile in PreliminaryEventProfile, Token with type
_Type : "int"
event_connective
EventN_PreliminaryRate
preliminary_budget
EventN_PreliminaryExposure
preliminary_label
EventN_PreliminaryExposure
end

EventN_PreliminaryRate in Rate, Token with failure_rate

_failureRate : 4.50E-06
end

EventN_PreliminaryExposure in Exposure, Token with period

_Period : 0.004167
end

EventN_PreliminaryLabel in SimpleLabel, Token with simple_description
EventN_SimpleEventDescription
end
Tracing Development & Assessment Artifacts

EventOSimpleEventDescription
end

EventOSimpleEventDescription in SimpleEventDescription, Token with entity
  _Entity : "Command_Channel_I/0"
  qualifying_entity
  qualifyingEntity : "BSCU1"
condition
  _Condition : "I/O Failure Causes Bad Data"
end

Gate definitions

OrGate6P in OrGate, Token with input
  _Input1 : EventPPreliminaryProfile;
  _Input2 : EventQPreliminaryProfile
end

Definition of Event `BSCU1CPUDE`

EventP in Event, Token with identifier
  _Identifier : "BSCU1CPUDE"
preliminary_profile
  preliminaryProfile : EventPPreliminaryProfile
end

EventPPreliminaryProfile in PreliminaryEventProfile, Token with type
  _Type : "bas"
  preliminary_budget
    preliminaryBudget : EventPPreliminaryBudget
  preliminary_rate
    preliminaryRate : EventPPreliminaryRate
  preliminary_exposure
    preliminaryExposure : EventPPreliminaryExposure
  preliminary_label
    preliminaryLabel : EventPPreliminaryLabel
end

EventPPreliminaryBudget in BudgetProbability, Token with probability
  _Probability : 3.12E-08
end

EventPPreliminaryRate in Rate, Token with failure_rate
  failureRate : 7.50E-06
end

EventPPreliminaryExposure in Exposure, Token with period
  _Period : 0.004167
end

EventPPreliminaryLabel in SimpleLabel, Token with simple_description
  simpleDescription :
EventPSimpleEventDescription
end

Definition of Event `BSCU1CPUDE`

EventQ in Event, Token with identifier
  _Identifier : "BSCU1CPUDE"
preliminary_profile
  preliminaryProfile : EventQPreliminaryProfile
end

EventQPreliminaryProfile in PreliminaryEventProfile, Token with type
  _Type : "und"
  preliminary_budget
    preliminaryBudget : EventQPreliminaryBudget
  preliminary_label
    preliminaryLabel : EventQPreliminaryLabel
end

EventQPreliminaryBudget in BudgetProbability, Token with probability
  _Probability : 0.00E+00
end

EventQPreliminaryLabel in SimpleLabel, Token with simple_description
  simpleDescription :
EventQSimpleEventDescription
end

Gate definitions

AndGate3P in AndGate, Token with input
  _Input : RSEventSetP
end

RSEventSetP in EventSet, Token with event_profile
  eventProfile1 : EventRPreliminaryProfile;
  eventProfile2 : EventSPreliminaryProfile
end

Definition of Event `SWITCH2`
EventConnective : OrGate7P
preliminary_budget
  preliminaryBudget : EventR
  preliminaryLabel : EventR
end
EventR in BudgetProbability, Token with probability _Probability : 2.50E-01 end
EventR in SimpleLabel, Token with simple_description simpleDescription : EventR end
EventR in SimpleEventDescription, Token with entity _Entity : "Select_Switch" qualifying_entity qualifyingEntity : "BSCU" condition _Condition : "Switch 2 Position"
Definition of Event 'BSCU2INADCM'
EventS in Event, Token with identifier _Identifier : "BSCU2INADCM" preliminary_profile EventSPreliminaryProfile end EventSPreliminaryProfile in PreliminaryEventProfile, Token with type _Type : "int"
event_connective EventConnective : OrGate8P preliminary_budget EventSPreliminaryBudget preliminary_label EventSPreliminaryLabel end EventSPreliminaryBudget in BudgetProbability, Token with probability _Probability : 2.00E-09 end EventSPreliminaryLabel in SimpleLabel, Token with simple_description simpleDescription : EventSPreliminaryDescription end EventSPreliminaryDescription in SimpleEventDescription, Token with entity _Entity : "BSCU2" condition _Condition : "Fault Causes Inadvertent Braking Command" end
Gate definitions
OrGate7P in OrGate, Token with input _Input1 : EventTPreliminaryProfile; _Input2 : EventUPreliminaryProfile end
OrGate8P in OrGate, Token with input _Input1 : EventVreliminaryProfile; _Input2 : EventWreliminaryProfile end
Definition of Event 'BSCU1TF'
EventT in Event, Token with identifier _Identifier : "BSCU1TF" preliminary_profile EventTPreliminaryProfile end EventTPreliminaryProfile in PreliminaryEventProfile, Token with type _Type : "bas"
preliminary_budget EventTPreliminaryBudget preliminary_rate EventTPreliminaryRate preliminary_exposure EventTPreliminaryExposure preliminary_label EventTPreliminaryLabel end EventTPreliminaryBudget in BudgetProbability, Token with probability _Probability : 2.36E-01 end EventTPreliminaryExposure in Exposure, Token with period _Period : 5.00 end EventTPreliminaryLabel in SimpleLabel, Token with simple_description simpleDescription : EventTSimpleEventDescription end EventTSimpleEventDescription in SimpleEventDescription, Token with entity _Entity : "BSCU1" condition _Condition : "Detected Failure" end
Definition of Event 'SWIFAIL2'
EventU in Event, Token with identifier _Identifier : "SWIFAIL2" preliminary_profile EventUPreliminaryProfile end
Tracing Development & Assessment Artifacts

EventUPreliminaryProfile in PreliminaryEventProfile, Token with
type
   _Type : "bas"
preliminary_budget
   preliminaryBudget : EventUPreliminaryBudget
preliminary_rate
   preliminaryRate : EventUPreliminaryRate
preliminary_exposure
   preliminaryExposure : EventUPreliminaryExposure
preliminary_label
   preliminaryLabel : EventUPreliminaryLabel end

EventUPreliminaryBudget in BudgetProbability, Token with
probability
   _Probability : 1.48E-02 end

EventUPreliminaryRate in Rate, Token with
failure_rate
   failureRate : 1.00E-06 end

EventUPreliminaryExposure in Exposure, Token with
period
   _Period : 14750.00 end

EventUPreliminaryLabel in SimpleLabel, Token with
simple_description
   simpleDescription : EventUSimpleEventDescription end

EventUSimpleEventDescription in SimpleEventDescription, Token with
entity
   _Entity : "Select-Switch", qualifying_entity
   qualifyingEntity : "BSCU" condition
   _Condition : "Stuck at Position 21" end

Definition of Event 'BSCU2PSIND'

EventV in Event, Token with
identifier
   _Identifier : "BSCU2PSIND" preliminary_profile
   preliminaryProfile : EventVPreliminaryProfile end

EventVPreliminaryProfile in PreliminaryEventProfile, Token with
type
   _Type : "int"
event_connective
   EventConnective : AndGate4P preliminary_budget
   preliminaryBudget : EventVPreliminaryBudget
preliminary_label
   preliminaryLabel : EventVPreliminaryLabel end

EventVPreliminaryBudget in BudgetProbability, Token with
probability
   _Probability : 1.00E-09 end

EventVPreliminaryLabel in SimpleLabel, Token with
simple_description
   simpleDescription : EventVSimpleEventDescription end

EventVSimpleEventDescription in SimpleEventDescription, Token with
entity
   _Entity : "Power_Supply", qualifying_entity
   qualifyingEntity : "BSCU" condition
   _Condition : "Power Supply Failure Causes Bad Data" end

Definition of Event 'BSCU2CDIND'

EventW in Event, Token with
identifier
   _Identifier : "BSCU2CDIND" preliminary_profile
   preliminaryProfile : EventWPreliminaryProfile end

EventWPreliminaryProfile in PreliminaryEventProfile, Token with
type
   _Type : "int"
event_connective
   EventConnective : AndGate5P preliminary_budget
   preliminaryBudget : EventWPreliminaryBudget
preliminary_label
   preliminaryLabel : EventWPreliminaryLabel end

EventWPreliminaryBudget in BudgetProbability, Token with
probability
   _Probability : 1.00E-09 end

EventWPreliminaryLabel in SimpleLabel, Token with
simple_description
   simpleDescription : EventWSimpleEventDescription end

EventWSimpleEventDescription in SimpleEventDescription, Token with
entity
   _Entity : "Computational_Channel", qualifying_entity
   qualifyingEntity : "BSCU" condition
   _Condition : "Failure/Error Causes Bad Data" end

Gate definitions

AndGate4P in AndGate, Token with
input
   _Input : XYEventSetP end

AndGate5P in AndGate, Token with
input
   _Input : ZAAEventSetP end
XYEventSetP in EventSet, Token with event_profile
  eventProfile1 :
  EventXPreliminaryProfile;
  eventProfile2 :
  EventYPreliminaryProfile end

ZAAEventSetP in EventSet, Token with event_profile
  eventProfile1 :
  Event2PreliminaryProfile;
  eventProfile2 :
  EventAAAPreliminaryProfile end

Definition of Event `BS2PSMOFV'

EventX in Event, Token with identifier
  _Identifier : "BS2PSMOFV"
  preliminary_profile
    preliminaryProfile :
    EventXPreliminaryProfile end

EventXPreliminaryProfile in PreliminaryEventProfile, Token with type
  _Type : "bas"
    preliminary_budget
      preliminaryBudget :
      EventXPreliminaryBudget
    preliminary_rate
      preliminaryRate :
      EventXPreliminaryRate
    preliminary_exposure
      preliminaryExposure :
      EventXPreliminaryExposure
    preliminary_label
      preliminaryLabel :
      EventXPreliminaryLabel end

EventXPreliminaryBudget in BudgetProbability, Token with probability
  _Probability : 2.00E-02
end

EventXPreliminaryRate in Rate, Token with failure_rate
  failureRate : 2.00E-07
end

EventXPreliminaryExposure in Exposure, Token with period
  _Period : 100000.00
end

EventXPreliminaryLabel in SimpleLabel, Token with simple_description
  simpleDescription :
  EventXSimpleEventDescription end

EventXSimpleEventDescription in SimpleEventDescription, Token with entity
  _Entity : "Power_Supply_Monitor"
  qualifying_entity
    qualifyingEntity : "BSCU2"
  condition
    _Condition : "Monitor Stuck Valid"
end

Definition of Event `BSCU2PSF'

EventY in Event, Token with identifier
  _Identifier : "BSCU2PSF"
  preliminary_profile
    preliminaryProfile :
    EventYPreliminaryProfile end

EventYPreliminaryProfile in PreliminaryEventProfile, Token with type
  _Type : "bas"
    preliminary_budget
      preliminaryBudget :
      EventYPreliminaryBudget
    preliminary_rate
      preliminaryRate :
      EventYPreliminaryRate
    preliminary_exposure
      preliminaryExposure :
      EventYPreliminaryExposure
    preliminary_label
      preliminaryLabel :
      EventYPreliminaryLabel end

EventYPreliminaryBudget in BudgetProbability, Token with probability
  _Probability : 5.00E-08
end

EventYPreliminaryRate in Rate, Token with failure_rate
  failureRate : 1.20E-05
end

EventYPreliminaryExposure in Exposure, Token with period
  _Period : 0.004167
end

EventYPreliminaryLabel in SimpleLabel, Token with simple_description
  simpleDescription :
  EventYSimpleEventDescription end

EventYSimpleEventDescription in SimpleEventDescription, Token with entity
  _Entity : "Power_Supply"
  qualifying_entity
    qualifyingEntity : "BSCU2"
  condition
    _Condition : "Failure Causes Bad Data"
end

Definition of Event `BSCU2MORV'

EventZ in Event, Token with identifier
  _Identifier : "BSCU2MORV"
  preliminary_profile
    preliminaryProfile :
    EventZPreliminaryProfile end

EventZPreliminaryProfile in PreliminaryEventProfile, Token with type
  _Type : "int"
    event_connective
      EventConnective : OrGate9P
    preliminary_budget
      preliminaryBudget :
      EventZPreliminaryBudget
preliminary_label
preliminaryLabel:
EventZPreliminaryLabel
end

EventZPreliminaryBudget in
BudgetProbability, Token with
_probability: 2.00E-02
end

EventZPreliminaryLabel in SimpleLabel,
Token with
_simple_description:
EventZSimpleEventDescription
end

EventZSimpleEventDescription in
SimpleEventDescription, Token with
_entity: "Monitor_Channel"
qualifying_entity
qualifyingEntity: "BSCU2"
_condition: "Always Reports Valid"
end

Definition of Event 'BSCU2CDF'

EventAA in Event, Token with
_identifier:
preliminary_profile:
EventAAPreliminaryProfile
end

EventAAPreliminaryProfile in
PreliminaryEventProfile, Token with
type:_Type: "int"
event_connective
preliminary_budget:
preliminaryBudget:
EventAAPreliminaryBudget
preliminary_label:
preliminaryLabel:
EventAAPreliminaryLabel
end

EventAAPreliminaryBudget in
BudgetProbability, Token with
_probability: 5.00E-08
end

EventAAPreliminaryLabel in SimpleLabel,
Token with
_simple_description:
EventAASimpleEventDescription
end

EventAASimpleEventDescription in
SimpleEventDescription, Token with
_entity: "Command_Channel"
qualifying_entity
qualifyingEntity: "BSCU2"
_condition: "Failure/Error Causes Bad Data"
end

Gate definitions

OrGate9P in OrGate, Token with
_input1 : EventBBPreliminaryProfile;
_input2 : EventCCPreliminaryProfile
end

OrGate10P in OrGate, Token with
_input1 : EventDPreliminaryProfile;
_input2 : EventEPreliminaryProfile
end

Definition of Event 'BSCU2MOPFV'

EventBB in Event, Token with
_identifier:
preliminary_profile:
preliminaryProfile:
EventBBPreliminaryProfile
end

EventBBPreliminaryProfile in
PreliminaryEventProfile, Token with
type:_Type: "bas"
preliminary_budget:
preliminaryBudget:
EventBBPreliminaryBudget
preliminary_rate:
preliminaryRate:
EventBBPreliminaryRate
preliminary_exposure:
preliminaryExposure:
EventBBPreliminaryExposure
preliminary_label:
preliminaryLabel:
EventBBPreliminaryLabel
end

EventBBPreliminaryBudget in
BudgetProbability, Token with
_probability: 2.00E-02
end

EventBBPreliminaryRate in Rate, Token with
_failure_rate:
failureRate: 2.00E-07
end

EventBBPreliminaryExposure in Exposure,
Token with
_period: 100000.00
end

EventBBPreliminaryLabel in SimpleLabel,
Token with
_simple_description:
EventBBSimpleEventDescription
end

EventBBSimpleEventDescription in
SimpleEventDescription, Token with
_entity: "Validity_Monitor"
qualifying_entity
qualifyingEntity: "BSCU2"
_condition: "Failed Valid due to Hardware Failure"
end

Definition of Event 'BSCU2CMDE'

EventCC in Event, Token with
_identifier:
preliminary_profile:
preliminaryProfile:
Case Study II: A Brake System Control Unit for a Wheel Braking System of a Hypothetical Aircraft

EventCCPreliminaryProfile in PreliminaryEventProfile, Token with type _Type : "und"
preliminary_budget : EventCCPreliminaryBudget
preliminary_label : EventCCPreliminaryLabel
end

EventCCPreliminaryBudget in BudgetProbability, Token with probability _Probability : 0.00E+00
end

EventCCPreliminaryLabel in SimpleLabel, Token with simple_description : EventCCSimpleEventDescription
end

EventCCSimpleEventDescription in SimpleEventDescription, Token with entity _Entity : "Monitor_Channel"
qualifying_entity : "BSCU2"
condition _Condition : "Design Error"
end

Definition of Event 'BSCU2CPUBD'

EventDD in Event, Token with identifier _Identifier : "BSCU2CPUBD"
preliminary_profile : EventDDPreliminaryProfile
end

EventDDPreliminaryProfile in PreliminaryEventProfile, Token with type _Type : "int"
preliminary_budget : EventDDPreliminaryBudget
preliminary_label : EventDDPreliminaryLabel
end

EventDDPreliminaryBudget in BudgetProbability, Token with probability _Probability : 3.12E-08
end

EventDDPreliminaryLabel in SimpleLabel, Token with simple_description : EventDDSimpleEventDescription
end

EventDDSimpleEventDescription in SimpleEventDescription, Token with entity _Entity : "Command_Channel_CPU"
qualifying_entity : "BSCU2"
condition _Condition : "CPU Failure/Error Causes Bad Data"
end

Definition of Event 'BSCU2I/OF'

EventEE in Event, Token with identifier _Identifier : "BSCU2I/OF"
preliminary_profile : EventEEPreliminaryProfile
end

EventEEPreliminaryProfile in PreliminaryEventProfile, Token with type _Type : "bas"
preliminary_budget : EventEEPreliminaryBudget
preliminary_rate : EventEEPreliminaryRate
preliminary_exposure : EventEEPreliminaryExposure
preliminary_label : EventEEPreliminaryLabel
end

EventEEPreliminaryBudget in BudgetProbability, Token with probability _Probability : 1.88E-08
end

EventEEPreliminaryRate in Rate, Token with failure_rate : 4.50E-06
end

EventEEPreliminaryExposure in Exposure, Token with period _Period : 0.004167
end

EventEEPreliminaryLabel in SimpleLabel, Token with simple_description : EventEESimpleEventDescription
end

EventEESimpleEventDescription in SimpleEventDescription, Token with entity _Entity : "Command_Channel_I/O"
qualifying_entity : "BSCU2"
condition _Condition : "I/O Failure Causes Bad Data"
end

Gate definitions

OrGate1P in OrGate, Token with input _Input1 : EventFFPreliminaryProfile;
_input2 : EventGGreliminaryProfile
end

Definition of Event 'BSCU2CPUF'
Tracing Development & Assessment Artifacts

EventFF in Event, Token
with
  identifier
    _identifier : "BSCU2CPUF"
preliminary_profile
  preliminaryProfile :
EventFFPreliminaryProfile
end

EventFFPreliminaryProfile in
PreliminaryEventProfile, Token with
type
  _Type : "bas"
preliminary_budget
  preliminaryBudget :
EventFFPreliminaryBudget
preliminary_rate
  preliminaryRate :
EventFFPreliminaryRate
preliminary_exposure
  preliminaryExposure :
EventFFPreliminaryExposure
preliminary_label
  preliminaryLabel :
EventFFPreliminaryLabel
end

EventFFPreliminaryBudget in
BudgetProbability, Token with
probability
  _Probability : 3.12E-08
end

EventFFPreliminaryRate in Rate, Token with
failure_rate
  failureRate : 7.50E-06
end

EventFFPreliminaryExposure in Exposure,
Token with
period
  _Period : 0.004167
end

EventFFPreliminaryLabel in SimpleLabel,
Token with
simple_description
  simpleDescription :
EventFFSimpleEventDescription
end

Definition of Event `BSCU2CPUDE'

EventGG in Event, Token
with
  identifier
    _identifier : "BSCU2CPUDE"
preliminary_profile
  preliminaryProfile :
EventGGPreliminaryProfile
end

EventGGPreliminaryProfile in
PreliminaryEventProfile, Token with
type
  _Type : "und"
preliminary_budget
  preliminaryBudget :
EventGGPreliminaryBudget
preliminary_label
  preliminaryLabel :
EventGGPreliminaryLabel
end

EventGGPreliminaryBudget in
BudgetProbability, Token with
probability
  _Probability : 0.00E+00
end

EventGGPreliminaryRate in Rate, Token with
failure_rate
  failureRate :
end

EventGGPreliminaryExposure in Exposure,
Token with
period
  _Period : 0.004167
end

EventGGPreliminaryLabel in SimpleLabel,
Token with
simple_description
  simpleDescription :
EventGGSimpleEventDescription
end

Definition of Event `BSCU2CPUDE'

EventGG in Event, Token
with
  identifier
    _identifier : "BSCU2CPUDE"
preliminary_profile
  preliminaryProfile :
EventGGPreliminaryProfile
end

EventGGPreliminaryProfile in
PreliminaryEventProfile, Token with
type
  _Type : "und"
preliminary_budget
  preliminaryBudget :
EventGGPreliminaryBudget
preliminary_label
  preliminaryLabel :
EventGGPreliminaryLabel
end

Instantiation of Preliminary Fault Tree

BscuPssaPrelimFT in
PreliminaryFaultTree, Token with
ft_and_gate
  ftAndGate1 : AndGate1P;
  ftAndGate2 : AndGate2P;
  ftAndGate3 : AndGate3P;
  ftAndGate4 : AndGate4P;
  ftAndGate5 : AndGate5P
ft_or_gate
  ftOrGate1 : OrGate1P;
  ftOrGate2 : OrGate2P;
  ftOrGate3 : OrGate3P;
  ftOrGate4 : OrGate4P;
  ftOrGate5 : OrGate5P;
  ftOrGate6 : OrGate6P;
  ftOrGate7 : OrGate7P;
  ftOrGate8 : OrGate8P;
  ftOrGate9 : OrGate9P;
  ftOrGate10 : OrGate10P;
  ftOrGate11 : OrGate11P
ft_event_set
  ftEventSet1 : HIEventSetP;
  ftEventSet2 : JKEventSetP;
  ftEventSet3 : KSEventSetP;
  ftEventSet4 : XYEventSetP;
  ftEventSet5 : ZAAEventSetP
ft_event_profile
  ftEventProfile1 : HIEventProfile;
  ftEventProfile2 : JKEventProfile;
  ftEventProfile3 : KSEventProfile;
  ftEventProfile4 : XYEventProfile;
  ftEventProfile5 : ZAAEventProfile;
  ftEventProfile33 :
  ftBudget1 : EventABudget;
6.3.4 System Safety Assessment - Brake System Control Unit

At this point, we assume the PSSA is sufficiently developed to allow detailed design implementation to proceed. Therefore our example resumes post-implementation by considering System Safety Assessment of the BSCU towards verification of PSSA safety objectives. This includes Functional and Piece-Part FMEAs of the power supply and power supply monitor respectively, as well as updating of the preliminary fault tree introduced in 6.3.3.2. Artifacts produced by these activities will be used to demonstrate instantiation of the FMEA and Fault Tree Analysis structures, together with a partial population of the Circuit Diagram meta-model from Chapter Three.

6.3.4.1 Background on BSCU Power Supply Design

The BSCU1 and BSCU2 power supplies are identical in both their design and implementation and are located in physically remote areas of the BSCU circuit card assembly. Within each BSCU, the power supply and power supply monitor functions are physically independently located. Power supply design is depicted in the block diagram6 in figure 6.15, whilst a detailed circuit schematic of the +5 volt monitor appears in figure 6.16.

6 Failure rates in figure 6.15 are determined by the functional FMEA; e.g., +5 volt rate derived from the sum of failures shown in table 6.3 (subsection 6.3.4.2.1).
The power supply monitors are window comparators\(^7\). Both +5 volt and +15 volt (shown as ‘Power Supply Monitor’ in figure 6.15) are monitored for over and under voltage conditions and their outputs And-ed together such that if voltage exceeds the trip-point, high or low, monitor output is pulled low.

---

\(^7\) Window comparators monitor an output line and inform the controller when voltage moves outside specific settings.
Case Study II: A Brake System Control Unit for a Wheel Braking System of a Hypothetical Aircraft

Definition of Power Supply Monitor Circuit Schematic

CircuitDiagram in DevelopmentStructure, SimpleClass

isA AerospaceEngineeringObject with has_property
circuit_name : String

has_element

- resistor : Resistor;
- capacitor : Capacitor;
- comparator : Comparator;
- and_gate : AndGate;
- dc_source : DCSource

PowerSupplyMonitorCD in CircuitDiagram, Token with

circuit_name : "Power Supply Monitor"

circuitName : Power Supply Monitor

resistor

- resistor1 : R1;
- resistor2 : R2;
- resistor3 : R3;
- resistor4 : R4;
- resistor5 : R5;
- resistor6 : R6;
- resistor7 : R7

capacitor

- capacitor1 : C1;
- capacitor2 : C2

comparator

- comparator1 : U1A;
- comparator2 : U1B

and_gate

- andGate : U2

dc_source

- dcSource : U3

We return to the Circuit Diagram meta-model in subsection 6.3.5 when demonstrating trace relations between development and assessment artifacts.

6.3.4.2 Failure Modes and Effects Analysis

This FMEA considers the BSCU power supply with respect to safety objectives for basic events in the 'inadvertent wheel braking' fault tree from 6.3.3.2. Basic events supported by this particular analysis are Power Supply • Failure Causes Bad Data and Power Supply Monitor • Monitor Stuck Valid.

The FMEA is performed in two parts: i) a functional analysis of the entire power supply; and ii) a piece-part analysis of the power supply monitor. For the purpose of this example, the latter is assumed to be necessary after results of the functional FMEA did not meet the safety objectives. Note, that the functional FMEA on the power supply monitor is superseded by the piece-part FMEA.

6.3.4.2.1 Functional Failure Modes and Effects Analysis

According to the scenario outlined in ARP 4761, initial analysis of the power supply was conducted by computing the total power supply failure rate based on parts counts and failure rates. Conservative analysis (details of which are not included) failed to meet the safety budget for Power Supply • Failure Causes Bad Data (namely 1.20E-05 as shown in figure 6.14) and so a functional FMEA was performed to provide better resolution to the failure rates of the various failure modes. This investigation concluded that the only failure that may cause bad data and may not be detected by a fully functioning power supply monitor is 'loss of filtering'. This failure can cause increased ripple on the output.
voltages at such a level and frequency that it goes undetected by the monitor. Table 6.3 shows a partial results table for the BSCU power supply functional FMEA showing analysis of the +5 volt function.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Failure Mode</th>
<th>Flight Phase</th>
<th>Failure Rate</th>
<th>Failure Effect</th>
<th>Detection Method</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>+5 Volt Power Supply</td>
<td>+5 volt out of spec</td>
<td>All</td>
<td>0.2143E-06</td>
<td>Power Supply shut down.</td>
<td>The Power Supply Monitor is tripped.</td>
<td>The BSCU channel fails.</td>
</tr>
<tr>
<td></td>
<td>+5 volt short to ground</td>
<td>All</td>
<td>0.2857E-06</td>
<td>Power Supply shut down.</td>
<td>The Power Supply Monitor will pass invalid power supply to other BSCU.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>loss of filtering</td>
<td>All</td>
<td>0.3571E-06</td>
<td>Increased Ripple.</td>
<td>May emit out of spec voltage if ripple is not detected by monitor. May cause spurious trip.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+ 5 volt open</td>
<td>All</td>
<td>0.5714E-06</td>
<td>Power Supply shut down.</td>
<td>The Power Supply Monitor will pass invalid power supply to other BSCU.</td>
<td></td>
</tr>
<tr>
<td>No Effect</td>
<td>All</td>
<td></td>
<td>0.1429E-06</td>
<td>No Effect.</td>
<td>None/No Effect.</td>
<td>No Effect.</td>
</tr>
</tbody>
</table>

Table 6.3 - ‘Functional FMEA (Partial) of BSCU Power Supply’

ARP 4761 states that the actual failure rate for Power Supply Failure Causes Bad Data is 1.06E-05 (derivation not shown) which satisfies the requirement of 1.20E-05 failures per hour. This information will be used in subsection 6.3.4.3 to update the BSCU preliminary fault tree.

### 6.3.4.2.2 Instantiation of Functional Failure Modes and Effects Analysis Meta-model

We now instantiate the O-Telos representation of our Functional Failure Modes and Effects Analysis meta-model from section 5.3 with a subset of information from table 6.3.

```javascript
+ 5 Volt Power Supply Function Failure Mode Descriptions
+5voltpsFailureDescription in FunctionFailureDescription, Token with fnc_id : "+ 5 Volt Power Supply"
  fnc_fln : "+5 volt out of spec"
  fncFailModeDescription1 : VoltageOutofSpecFailureModeDescription;
  fncFailModeDescription2 : ShortToGndFailureModeDescription;
  fncFailModeDescription3 : LossOfFilteringFailureModeDescription;
  fncFailModeDescription4 : VoltageOpenFailureModeDescription
end
VoltageOutofSpecFailureModeDescription in FunctionFailureModeDescription, Token with fnc_fail_mode
  fncFailMode : "+5 volt out of spec"
  fnc_FlightPhaseDesc : VoltageOutofSpecFlightPhaseFailureDescription
  fnc_detection
  fnc_detection : VoltageOutofSpecDetection
  fnc_comment
  fncComment : VoltageOutofSpecComments
end
VoltageOutofSpecFlightPhaseFailureDescription in FlightPhaseFailureDescription, Token with fnc_flight_phase
  fncFlightPhase : "All"
  fnc_fail_rate
    fncFailRate : 0.2143E-06
    fnc_phase_fail_effect
      fncPhaseFailEffect : "Power Supply shut down."
end
VoltageOutofSpecDetection in MatraNaturalLanguageStructure, Token with mnls_composite
  mnlsComposite1 : VOoSDC1;
  mnlsComposite2 : VOoSDC2
  mnls_plain_text
    mnlsText1 : VOoSDPT1;
    mnlsText2 : VOoSDPT2;
    mnlsText3 : VOoSDPT3
  mnls_module
    mnlsModule1 : PowerSupplyMonitorNode
  mnls_condition
    mnlsCondition1 : PowerSupplyMonitorTripNode
end
VOoSDC1 in MatraNLSComposite, Token with
```
preceding_fragment
precedingFragment : VooSDPT1
subject_node
subjectNode : PowerSupplyMonitorNode
following_fragment
followingFragment : VooSDC2
end
VooSDPT1 in PlainTextNode, Token with
mlns_text
mlnsText : "The ". end
PowerSupplyMonitorNode in ModuleNode,
Token with
module_name
moduleName : "Power Supply Monitor" end
VooSDC2 in MatraNLSComposite, Token with
preceding_fragment
precedingFragment : VooSDPT2
subject_node
subjectNode : PowerSupplyMonitorTripNode
following_fragment
followingFragment : VooSDT3
end
VooSDPT2 in PlainTextNode, Token with
mlns_text
mlnsText : " is ". end
PowerSupplyMonitorTripNode in ConditionNode, Token with
condition_name
conditionName : "tripped" end
VooSDT3 in PlainTextNode, Token with
mlns_text
mlnsText : "." end
VoltageOutofSpecComments in MatraNaturalLanguageStructure, Token with
mlns_composite
mlnsComposite1 : VooSCC1
mlnsPlainText1 : VooSCPT1;
mlnsPlainText2 : VooSCPT2
mlns_condition_node
mlnsConditionNode1 : BSCUChannelFailNode end
VooSCC1 in MatraNLSComposite, Token with
preceding_fragment
precedingFragment : VooSCPT1
subject_node
subjectNode : BSCUChannelFailNode
following_fragment
followingFragment : VooSCPT2
end
VooSCPT1 in PlainTextNode, Token with
mlns_text
mlnsText : "The ". end
BSCUChannelFailNode in ConditionNode, Token with
condition_name
conditionName : "channel fails" qualified_by
qualifiedBy : BSCUNode end
BSCUNode in ModuleNode, Token with
module_name
moduleName : "BSCU" end
VooSCPT2 in PlainTextNode, Token with
mlns_text
mlnsText : "." end
ShortToGndFailureModeDescription in FunctionFailureModeDescription, Token with
fnc_fail_mode
fncFailMode : "+5 volt short to ground"
fnc_flight_phase_desc
fncFlightPhaseDesc1 : ShortToGndFlightPhaseFailureDescription
defc_detection
defcDetection : ShortToGndDetection end
ShortToGndDetection in MatraNaturalLanguageStructure, Token with
mlns_plain_text
mlnsPlainText1 : StGDP1
end
StGDP1 in PlainTextNode, Token with
mlns_text
mlnsText : "The Power Supply Monitor will pass invalid power supply to other BSCU." end
-- Note : primitives not demarcated in this and following MNLS examples
LossOfFilteringFailureModeDescription in FunctionFailureModeDescription, Token with
fnc_fail_mode
fncFailMode : "loss of filtering"
fnc_flight_phase_desc
fncFlightPhaseDesc1 : LossofFilteringFlightPhaseFailureDescription
defc_detection
defcDetection : LossofFilteringDetection defc_comment
defcComment : LossofFilteringComments end
LossofFilteringDetection in MatraNaturalLanguageStructure, Token with
mlns_plain_text
6.3.4.2.3 Piece-Part Failure Modes and Effects Analysis

Turning to the other basic event under consideration, Monitor Stuck Valid; our scenario assumes initial analysis of the power supply monitor was part of the functional FMEA of the entire power supply. However, it was found that the total failure rate of the power supply monitor was 3.06E-7 failures per hour which does not comply with the budgeted requirement of 2.0E-7 (as shown in figure 6.14).

Accordingly a detailed piece-part FMEA was conducted to provide better resolution into the probability of Monitor Stuck Valid. Partial results of this analysis are shown in table 6.4.
<table>
<thead>
<tr>
<th>Component</th>
<th>Component Type</th>
<th>Failure Mode</th>
<th>Failure Rate</th>
<th>Failure Effect</th>
<th>Detection Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Ceramic Capacitor</td>
<td>short</td>
<td>0.0073E-06</td>
<td>Monitor stuck tripped</td>
<td>Power Supply shut down by Monitor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>open</td>
<td>0.0013E-06</td>
<td>Loss of delay</td>
<td>Spurious monitor trip</td>
</tr>
<tr>
<td></td>
<td></td>
<td>low cap</td>
<td>0.0019E-06</td>
<td>Decrease delay to trip</td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>Ceramic Capacitor</td>
<td>short</td>
<td>0.0073E-06</td>
<td>Monitor stuck tripped</td>
<td>Power Supply shut down by Monitor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>open</td>
<td>0.0013E-06</td>
<td>Loss of delay</td>
<td>Spurious monitor trip</td>
</tr>
<tr>
<td></td>
<td></td>
<td>low cap</td>
<td>0.0019E-06</td>
<td>Decrease delay to trip</td>
<td></td>
</tr>
<tr>
<td>U1A</td>
<td>Comparator IC</td>
<td>output open</td>
<td>0.0124E-06</td>
<td>Monitor stuck valid</td>
<td>Bench test.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>output grounded</td>
<td>0.0056E-06</td>
<td>Monitor trip</td>
<td>Power Supply shut down.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>high offset voltage</td>
<td>0.0062E-06</td>
<td>Loss of monitor sensitivity</td>
<td>Bench test.</td>
</tr>
<tr>
<td>U1B</td>
<td>Comparator IC</td>
<td>output open</td>
<td>0.0124E-06</td>
<td>Monitor stuck valid</td>
<td>Bench test.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>output grounded</td>
<td>0.0056E-06</td>
<td>Monitor trip</td>
<td>Power Supply shut down.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>high offset voltage</td>
<td>0.0062E-06</td>
<td>Loss of monitor sensitivity</td>
<td>Bench test.</td>
</tr>
<tr>
<td>R1</td>
<td>Film Resistor</td>
<td>open</td>
<td>0.0009E-06</td>
<td>Monitor trip</td>
<td>Power Supply shut down.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>increase resistance</td>
<td>0.0005E-06</td>
<td>Trip window shifts down</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>decrease resistance</td>
<td>0.0004E-06</td>
<td>Trip window shifts up</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>Film Resistor</td>
<td>open</td>
<td>0.0009E-06</td>
<td>Monitor trip</td>
<td>Power Supply shut down.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>increase resistance</td>
<td>0.0005E-06</td>
<td>Trip window shifts up</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>decrease resistance</td>
<td>0.0004E-06</td>
<td>Trip window shifts down</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>Film Resistor</td>
<td>open</td>
<td>0.0009E-06</td>
<td>Monitor trip</td>
<td>Power Supply shut down.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>increase resistance</td>
<td>0.0005E-06</td>
<td>Trip window shifts up</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>decrease resistance</td>
<td>0.0004E-06</td>
<td>Trip window shifts down</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>Film Resistor</td>
<td>open</td>
<td>0.0009E-06</td>
<td>Monitor stuck valid</td>
<td>Bench test.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>increase resistance</td>
<td>0.0005E-06</td>
<td>Trip window tightens</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>decrease resistance</td>
<td>0.0004E-06</td>
<td>Trip window widens Monitor stuck valid</td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td>Film Resistor</td>
<td>open</td>
<td>0.0009E-06</td>
<td>Monitor trip</td>
<td>Power Supply shut down.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>increase resistance</td>
<td>0.0005E-06</td>
<td>Trip window shifts down</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>decrease resistance</td>
<td>0.0004E-06</td>
<td>Trip window shifts up</td>
<td></td>
</tr>
</tbody>
</table>
Tracing Development & Assessment Artifacts

<table>
<thead>
<tr>
<th>Component</th>
<th>Component Type</th>
<th>Failure Mode</th>
<th>Failure Rate</th>
<th>Failure Effect</th>
<th>Detection Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>R6</td>
<td>Film Resistor</td>
<td>open</td>
<td>0.0009E-06</td>
<td>Monitor stuck tripped</td>
<td>Power Supply shut down.</td>
</tr>
<tr>
<td>R7</td>
<td>Film Resistor</td>
<td>open</td>
<td>0.0009E-06</td>
<td>Monitor stuck tripped</td>
<td>Power Supply shut down.</td>
</tr>
<tr>
<td>U2</td>
<td>AND Gate</td>
<td>stuck high</td>
<td>0.0108E-06</td>
<td>Monitor stuck valid</td>
<td>Bench test.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>stuck low</td>
<td>0.0054E-06</td>
<td>Monitor stuck tripped</td>
<td>Power Supply shut down.</td>
</tr>
<tr>
<td>U3</td>
<td>Voltage Reference</td>
<td>inop</td>
<td>0.0110E-06</td>
<td>Monitor trip</td>
<td>Power Supply shut down.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>out of spec</td>
<td>0.0058E-06</td>
<td>Window shift</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>short</td>
<td>0.0026E-06</td>
<td>Monitor trip</td>
<td>Power Supply shut down.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>open</td>
<td>0.0245E-06</td>
<td>Monitor trip</td>
<td>Power Supply shut down.</td>
</tr>
</tbody>
</table>

Table 6.4 - 'Piece-Part FMEA (Partial) of BSCU Power Supply Monitor'

Examination of table 6.4 reveals several potential failure modes that may cause a monitor to become 'stuck valid', namely U1A • output open, U1B • output open, R4 • open, R4 • decrease resistance and U2 • stuck high. Summed rates (including those not shown in the above) for contributing failures yield an actual failure rate of 1.429E-07, which satisfies the budget of 2.00E-07. This information will also be used in subsection 6.3.4.3 to update the BSCU preliminary fault tree.

6.3.4.2.4 Instantiation of Piece-Part Failure Modes and Effects Analysis Meta-model

We now instantiate elements from the O-Telos representation of our Piece-Part Failure Modes and Effects Analysis meta-model with information from table 6.4.
Case Study II: A Brake System Control Unit for a Wheel Braking System of a Hypothetical Aircraft

ComDetection : C10penDetection
end

C10penDetection in MatraNaturalLanguageStructure, Token with
mls_plain_text
mlsPlainText1 : C10penDPT1
end

C10penDPT1 in PlaintextNode, Token with
mls_text
mlsText : "Power Supply shut down."
end

C1LowCapFailModeDescription in ComponentFailureModeDescription, Token with
com_fail_mode
comFailMode : "low cap"
com_fail_rate
comFailRate : 0.0019E-06
com_fail_effect
comFailEffect1 : "Decrease delay to trip"
end

Definition of component C2 failure description

C2FailureDescription in ComponentFailureDescription, Token with
com_name
comName : "C2"
com_type
comType : "Ceramic Capacitor"
com_fail_mode_description
comFailModeDescription1 : C2ShortFailModeDescription;
comFailModeDescription2 : C2OpenFailModeDescription;
comFailModeDescription3 : C2LowCapFailModeDescription
end

C2ShortFailModeDescription in ComponentFailureModeDescription, Token with
com_fail_mode
comFailMode : "short"
com_fail_rate
comFailRate : 0.0073E-06
com_fail_effect
comFailEffect1 : "Monitor stuck tripped"
com_detection
comDetection : C2ShortDetection
end

C2ShortDetection in MatraNaturalLanguageStructure, Token with
mls_plain_text
mlsPlainText1 : C2ShortDPT1
end

C2ShortDPT1 in PlaintextNode, Token with
mls_text
mlsText : "Power Supply shut down."
end

C2OpenFailModeDescription in ComponentFailureModeDescription, Token with
com_fail_mode
comFailMode : "open"
com_fail_rate
comFailRate : 0.0013E-06
com_fail_effect
comFailEffect1 : "Loss of delay";
comFailEffect2 : "Spurious monitor trip"
com_detection
comDetection : C2OpenDetection
end

C2OpenDetection in MatraNaturalLanguageStructure, Token with
mls_plain_text
mlsPlainText1 : C2OpenDPT1
end

C2OpenDPT1 in PlaintextNode, Token with
mls_text
mlsText : "Power Supply shut down."
end

C2LowCapFailModeDescription in ComponentFailureModeDescription, Token with
com_fail_mode
comFailMode : "low cap"
com_fail_rate
comFailRate : 0.0019E-06
com_fail_effect
comFailEffect1 : "Decrease delay to trip"
end

Definition of component U1A failure description

U1AFailureDescription in ComponentFailureDescription, Token with
com_name
comName : "U1A"
com_type
comType : "Comparator IC"
com_fail_mode_description
comFailModeDescription1 : U1AOutputOpenFailModeDescription;
comFailModeDescription2 : U1AGroundedFailModeDescription;
comFailModeDescription3 : U1AHighOffsetVoltageFailModeDescription
end

U1AOutputOpenFailModeDescription in ComponentFailureModeDescription, Token with
com_fail_mode
comFailMode : "output open"
com_fail_rate
comFailRate : 0.0124E-06
com_fail_effect
comFailEffect1 : "Monitor stuck valid"
com_detection
comDetection : U1AOutputOpenDetection
end

U1AOutputOpenDetection in MatraNaturalLanguageStructure, Token with
mls_plain_text
mlsPlainText1 : U1AOutputOpenDPT1
end

U1AOutputOpenDPT1 in PlaintextNode, Token with
mls_text
mlsText : "Bench test."
end

U1AGroundedFailModeDescription in ComponentFailureModeDescription, Token with
com_fail_mode
comFailMode : "grounded"
com_fail_rate
comFailRate : 0.001E-06
com_fail_effect
comFailEffect1 : "Monitor stuck valid"
com_detection
comDetection : U1AGroundedDetection
end

U1AGroundedDetection in MatraNaturalLanguageStructure, Token with
mls_plain_text
mlsPlainText1 : U1AGroundedDPT1
end

U1AGroundedDPT1 in PlaintextNode, Token with
mls_text
mlsText : "Bench test."
end

342
UIAOutputGroundedFailModeDescription in ComponentFailureModeDescription, Token with
  com_fail_mode
    comFailMode : "output grounded"
  com_fail_rate
    comFailRate : 0.0056E-06
  com_fail_effect
    comFailEffect1 : "Monitor trip"
  com_detection
    comDetection : UIAOutputGroundedDetection
end

UIAOutputGroundedDetection in MatraNaturalLanguageStructure, Token with
  mnls_plain_text
    mnlsPlainText1 : UIAOutputGroundedDPT1
end

UIAOutputGroundedDPT1 in PlainTextNode, Token with
  mnls_text
    mnlsText : "Power Supply shut down."
end

UIAHighOffsetVoltageFailModeDescription in ComponentFailureModeDescription, Token with
  com_fail_mode
    comFailMode : "high offset voltage"
  com_fail_rate
    comFailRate : 0.0062E-06
  com_fail_effect
    comFailEffect1 : "Loss of monitor sensitivity"
  com_detection
    comDetection : U1BHighOffsetVoltageDetection
end

UIAHighOffsetVoltageDetection in MatraNaturalLanguageStructure, Token with
  mnls_plain_text
    mnlsPlainText1 : U1BHighOffsetVoltageDPT1
end

UIBOutputOpenDPT1 in PlainTextNode, Token with
  mnls_text
    mnlsText : "Power Supply shut down."
end

UIBFailureDescription in ComponentFailureDescription, Token with
  com_name
    comName : "U1B"
  com_type
    comType : "Comparator IC"
  com_fail_mode_description
    comFailModeDescription1 : U1BOutputOpenFailModeDescription;
    comFailModeDescription2 : U1BOutputGroundedFailModeDescription;
    comFailModeDescription3 : U1BHighOffsetVoltageFailModeDescription
end

UIBOutputOpenFailModeDescription in ComponentFailureModeDescription, Token with
  com_fail_mode
    comFailMode : "output open"
  com_fail_rate
    comFailRate : 0.0124E-06
  com_fail_effect
    comFailEffect1 : "Monitor stuck valid"
  com_detection
    comDetection : U1BOutputOpenDetection
end

U1BOutputOpenDetection in MatraNaturalLanguageStructure, Token with
  mnls_plain_text
    mnlsPlainText1 : U1BOutputOpenDPT1
end
Case Study II: A Brake System Control Unit for a Wheel Braking System of a Hypothetical Aircraft

Definition of component R1 failure description

R1FailureDescription in ComponentFailureDescription, Token with
com_name comName : "R1"
com_type comType : "Film Resistor"
com_fail_mode_description comFailModeDescription1
R1OpenFailModeDescription;
comFailModeDescription2
R1IncreaseResistanceFailModeDescription;
comFailModeDescription3
R1DecreaseResistanceFailModeDescription end

R1OpenFailModeDescription in ComponentFailureModeDescription, Token with
com_fail_mode comFailMode : "open"
com_fail_rate comFailRate : 0.0009E-06
com_fail_effect comFailEffect1 : "Monitor trip"
com_detection comDetection : R1OpenDetection end

R1OpenDetection in MatraNaturalLanguageStructure, Token with
mnls_plain_text mnlsPlainText1 : R1OpenDPT1 end

R1OpenDPT1 in PlainTextNode, Token with
mnls_text mnlsText : "Power Supply shut down."
end

R1IncreaseResistanceFailModeDescription in ComponentFailureModeDescription, Token with
com_fail_mode comFailMode : "increase resistance"
com_fail_rate comFailRate : 0.0005E-06
com_fail_effect comFailEffect1 : "Trip window shifts up" end

R1DecreaseResistanceFailModeDescription in ComponentFailureModeDescription, Token with
com_fail_mode comFailMode : "decrease resistance"
com_fail_rate comFailRate : 0.0004E-06
com_fail_effect comFailEffect1 : "Trip window shifts down" end

R2IncreaseResistanceFailModeDescription in ComponentFailureModeDescription, Token with
com_fail_mode comFailMode : "increase resistance"
com_fail_rate comFailRate : 0.0005E-06
com_fail_effect comFailEffect1 : "Trip window shifts up" end

R2DecreaseResistanceFailModeDescription in ComponentFailureModeDescription, Token with
com_fail_mode comFailMode : "decrease resistance"
com_fail_rate comFailRate : 0.0004E-06
com_fail_effect comFailEffect1 : "Trip window shifts down" end

Definition of component R3 failure description

R3FailureDescription in ComponentFailureDescription, Token with
com_name comName : "R3"
com_type comType : "Film Resistor"
com_fail_mode_description comFailModeDescription1
R3OpenFailModeDescription;
comFailModeDescription2
R3IncreaseResistanceFailModeDescription;
comFailModeDescription3
R3DecreaseResistanceFailModeDescription end

R3OpenFailModeDescription in ComponentFailureModeDescription, Token with
com_fail_mode comFailMode : "open"
com_fail_rate comFailRate : 0.0009E-06
com_fail_effect comFailEffect1 : "Monitor trip"
com_detection comDetection : R3OpenDetection end

R3OpenDetection in MatraNaturalLanguageStructure, Token with
mnls_plain_text mnlsPlainText1 : R3OpenDPT1 end

R3OpenDPT1 in PlainTextNode, Token with
mnls_text mnlsText : "Power Supply shut down."
end

Definition of component R2 failure description

R2FailureDescription in ComponentFailureDescription, Token with
com_name comName : "R2"
com_type comType : "Film Resistor"
com_fail_mode_description
comFailMode : "open"
com_fail_rate  
comFailRate : 0.0009E-06
com_fail_effect  
comFailEffect1 : "Monitor trip"
com_detection  
comDetection : R3OpenDetection  
end

R3OpenDetection in  
MatraNaturalLanguageStructure, Token with

R3OpenDPT1 in PlainTextNode, Token with

R3IncreaseResistanceFailModeDescription in  
ComponentFailureModeDescription, Token with

R3DecreaseResistanceFailModeDescription in  
ComponentFailureModeDescription, Token with

Definition of component R4 failure description

R4FailureDescription in  
ComponentFailureDescription, Token with

R4OpenFailureModeDescription;  
R4IncreaseResistanceFailureModeDescription;  
R4DecreaseResistanceFailureModeDescription

R4OpenFailureModeDescription in  
ComponentFailureModeDescription, Token with

R4OpenDPT1 in PlainTextNode, Token with

R4IncreaseResistanceFailureModeDescription in  
ComponentFailureModeDescription, Token with

R4DecreaseResistanceFailureModeDescription in  
ComponentFailureModeDescription, Token with

Definition of component R5 failure description

R5FailureDescription in  
ComponentFailureDescription, Token with

R4OpenDPT1 in PlainTextNode, Token with

R4IncreaseResistanceDetection in

R4DecreaseResistanceDetection in  
MatraNaturalLanguageStructure, Token with

R4DecreaseResistanceDPT1 in PlainTextNode, Token with

R4DecreaseResistanceFailureModeDescription in  
ComponentFailureModeDescription, Token with

R5FailureDescription in  
ComponentFailureDescription, Token with

R4DecreaseResistanceDPT1 in PlainTextNode, Token with

Definition of component R5 failure description

R5IncreaseResistanceDetection in

R5DecreaseResistanceDetection in  
MatraNaturalLanguageStructure, Token with

R5DecreaseResistanceDPT1 in PlainTextNode, Token with

Definition of component R5 failure description

R5IncreaseResistanceDetection in

R5DecreaseResistanceDetection in  
MatraNaturalLanguageStructure, Token with

R5DecreaseResistanceDPT1 in PlainTextNode, Token with

Definition of component R5 failure description

R5FailureDescription in  
ComponentFailureDescription, Token with

R4OpenDetection

R4OpenDPT1

R4IncreaseResistanceDetection

R4DecreaseResistanceDetection

R4IncreaseResistanceDPT1

R4DecreaseResistanceDPT1

R4IncreaseResistanceDPT1

R4DecreaseResistanceDPT1

R4IncreaseResistanceDPT1

R4DecreaseResistanceDPT1
Case Study 11: A Brake System Control Unit for a Wheel Braking System of a Hypothetical Aircraft

com_fail_rate
comFailRate : 0.0009E-06
com_fail_effect
comFailEffect1 : "Monitor stuck tripped"
com_detection
comDetection : R6OpenDetection
end

R6OpenDetection in MatraNaturalLanguageStructure, Token with
mls_plain_text
mlsPlainText1 : R6OpenDPT1
end

R6OpenDPT1 in PlainTextNode, Token with
mls_text
mlsText : "Power Supply shut down."
end

R6FailureDescription in ComponentFailureDescription, Token with
com_name
comName : "R6"
com_type
comType : "Film Resistor"
com_fail_mode_description
comFailModeDescription1 :
comFailModeDescription2 :
R5IncreaseResistanceFailModeDescription;
comFailModeDescription3 :
R5DecreaseResistanceFailModeDescription
end

R5IncreaseResistanceFailModeDescription in ComponentFailureModeDescription, Token with
com_fail_mode
comFailMode : "increase resistance"
com_fail_rate
comFailRate : 0.0005E-06
com_fail_effect
comFailEffect1 : "Trip window shifts down"
com_detection
comDetection : R5OpenDetection
end

R5OpenDetection in MatraNaturalLanguageStructure, Token with
mls_plain_text
mlsPlainText1 : R5OpenDPT1
end

R5OpenDPT1 in PlainTextNode, Token with
mls_text
mlsText : "Power Supply shut down."
end

R5DecreaseResistanceFailModeDescription in ComponentFailureModeDescription, Token with
com_fail_mode
comFailMode : "decrease resistance"
com_fail_rate
comFailRate : 0.0004E-06
com_fail_effect
comFailEffect1 : "Trip window shifts up"
com_detection
comDetection : R5OpenDetection
end

Definition of component R7 failure description

R7FailureDescription in ComponentFailureDescription, Token with
com_name
comName : "R7"
com_type
comType : "Film Resistor"
com_fail_mode_description
comFailModeDescription1 :
comFailModeDescription2 :
R7OpenFailModeDescription
end

R7OpenFailModeDescription in ComponentFailureModeDescription, Token with
com_fail_mode
comFailMode : "open"
com_fail_rate
comFailRate : 0.0009E-06
com_fail_effect
comFailEffect1 : "Monitor stuck tripped"
com_detection
comDetection : R7OpenDetection
end

R7OpenDetection in MatraNaturalLanguageStructure, Token with
mls_plain_text
mlsPlainText1 : R7OpenDPT1
end

R7OpenDPT1 in PlainTextNode, Token with
mls_text
mlsText : "Power Supply shut down."
end

R7OpenDetection in MatraNaturalLanguageStructure, Token with
mls_plain_text
mlsPlainText1 : R7OpenDPT1
end

R7OpenDPT1 in PlainTextNode, Token with
mls_text
mlsText : "Power Supply shut down."
end

Definition of component U2 failure description

U2FailureDescription in ComponentFailureDescription, Token with
com_name
comName : "U2"
com_type
comType : "AND Gate"
com_fail_mode_description
comFailModeDescription1 :
U2StuckHighFailModeDescription;
comFailModeDescription2 :
U2StuckLowFailModeDescription
end

U2StuckHighFailModeDescription in ComponentFailureModeDescription, Token with
com_fail_mode
comFailMode : "open"
Tracing Development & Assessment Artifacts

end

U2StuckHighFailModeDescription in ComponentFailureModeDescription, Token
with
com_fail_mode
  comFailMode : "stuck high"
com_fail_rate
  comFailRate : 0.0108E-06
com_fail_effect
  comFailEffect1 : "Monitor stuck valid"
com_detection
  comDetection : U2StuckHighDetection
end

U2StuckHighDetection in MatraNaturalLanguageStructure, Token
with
mnls_plain_text
  mnlsText : U2StuckHighDPT1
end

U2StuckHighDPT1 in PlainTextNode, Token
with
mnls_text
  mnlsText : "Bench test."
end

U2StuckLowFailModeDescription in ComponentFailureModeDescription, Token
with
com_fail_mode
  comFailMode : "stuck low"
com_fail_rate
  comFailRate : 0.0054E-06
com_fail_effect
  comFailEffect1 : "Monitor stuck tripped"
com_detection
  comDetection : U2StuckLowDetection
end

U2StuckLowDetection in MatraNaturalLanguageStructure, Token
with
mnls_plain_text
  mnlsText : U2StuckLowDPT1
end

U2StuckLowDPT1 in PlainTextNode, Token
with
mnls_text
  mnlsText : "Power Supply shut down."
end

Definition of component U3 failure description

U3FailureDescription in ComponentFailureDescription, Token
with
com_name
  comName : "U3"
com_type
  comType : "Voltage Reference"
com_fail_mode_description
  comFailModeDescription1 : U3InopFailModeDescription;
  comFailModeDescription2 : U3OutOfRangeSpecFailModeDescription;
  comFailModeDescription3 : U3ShortFailModeDescription;
  comFailModeDescription4 : U3OpenFailModeDescription
end

U3InopFailModeDescription in ComponentFailureModeDescription, Token
with
com_fail_mode
  comFailMode : "inop"
com_fail_rate
  comFailRate : 0.0110E-06
com_fail_effect
  comFailEffect1 : "Monitor trip"
com_detection
  comDetection : U3InopDetection
end

U3InopDetection in MatraNaturalLanguageStructure, Token
with
mnls_plain_text
  mnlsText : U3InopDPT1
end

U3InopDPT1 in PlainTextNode, Token
with
mnls_text
  mnlsText : "Power Supply shut down."
end

U3OutOfRangeSpecFailModeDescription in ComponentFailureModeDescription, Token
with
com_fail_mode
  comFailMode : "out of spec"
com_fail_rate
  comFailRate : 0.0058E-06
com_fail_effect
  comFailEffect1 : "Window shift"
end

U3ShortFailModeDescription in ComponentFailureModeDescription, Token
with
com_fail_mode
  comFailMode : "short"
com_fail_rate
  comFailRate : 0.0026E-06
com_fail_effect
  comFailEffect1 : "Monitor trip"
com_detection
  comDetection : U3ShortDetection
end

U3ShortDetection in MatraNaturalLanguageStructure, Token
with
mnls_plain_text
  mnlsText : U3ShortDPT1
end

U3ShortDPT1 in PlainTextNode, Token
with
mnls_text
  mnlsText : "Power Supply shut down."
end

U3OpenFailModeDescription in ComponentFailureModeDescription, Token
with
com_fail_mode
  comFailMode : "open"
com_fail_rate
  comFailRate : 0.0245E-06
com_fail_effect
  comFailEffect1 : "Monitor trip"
com_detection
  comDetection : U3OpenDetection
end

U3OpenDetection in MatraNaturalLanguageStructure, Token
with
mnls_plain_text
  mnlsText : U3OpenDPT1
end
6.3.4.3 Fault Tree Analysis - Updated

The results of Failure Modes & Effects Analyses, together with those of other techniques such as Common Mode Analysis are used to update the fault trees developed as part of Preliminary System Safety Assessment. In doing so, evidence is provided for certification demonstrating that safety requirements for undesirable top events have been met.

In this subsection we update failure rates and probabilities for the fault tree in 6.3.3.2, which recall considered as its top event inadvertent braking attributable to the BSCU, and which has a requirement of 2.5E-09. Note the description for this example is the same as given previously.

Rather than repeat what is essentially the same tree but with additional and revised quantitative data, we include a subset of events (figure 6.17) from its upper levels (down to the transfer gates), with the remainder appearing in Appendix D.
Besides this additional and revised quantitative data, the label of event BSCUUNDF has been modified from ‘single undetected . . . .’ to ‘single undetectable . . . .’, and its type from external to basic. In the following subsection, we complete instantation of the Fault Tree Analysis begun in 6.3.3.2.1 by creating updated profiles capturing these modifications for events shown in figure 6.17 (readers are again referred to Appendix D for the complete tree).

6.3.4.3.1 Instantiation of Fault Tree Analysis Meta-Model - Updated Fault Tree

Updated Event Definition ‘BSCUINADD’

EventA in Event, Token with
updated_profile
updatedProfile : EventAUpdatedProfile
end

EventAUpdatedProfile in
UpdatedEventProfile, Token with
type
_Type : "top"
 event_connective
 eventConnective : OrGate1U
 actual_probability
 actualProbability : EventAAActualProbability
 preliminary_budget
 preliminaryBudget : EventAPreliminaryBudget
 actual_label
 actualLabel : EventAPreliminaryLabel
end

EventAAActualProbability in
ActualProbability, Token with
.probability
_Probability : 1.23E-09
end

Gate definitions

OrGate1U in OrGate, Token with input

Updated Event Definition ‘BSCUUNDF’

EventB in Event, Token with
updated_profile
updatedProfile : EventBUpdatedProfile
end

EventBUpdatedProfile in
UpdatedEventProfile, Token with
type
_Type : "bas"
 actual_probability
 actualProbability : EventBAActualProbability
 preliminary_budget
 preliminaryBudget : EventB PreliminaryBudget
 actual_label
 actualLabel : EventBActualLabel
end

EventBAActualProbability in
ActualProbability, Token with
.probability
_Probability : 0.00E+00
end

EventBActualLabel in SimpleLabel, Token with
Failure effects and probabilities from the item level SSA are subsequently used to verify system level analyses (including fault trees), and thence aircraft level analyses. However, these are beyond the scope of this investigation; interested readers are again referred to EUROCAE (1996b) for more details.

6.3.5 Trace Relations

This case study has further demonstrated population of meta-models within the traceability Workspace. Trace relations may again be added to provide means of navigating between these models, and elements of these models, as we now illustrate.

From the artifacts represented, a number of possible associations can be established as shown in figure 6.18 (and summarised in table 6.5):-
Figure 6.18 - 'Exemplar Intra-Micro Horizontal & Vertical Traceability Relations'

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>RELATION TYPE</th>
<th>TARGET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Element</td>
<td>Model</td>
</tr>
<tr>
<td>Power Supply Function Block Diagram</td>
<td>Power Supply Monitor (Function)</td>
<td>Implemented-By</td>
</tr>
<tr>
<td>Power Supply Monitor Circuit Diagram</td>
<td>U1B (Capacitor)</td>
<td>Assessed-By</td>
</tr>
<tr>
<td>Failure Modes &amp; Effects Analysis Table</td>
<td>Monitor stuck valid (Failure Mode Description)</td>
<td>Effects</td>
</tr>
<tr>
<td>Failure Modes &amp; Effects Analysis Table</td>
<td>output open (Failure Mode Description)</td>
<td>Contributes-to</td>
</tr>
</tbody>
</table>

Table 6.5 - ‘Summary of Traceability Relations’ (from figure 6.18)

To clarify the above relations according to the traceability axes from Chapter One (subsection 1.4.3): all are deemed intra-micro since - in ARP 4761 terminology - they describe various views within a single ‘item’ (BSCU) of the Wheel Braking System. Assessed-By and Effects are vertical relations (from implementation to assessment artifacts and from assessment to design respectively), as is Implemented-By (assuming we regard the block diagram as a design artifact and the Circuit Diagram an implementation artifact). Finally, Contributes-to is a horizontal relation among SSA assessment artifacts.

Again, we reiterate that these associations are fairly arbitrary and that a literature survey together with practitioner consultation will be necessary to produce a comprehensive set relevant to the aerospace

---

8 The BSCU may also be regarded as a system in its own right and hence components such as U1, as constituent items of that system.
industry and artifacts from that domain (see further work item in Chapter Seven, subsection 7.4.3).

6.3.5.1 Instantiation of Trace Relations

To demonstrate application of these relations, we instantiate the appropriate O-Telos base classes - specialisations of AerospaceEngineeringAssociation - introduced in Chapter Three (subsection 3.3.6.3.2).

Our first example populates the ImplementedBy relation; its source (fromEntity) is the Power Supply Monitor (PowerSupplyMonitor) function (element) of the Function Block Diagram, and its destination (toEntity), the Power Supply Monitor Circuit Diagram (PowerSupplyMonitorCD).

\[
\text{ImplementedBy01 in ImplementedBy, Token with} \\
\quad \text{from_entity fromEntity : PowerSupplyMonitor} \\
\quad \text{to_entity toEntity : PowerSupplyMonitorCD} \\
\]

Example two populates AssessedBy, in this case between U1B, a comparator element of the Power Supply Monitor Circuit Diagram - and the corresponding description of failures for that particular component (in terms of component, type and failure mode\(^9\)) from the piece-part FMEA table (U1BFailureDescription).

\[
\text{AssessedBy01 in AssessedBy, Token with} \\
\quad \text{from_entity fromEntity : U1B} \\
\quad \text{to_entity toEntity : U1BFailureDescription} \\
\]

Our third example populates the Effects association; it occurs between an instance of failure effect details (i.e., mode, effect, rate, etc.) from the FMEA table (U1BOutputOpenFailModeDescription)\(^{10}\), and the PowerSupplyMonitor function of the Function Block Diagram.

\[
\text{Effects01 in Effects, Token with} \\
\quad \text{from_entity fromEntity : U1BOutputOpenFailModeDescription} \\
\quad \text{to_entity toEntity : PowerSupplyMonitor} \\
\]

Our final example populates the ContributesTo association between details of a failure effect (again U1BOutputOpenFailModeDescription from example three) and a fault tree updated profile (EventHUpdatedProfile). Note, here, using aggregated objects as source and target (i.e., opting for a relatively coarse granularity) adds context to the association (as it does with examples two and three above) which in this case might otherwise simply link rates in the FMEA to those in the fault tree.

\[
\text{ContributesTo01 in ContributesTo, Token with} \\
\quad \text{from_entity fromEntity : U1BOutputOpenFailModeDescription} \\
\quad \text{to_entity toEntity : EventHUpdatedProfile} \\
\]

---

9 This relates to the aggregated ComponentFailureDescription class from the FMEA meta-model in subsection 5.3.3.2.1.

10 This relates to the aggregated ComponentFailureModeDescription class from the FMEA meta-model in subsection 5.3.3.2.1; similar Effects relations apply to the other FMEA failure descriptions shown in figure 6.18 as these also impact on the Monitor.
6.3.6 Summary

This section has demonstrated application of the Fault Tree Analysis and Failure Modes and Effects Analysis structures introduced in sections 5.2 and 5.3 respectively. In doing so, extracts of artifacts described in a case study from an aerospace industry standard (ARP 4761) were used to populate O-Telos implementations of both structures. We then illustrated how these models and their elements may be linked using various trace relations. The results of this case study will again be considered as part of our overall thesis evaluation in Chapter Seven.
6.4 Chapter Summary

This chapter has exhibited a number of traceability structures (meta-models) introduced in Chapters Four and Five using case studies supplied by aerospace practitioners. These, along with worked examples from the previous chapters, provide a flavour of one possible means of automation for MATrA.

Our first case study used specification documents for a Mission Planning System for the Hawk Aircraft to populate the User Centred Requirements Structure (incorporating the MATrA Natural Language Structure). Three Use Case Models were instantiated, along with textual Scenarios and Message Sequence Chart representations of paths through one particular use case. Two further path fragments were included to demonstrate population of model elements for timing events and event grouping. We also featured a small number of example trace associations to show potential ways in which UCRS elements may be linked using the AerospaceEngineeringAssociation concept (from subsection 3.3.6.3.2). The case study confirmed that UCRS is capable of representing commercial specifications expressed in it constituent notations and that traceability linkages can be established between its elements. While acknowledging the implementation yields a large number of O-Telos objects, readers are again reminded that these will be created automatically by the tool2matra function and 'concealed' from users by an appropriate interface.

The second case study used material from Appendix L of ARP 4761\textsuperscript{11} (featuring a braking system for a hypothetical aircraft) to populate the Fault Tree and Failure Modes and Effects Analysis Structures (the latter again incorporating MNLS). Featured artifacts included preliminary and updated fault trees, as well as fragments of functional and piece-part FMEA tables. Again example associations demonstrating potential traceability paths between elements of these structures were also included. The study showed that both the Fault Tree and Failure Modes and Effects Analysis Structures are able to represent, and support traceability across, artifacts expressed in these notations.

We regard the benefits of using MATrA shown by these case studies to be twofold. The first is simply improved traceability; engineers are able to create links, both among elements of individual structures (as shown by the Hawk case study), but perhaps more importantly, between elements of different structures (as demonstrated by the ARP 4761 case study). Given that featured notations are likely to be supported by disparate tools, this was the main aim of the thesis.

The second benefit stems from rigour imposed on developers when encoding their specifications in the MATrA structures. This was particularly true for the Hawk case study which features structures for notations that are not rigorously defined (for instance, use cases and textual scenarios). Thus, given implementation of the well-formedness and PDS consistency checks featured throughout (see future work item 7.4.13), the richness and granularity at which the structures are represented can potentially lead to specifications that are not only more traceable, but also of better quality.

\textsuperscript{11} Several deficiencies exist with the case study material presented in ARP 4761 (see Dawkins et al., 1999). However it is acknowledged as the best published example from the safety literature and is often used for academic and training purposes.
Chapter 7  Conclusions

7.1 Introduction
This chapter summarises the dissertation, highlights its contributions and provides some directions for further work.

7.2 Concluding Remarks
It is useful to begin by revisiting the thesis argument from Chapter One (subsection 1.3) which stated the following:-

Most notations for development, assessment and product management used by aerospace practitioners have (bespoke or commercial) tool support; however a lack of well-defined approaches to integration limits traceability between their respective data sets.

We argue that traceability across tools can be achieved by exporting these data sets to an integrated environment consisting of: i) a Workspace comprising a set of structures or meta-models (literally, models describing models) capturing data elements for a representative set of development, assessment and product management notations; ii) a further structure capturing fundamental elements of the emerging product that maintains consistency within the Workspace (where 'fundamental elements' refers to system components, their functions and behaviour, etc.); iii) well-formedness constraints over these structures; and iv) associations and consistency constraints between the structures.

We also argue that development and assessment information populates four traceability dimensions, of which three record links between project artifacts (conceptualised as a cube), while the fourth relates artifacts across different projects (i.e., cubes) and across product families within the same project. Hence, the structures must provide coverage across four dimensions.

This thesis has defined and demonstrated an approach supporting traceability for the development and assessment of complex aerospace (in particular avionics) systems. Its contribution lies in five areas:-

- A meta-framework defining traceability in the context of aerospace (avionics) systems engineering and a visual conceptualisation of the traceability 'dimensions'.
- Classification of traceability techniques.
- A concrete framework for traceability across artifacts from aerospace (avionics) systems engineering projects.
- Meta-models for a representative set of development notations.
- Meta-models for a representative set of safety assessment and product management notations, together with a structure for configuration (i.e. version) management.

In the following sections we summarise the thesis and draw some conclusions from each of these elements of the research.
7.2.1 Traceability for Avionics: Definitions & Drivers

In Chapter One, we provided a context for the thesis by stating our motivation for investigating traceability for avionics. Essentially this was derived from observations on the characteristic nature of aerospace projects gleaned in part from the literature, but mainly from discussions with practitioners. Properties such as protracted life-cycles, the complexity of products and processes used to develop them, use of fault tolerant architectures and the need for certification collectively challenge the provision of effective traceability within that industry.

Most existing work on traceability stems from software engineering literature. This is evident from definitions of the term that are couched in a language which undermines their application to systems engineering. For instance, they are often hardwired to an implied ‘waterfall’ model, explicitly state artifact types with a software orientation and fail to address the issues involved in tracing across system and component descriptions.

Understanding of the term ‘traceability’ has moved on considerably in the last ten years. Most authors are now agreed that we can differentiate aspects of traceability by considering such factors as the direction and type of artifact being traced. It is therefore common to refer to forward and backward traceability in describing navigation to ‘up-stream’ and ‘downstream’ artifacts. Likewise the notion of horizontal and vertical traceability describes facilities for navigating between artifacts of the same and different types respectively.

While traceability must surely be a basic necessity for all but the most trivial of engineering endeavours, work investigating means for its realisation has mainly surfaced since the 1990s. We attributed this general growth to two factors, the emergence of requirements engineering as a topic of interest and increased use of compliance frameworks (specifically, process improvement models and standards and guidelines).

7.2.1.1 Contribution

From our observation that existing definitions of traceability do not reflect the complex structure of aerospace (including avionics) products and processes, we proposed an appropriate meta-framework. This was based on a definition taxonomy in which we refined the notion of horizontal and vertical traceability by introducing the qualifiers micro and macro to differentiate traceability within and across decomposition levels (e.g., system, sub-system, component), and the qualifiers intra and inter to distinguish traceability within and across system descriptions (i.e., systems that interact with one another). We also recognised traceability of revisions and variants through separate definitions in preference to overloading the horizontal definition (as is normally the case).

To provide a visual conceptualisation, horizontal, vertical and revision traceability were represented as dimensions of a cube. This novel idea was extended to variant traceability (i.e., the variant dimension) by depicting traceability between cubes. The cube abstraction intuitively conveys salient features of
traceability that are not well-characterised by existing two-dimensional images which tend to disregard revisions and variants, usually as a corollary of failing to distinguish them from the horizontal dimension.

7.2.2 Modelling Concepts, Techniques & Tools
Chapter Two provided an overview of traceability techniques which we considered from three perspectives, namely structure (e.g., support for typing of artifacts and relationships, and for abstraction concepts such as generalisation and aggregation), integrity (e.g., means to specify rules and constraints over constituent elements of artifacts) and manipulation (e.g., means for selective information retrieval). For reader orientation, an overview of the UML, OCL and O-Telos representation techniques employed in this thesis was also provided. The chapter concluded by investigating various proprietary and commercial tools supporting the techniques considered. These included DOORS which we noted is moving towards the MATrA Workspace approach of transferring data from CASE tools onto information models expressed in a common language (termed DOORS 'surrogate' modules) and creating links between these models to support traceability.

7.2.2.1 Contribution
A means for classifying traceability techniques according to their basis of representation was proposed. Specifically, we differentiated between approaches employing higher-order and lower-order cross referencing (both of which are founded on graphs and matrices) from those using a data-modelling approach. The key difference between this classification scheme and others stems from its starting point - the premise that set theory is the mathematical foundation of all traceability techniques and so the level of sophistication afforded by a particular technique depends on what additional concepts (in terms of facilities for structure, integrity and manipulation) it adds to basic set theoretic constructs. This contrast with other classifications such as Gotel (1995) which do not divorce the underlying principles from their means of realisation.

7.2.3 MATrA: Towards A Concrete Framework for Traceability
Chapter Three proposed a concrete traceability framework for avionics engineering (a realisation of the dimensions meta-framework). Several principles were identified as ‘fundamental’ to this framework, while a review of the literature revealed existing work that provided a ‘foundation’ for its development.

Framework principles included a Workspace of notation dependent structures (meta-models) representing project data transferred from CASE tools and linkages between this data; a Meta-class model providing a common underlying representation for Workspace meta-models; a mapping function (tool2matra) to allow data transfer from CASE tools to the Workspace; the Product Data Synthesis, a notation independent structure representing fundamental elements of the emerging system (e.g., components, functions, behaviour and means for their association) which maintains Workspace consistency by preventing bad data from entering via tool2matra (i.e., preventing data being mapped to a notation dependent structure) unless the PDS contains corresponding data elements, and by
Conclusions

preventing violations once the data is ‘inside’; and the Framework Model that provides a basis for managing framework elements and their inter-relationships. The main foundations of these principles were ESPRIT project NATURE (Pohl, 1996), ESPRIT project SEDRES (Johnson, 1997) and the Design Rational Capture System (Klein, 1993a).

In particular, we were influenced by NATURE’s use of a conceptual modelling language to structure object-based representations of requirements notations for IS Development. Accordingly, we adopted a similar approach to developing Workspace meta-models for design, implementation, safety analysis and product management notations; the aim being to provide a seamless environment for traceability that is unconstrained by tool boundaries. We were also influenced by NATURE partners use of a common meta-level structure as is evident in the MATrA Meta-class model, and by their approach to creating associations between the various meta-model representations.

SEDRES’ main influence on MATrA was in providing evidence of the ability to map data from CASE tools onto information models. This enabled us to treat Workspace population as a ‘black-box’ through the interface to an undefined function (tool2matra) and to concentrate on other aspects of the traceability framework. The symmetry with DOORS’ notion of surrogate modules was again cited as further evidence of feasibility.

The main influence of the Design Rational Capture System on MATrA has been development of the Product Data Synthesis which takes as its starting point the Artifact Synthesis structure component of DRCS. Extensions were necessary however as the Artifact Synthesis lacks elements to record functional architecture and behaviour. The literature provided a basis for developing these extensions, in particular, work by Oliver (1994).

Chapter Three also stressed that the emphasis of this thesis was to be on development and validation of Workspace meta-models for a representative and diverse set of notations. Selection was guided not only by their relevance to avionics engineering, but also by a desire to include different representation formats (textual, graphical, tabular and program code) and hence address the different modelling challenges presented by each. Featured notations therefore included a Natural Language Structure, a User Centred Requirements Structure, a Real-Time Network Specification Language (RTN-SL) Structure, a SPARK Ada Structure, a Fault Tree Analysis (FTA) Structure, a Failure Mode and Effects Analysis (FMEA) Structure and a Programme Evaluation & Review Technique (PERT) Structure. For reader orientation, we used a further notation (Circuit Diagrams) to show how work on these structures would be presented in Chapters Four and Five.

7.2.3.1 Contribution

MATrA has integrated and extended a number of existing works towards a concrete traceability framework for avionics engineering. What sets it apart from the majority is consideration of traceability between notations. Besides SEDRES (through its support for data exchange), NATURE and to some
extent SAM (Wilson & McDermid, 1995), no other approach has addressed this issue. Moreover, NATURE concentrates on the requirements phase, functions independently of CASE tools and ignores consistency between notations. SAM meanwhile has an even narrower focus, offering support for traceability as a by-product of structuring descriptions of safety arguments for critical systems.

In addition, we are unaware of any other work addressing all four traceability dimensions 'in concert' as MATrA does; the framework as introduced in Chapter Three provided support for horizontal and vertical traceability which we then extended in Chapter Five (see 7.25) for the revision and variant dimensions.

7.2.4 Meta-models for System Development

As per the stated emphasis of our work, Chapter Four proposed meta-models for representing artifacts expressed in Natural Language, as Use Case Models, Scenarios and Message Sequence Charts via the User Centred Requirements Structure, in the Real-Time Network Specification Language and in SPARK Ada. In each case we described factors motivating their inclusion, introduced the basic concepts, proposed a meta-model in UML with well-formedness and PDS consistency checks expressed in OCL and gave a ‘flavour’ of tool support by implementing the base classes in O-Telos (using ConceptBase). We further provided worked examples instantiating these base classes, or else committed to a more detailed case study in Chapter Six.

7.2.4.1 Contribution

The MATrA Natural Language Structure is based on an approach to representing textual artifacts proposed by project NATURE and supports fine grained traceability at a user-determined granularity. Its main role in this thesis has been to augment the design of other structures (e.g., to represent pre and post-conditions of the Use Case Model in subsection 4.3 and the detection column for FMEA tables in 5.3), the idea being to replace a standard String type with an MNLS wherever PDS elements embedded in a prose statement must be promoted to primitives for traceability purposes. The introduction of node typing to enable PDS consistency checks on these primitives makes MNLS fundamentally different to the NATURE structure, whilst adding to its practicality. It also differentiates our work from functionally similar facilities offered by the DOORS and RTM tools. MNLS was evaluated by case studies in Chapter Six (see 7.2.6).

The User Centred Requirements Structure captures syntactic elements of three complementary requirements elicitation, analysis and documentation notations: Use Case Models, Scenarios and Message Sequence Charts. UCRS is entirely novel and was devised from observations on usage of these notations in actual commercial specifications. Accordingly, the structure allows management of, including the sharing of elements across, multiple Use Case Models. Paths through each use case are described in terms of interactions using textual Scenarios and/or Message Sequence Charts (the former expressed using a specialisation of MNLS), both of which share a common underlying representation on the basis that they contain the same core elements (allowing generation of MSCs from Scenarios and
Conclusions

Formal conditions were also defined to preserve PDS consistency, usage restrictions and well-formedness, as well as consistency within the UCRS itself (i.e., between the three constituent notations). Again, the User Centred Requirements Structure was evaluated by case study in Chapter Six (see 7.2.6).

To demonstrate application of meta-modelling to a graphical design notation, we developed a novel structure representing artifacts expressed using the Real-Time Network Specification Language (RTN-SL) in which concurrent processing components exchange information and synchronise through shared data in the connections. All key syntactic elements are represented by the structure, although we confined definition of constraints to the principal restrictions needed to ensure an RTN-SL specification is internally consistent; a number of PDS consistency checks were also stated. The RTN-SL structure was then evaluated by implementing a number of (mostly hypothetical) worked examples in O-Telos using ConceptBase. Whilst providing proof-of-concept in that the structure allows us to represent (and hence with the insertion of appropriate linkages, trace to and from) artifacts represented in this particular language, we concede that some further evaluation using actual commercial specifications is desirable simply to increase confidence in these findings and to determine scalability (see 7.4.8).

Finally, to demonstrate how meta-modelling can be applied to the representation of software source code, we presented a novel structure capturing a subset of the concrete (well-formedness) syntax for SPARK Ada, a programming language used to develop safety-critical systems. The structure was developed in parallel with a modelling philosophy - a set of guidelines for representing languages with a formal grammar - and was presented as a series of schemas showing the source BNF (Backus Naur Form) syntax, corresponding UML Class Diagram and its implementation in O-Telos. The structure was evaluated by expressing worked examples of two source code fragments in ConceptBase. These successfully showed the ability to represent our chosen SPARK Ada subset and hence with the insertion of appropriate linkages, trace to and from its constituent elements. While use of a larger subset and worked examples would have provided further proof of concept, we felt there is little to be gained from this other than perhaps in terms of assessing scalability (again see 7.4.8). Finally, to evaluate versatility of the above-mentioned guidelines, we successfully applied them to development of a further structure representing a small subset of the RTN-SL textual syntax. This showed that the guidelines can potentially be used in extending MATrA to support any formal language expressed in BNF.

7.2.5 Meta-models for Safety Assessment & Product Management

Continuing the theme of our emphasis, Chapter Five introduced meta-models for expressing results of Fault Tree Analysis and Failure Modes and Effects Analysis, as well as artifacts represented using the Programme Evaluation & Review Technique. Again, we described for each the factors motivating their inclusion, introduced the basic concepts, proposed a UML meta-model with rules and constraints expressed in OCL and provided a foundation for tool support by implementing the base classes in O-Telos using ConceptBase. We further included a worked example of each meta-model, or else resolved to do a more detailed case study in Chapter Six. Finally, to support traceability of revisions and variants...
The MATrA Configuration Model was introduced.

7.2.5.1 Contribution
To demonstrate how meta-modelling can be applied to graphical safety analysis techniques, we developed a novel Fault Tree Structure which captures results of Fault Tree Analysis, a deductive approach to determining and relating all events which may lead to a failure condition. The structure was guided by observations on usage of fault trees, notably in the context of ARP 4761 guidelines for certification of civil aircraft. Therefore in addition to supporting the main concepts underpinning Fault Tree Analysis, the structure enables representation and management of preliminary trees expressing budget safety objectives, together with their updated counterparts showing realisation of these requirements. To remove duplication of data between the two trees whilst helping to maintain consistency and increase traceability, the Fault Tree Structure permits sharing of events (or more specifically, elements of their descriptions) through the notion of 'event profiles'. In addition, well-formedness constraints (building on earlier work in Mason & Saeed, 1998) were expressed over the logical tree structure, together with safety-criteria identifying common cause and single failures. We further introduced a 'light-weight' approach to the formalisation of event labels by typing events according to a taxonomy from the literature and by specifying their content at a level of granularity that allows for their verification against the PDS. The Fault Tree Structure was evaluated by case study in Chapter Six (see subsection 7.2.6).

To demonstrate how meta-modelling can also be applied to tabular safety analysis techniques, we developed a novel structure for expressing the results of Failure Modes and Effects Analysis, an inductive approach to tracking the effects of failures within a system and determining their consequences. The structure supports recording of results for both functional and piece-part (i.e. component) analyses, while its content is consistent with the suggested worksheet formats detailed in ARP 4761; a number of formal checks verifying FMEA elements against the Product Data Synthesis were also defined. Again, the Failure Modes and Effects Analysis Structure was evaluated by case study in Chapter Six (see 7.2.6).

Finally, (for completeness and) to demonstrate how meta-modelling can be applied to the representation of product management structures, we proposed a novel structure capturing the graphical notation for the Programme Evaluation & Review Technique which represents inter-relationships between the timing of events and project activities in the form of a network. Again we captured the main syntactic elements and principal well-formedness restrictions. The structure was evaluated using hypothetical worked examples implemented in O-Telos using ConceptBase, one of which we included in the main text and which provides proof of the ability to represent artifacts expressed in this notation.

To demonstrate another aspect of product management, the MATrA Configuration Model was proposed (and the Framework model from Chapter Three extended) to manage revisions and variants across the Workspace and Product Data Synthesis. The model is based on an intertwined approach (Conradi &
Conclusions

Westfechtel, 1998) and features elements to represent configurations and change deltas, as well as means to record various versioning related dependencies; rules were defined for deriving deltas and constraints to ensure correct instantiation of dependencies. A lengthy worked example successfully demonstrated aspects of the Configuration Model, including realisation of the intertwined approach: in particular, articulation of revisions to product structures within the PDS, the ability to establish dependencies between PDS and Workspace elements, to create configurations, perform impact analysis and to effect change based on this analysis. As previously stated, no other work has sought to address the four dimensions as a whole and hence to integrate configuration management and traceability as called for by Ramesh & Jarke (1999).

7.2.6 Application of the MATrA Framework

To evaluate aspects of the framework not demonstrated by worked example, Chapter Six featured two in depth case studies, both based on material supplied by aerospace practitioners.

Our first case study used data from a commercial specification supplied by BAE SYSTEMS to instantiate a ConceptBase implementation of base classes for the User Centred Requirements Structure. Specifically, we populated three Use Case Models, together with textual Scenario and Message Sequence Chart paths (both normal and exceptional) through one particular use case. In addition, fragments of two further paths were populated to show instantiation of additional concepts for representing timing events and event grouping. A small number of example associations between elements of the UCRS were also introduced to demonstrate traceability among the constituent notations (we return to this subject in subsection 7.4.3). The study therefore provided proof that UCRS is capable of representing commercial specifications expressed using its combination of notations, and of supporting traceability.

In the second case study, material from an example featured in Appendix L of ARP 4761 was used to populate ConceptBase implementations of the Fault Tree and Failure Modes and Effects Analysis Structures. Specifically, preliminary and updated fault trees were created, together with fragments of functional and piece part FMEA tables, with the latter expressed over elements instantiating the Circuit Diagram meta-model introduced in Chapter Three. Further examples of associations were also included to indicate traceability between elements of these structures (again, see subsection 7.4.3); the fact that the associations linked data originating from (potentially) disparate source tools underlines the seamless nature of the Workspace. Again, the study provided proof that the featured meta-models are able to represent and support traceability across data from commercial artifacts expressed in these notations.

In addition, both case studies showcased the MATrA Natural Language Structure, the former in representing pre and post-conditions of the Use Case Model and the latter, the comment and detection columns for FMEA tables. To demonstrate its versatility, a range of text strings were successfully converted to ‘MNLS form’, with significant primitives extracted accordingly.
Conclusions

7.2.7 Overall Contribution

As stated throughout, this thesis has concentrated on meta-modelling aspects of the MATrA framework. Aside from claims made in subsections 7.2.4.1 and 7.2.5.1 on the contribution made by these structures, discussions with practitioners indicates that the practice of developing such models can itself prove beneficial. That is, the investigative process involved in producing a MATrA meta-model can lead to better understanding of the notation it represents, or perhaps more importantly, the particular usage of the notation for a development or assessment activity. This is especially true for notations that are less rigorously defined but which offer flexibility as a result.

However, the main purpose of the thesis has been to provide an environment enabling traceability among the data sets of tools, along the dimensions introduced in Chapter One. Therefore the fundamental gain from the notation dependent meta-models is their ability to be linked to form navigable paths through the Workspace (note that 'dimension-wise', the type of traceability existing between two meta-models depends entirely on when and for what purpose they are being used relative to the development and assessment process). Furthermore, by identifying means by which to make CASE tools part of MATrA and by providing a way of increasing confidence in the consistency of data transferred from these tools (using a common set of typed primitives - e.g., module, function, condition, etc.), we have improved industrial practicality of the framework. Therefore, this thesis can justifiably claim to have maintained the argument stated by our position in section 1.3 (and re-iterated in 7.2).

7.3 Limitations

We are aware of certain limitations in our work, as well as in the confidence and validity of some claims made with respect to it. These are as follows:-

- Absence of a tool2matra function
- Duplication of CASE tool data in the Workspace
- Object proliferation

7.3.1 Absence of a tool2matra Function

The main limitation of work presented here relates to the fact we do not have in place a function to map data from CASE tool data structures onto MATrA Workspace meta-models. This potentially undermines confidence in the framework itself and material presented to support its validation. However, given that DOORS, a leading commercial traceability tool now employs a similar (although less rigorous) approach and given that reputable public domain evidence from SEDRES partners has addressed the technical issues involved, we suggest the feasibility of this aspect of MATrA is beyond reasonable doubt.

7.3.2 Duplication of CASE Tool Data in Workspace

The fact that CASE tool data is replicated in the Workspace is potentially a further limitation, especially given the issue of scale alluded to throughout. Vendors of the DOORS tool (which is widely used in the aerospace domain) would claim that this is a price developers are prepared to pay in return for the
improved traceability across tools that results. Moreover, we are proposing to investigate as a future work item a possible alternative which restricts persistent storage of Workspace elements to trace associations (see subsection 7.4.11).

7.3.3 Object Proliferation
Related to the previous point, it has been suggested that instantiation of MATrA structures yields an overly large number of objects (cf. subsection 6.2 and 6.3.) To counter such claims, we assert that the relationship between elements of notation dependent (i.e. meta-model) and CASE tool data structures is linear rather than exponential, that population will be automated by means of tool2matra and that irrespective of these issues, all such objects will be concealed from users beneath a graphical interface.

7.4 Further Work
During the course of our research, several areas worthy of further investigation were identified. These include the following:-

- Extension of Workspace notations
- Application of MATrA to other safety-critical domains
- Survey of domain requirements for trace associations
- Enriching the Product Data Synthesis
- Specifying requirements using the MATrA Natural Language Structure
- Systems Engineering process issues for MATrA
- Investigation of an inverse mapping function (matra2tool)
- Contiguous case study across all dimensions
- Investigation of analysis objectives
- Use of tool2matra to optimise Workspace revisions
- Confinement of persistent Workspace to trace associations
- Incorporation of standards knowledge into MATrA
- Use of a commercial tool as a basis for implementing MATrA
- Re-expression of structures in EXPRESS

In the following sections we provide a brief introduction to possible directions of further work in the areas identified above.

7.4.1 Extension of Workspace Notations
To maintain a tractable scope, the Workspace only featured a small (if representative) subset of notations and techniques used by aerospace practitioners. Clearly, to provide a practical engineering environment this subset must be increased. For example we could include a formal notation such as VDM (Jones, 1990) or Z (Spivey, 1989), both of which can potentially be represented by applying the modelling philosophy alluded to in subsection 4.5.3.1.1. Other candidates (some of which were developed but omitted due to space limitations) are Viewpoints (Mullery, 1979) and Piping & Instrumentation Diagrams for development (Turton et al., 1997), Event Tree Analysis (Villemeur,
1992) and Goal Structure Notation (Wilson et al., 1995) for safety and Responsibility Modelling
(Dobson & Strens, 1994) for product management. Moreover, given that we considered a diversity of
notations, both in terms of purpose (requirements, design, implementation, safety and product
management) and means of representation (natural language, tabular and graphical), the existing set of
structures provides an instructive source of reference for anyone seeking to extend the Workspace.

7.4.2 Application of MATrA to Other Safety-Critical Domains
While development of MATrA has been oriented towards avionics engineering, some of the industry
characteristics enumerated in Chapter One can be said to apply to other sectors. It would therefore be
worthwhile investigating the potential application of MATrA to other domains utilising complex
dependable systems, in particular the Nuclear and Rail industries. Prima facie, this is likely to involve
expansion of notations supported by the Workspace, although specific requirements would need to be
established through discussions with practitioners and a review of current literature, including standards
and guidelines.

7.4.3 Survey of Domain Requirements for Trace Associations
Despite considering a mechanism for creating relationships between Workspace meta-models, only a
small number of arbitrary examples were included in the thesis, their purpose being to demonstrate the
mechanism itself, rather than any specific traceability gain. We did however highlight the need to
compile (through literature review and practitioner consultation) a comprehensive set of associations for
relating requirements, design, implementation and other artifact types. Factors to consider include the
specific types of notations to be related and the processes that apply them. A useful starting point for
the latter was provided in subsection 1.2 (C2); recall our heuristic that regards the process of
transforming an input artifact to an output artifact as an abstraction of a traceability association between
the two.

7.4.4 Enriching the Product Data Synthesis
To increase utility of the Product Data Synthesis, consideration should be given to enriching the
structure. In particular, the following have been suggested to the author:-

- Predefined Aerospace Types
Following a SEDRES approach (Herzog & Scerri, 1998), benefits could be gained from separating PDS
definitions from their instances. This would reduce the modelling workload of developers and ensure
'like' profiles are both consistent and complete. It could be accomplished through an additional layer of
abstraction whereby build elements are promoted to meta-level classes whose instances define standard
modelling components. This is perhaps best explained by example.

At present, the existing Module class could be instantiated as the object TrimTank, with properties such as
Capacity and NormalWorkingPressure. We could also define an InnerTransferTank, an OuterTransferTank, a
CentralTank and so on, each with the same set of properties, but with variances in their respective
parameter values. However, such an approach means engineers would have to replicate the same basic
profile each time they introduced a new tank, with all the attendant risks of inconsistency that this form of action entails.

Alternatively, a generic FuelTank (with the properties previously stated) could be specified as an instance of the Module meta-class (and conceivably a specialisation of some Tank superclass) whose instances - TrimTank, OuterTransferTank and CentralTank - would automatically share a common definition profile. This principle could be extended to other 'standard' modules which continuing the fuel system theme, might include piping, valve and pump types.

Such an approach is only made possible by concentrating on a specific domain, and one which by nature is instinctively conservative. The resulting set of 'Predefined Aerospace Types' would enable MATrA to operate over standardised definitions, which in turn provides increased scope for analysis. Moreover, the cost of establishing such definitions could be offset against the benefits of reusability.

- **Spatial Relations**
  The set of build associations introduced in subsection 3.3.5 may be extended to include spatial-relations. These would be used for example, to integrate MATrA with CAD tools and to define properties associated with fault tolerance, such as segregation (maintenance of independence through use of a physical barrier between components) and separation (maintenance of independence by means of physical distance between components). A preliminary literature survey suggests a number of spatial relations may be of use; these are shown in Appendix E.

7.4.5 Specifying Requirements using the MATrA Natural Language Structure
An obvious use of the MNLS not considered in this thesis concerns the specification of textual requirements. Initial thoughts suggest that with some minor modifications, the structure could be used to express and partially validate certain properties of such statements. For instance, that they contain an imperative (such as must or shall) and that the singularity condition holds; i.e., the requirement is unambiguously expressed over one build element, and that said build element exists in the Product Data Synthesis. Moreover, were we to enrich the PDS with Predefined Aerospace Types in the manner described in 7.4.4, then the validity of values assigned to certain properties could be range checked.

7.4.6 Systems Engineering Process Issues for MATrA
Any traceability framework will be ineffectual without a well-defined mechanism for putting it into practice. MATrA raises several issues pertaining to its integration within a systems engineering environment that such mechanisms need to address. For instance, who should be given 'design authority' for managing and maintaining the Product Data Synthesis, when and how often should tool2matra be used to synchronise the CASE tool and Workspace data-sets and how are exceptions arising from this synchronisation to be dealt with. It is important to remember that traceability cannot be treated as a 'bolt-on'; as such, these and other process issues must be addressed before MATrA can be put to practical use.
7.4.7 Investigation of an Inverse Mapping Function (*matra2tool*)

To further integrate MATrA with the tool environment, an inverse function (*matra2tool*) capable of mapping data from the Workspace onto CASE tool data structures could be investigated. This would allow for limited editing to be performed within the Workspace itself, with results uploaded to the corresponding tool.

Early thoughts on definition of the function suggest the interface would take as its input parameters, an un-populated CASE tool data structure (uCDS) with a corresponding populated notation dependent structure (pNDS) and the Product Data Synthesis (PDS), and return a populated CASE tool data structure (pCDS).

\[ \text{pCDS} \ matra2tool \ (\text{uCDS}, \text{pNDS}, \text{PDS}) \]

Following our discussion in 7.4.6, it should be noted that this facility raises a number of additional process issues which would also need to be considered (e.g., alignment of the CASE tool and Workspace data sets).

7.4.8 Contiguous Case Study Across All Dimensions

Each of the structures considered in this thesis has been validated either through worked examples, or else by a more in-depth case study (mostly conducted using material provided by practitioners from the aerospace industry). However, for a more thorough test of its usefulness (particularly in terms of scalability) the MATrA framework needs exposure to a contiguous example spanning all the traceability dimensions. This would call for the representation and linking of requirements, design, implementation, safety assessment and product management artifacts (including revisions and variants) at aircraft, system, item and hardware/software levels for two or more interacting systems. Only then could we begin to gauge the potential for practical application of MATrA.

7.4.9 Investigation of Analysis Objectives

The traceability Workspace and Product Data Synthesis are both potentially rich sources of information; in order to realise this potential, means are required for its extraction. These may include, for example, mechanisms supporting impact analysis to determine the effects of change. However, given that MATrA structures have a formal foundation and therefore offer such broad scope for analysis, it would be worthwhile conducting a practitioner consultation to identify precise analysis needs.

7.4.10 Use of *tool2matra* to Optimise Workspace Revisions

While introducing the Configuration Model in subsection 5.5, it was remarked upon that ‘equivalent’ elements across revisions to Workspace models are actually represented by different objects. This is due to the assumed mapping from CASE tools into MATrA, which because of its theoretical treatment throughout, is intentionally simplistic. Thus for two models \( M_1 \) and \( M_2 \) containing objects \( Ob_1 \) and \( Ob_2 \) respectively, where \( M_2 \) ‘succeeds’ \( M_1 \) and \( Ob_2 \) represents the same build element as \( Ob_1 \) (without modification), there clearly exists some redundancy. However, consideration could be given to
development of a more sophisticated function capable of reusing the same object in more than one revision. Given the limitations discussed in subsection 7.3.3, this appears to be an area with potential for relieving the Workspace 'overhead'.

7.4.11 Confinement of Persistent Workspace to Trace Associations
Continuing the optimisation theme, it would be interesting to consider the possibility of restricting persistent storage of Workspace elements to trace associations. Such a possibility is motivated by observations that the additional trace data (i.e., dependencies among models and their elements) is what actually differentiates the CASE tool and Workspace data sets. First impressions suggest tool2matra would need to ensure that whenever the Workspace became active, 'existing' elements were recreated (by repeating the mapping of CASE tool data onto corresponding meta-models) using the original object identifiers. This would allow instances of AerospaceEngineeringAssociation subtypes to be 'reunited' with the correct from and to objects.

7.4.12 Incorporation of Standards Knowledge into MATrA
Adherence to development and assessment processes set out in various standards is an important issue within the aerospace domain as evidence supporting claims of compliance often forms part of the safety argument. It has therefore been suggested that the potential for managing such processes within MATrA be explored, in particular the ability to associate and (possibly) perform checks against, individual process steps, the requirements on the data they are to produce and the Workspace artifacts that satisfy these criteria. Moreover, where a standard mandates use of a particular design feature (e.g., dual independent channels), these requirements may be linked to the appropriate PDS elements discharging them.

7.4.13 Use of a Commercial Tool as a Basis for Implementing MATrA
The functionality provided by commercial traceability tools (e.g., DOORS) has moved on rapidly during the course of our research. Hence, while ConceptBase was used in this thesis to provide a 'flavour' of automation, it would be worth investigating use of commercial tools as a basis for developing MATrA further, notably to enable implementation of the rules and constraints stated in Chapters Three to Five. Therefore apart from being able to represent the structures themselves, the richness of set and boolean operators (or a procedural extension language capable of replicating them) provided by a tool is of particular interest; a lack of such facilities (relative to OCL) prevents many formal aspects of MATrA from being realised in ConceptBase.

7.4.14 Re-Expression of Structures in EXPRESS
A first step towards development of a tool2matra mapping function would be to re-express the structures described in this thesis using EXPRESS (see subsection 2.2.2.2.7). EXPRESS was developed as part of the STEP framework (introduced in subsection 3.2.2) which provides a tool neutral format for data representation and exchange. It would of course be necessary to determine means of translating data from the CASE tools into this neutral format and thence to a format interpretable by the target
MATrA tool. However, as work by Riddle (2000) demonstrates, the actual mechanics of re-expressing current object-based representations of the structures in EXPRESS is unlikely to pose a problem.

7.5 Epilogue
We conclude that work in this thesis has been shown to provide the basis for a framework enabling traceability between CASE tools used by avionics engineers. The framework supports traceability across four dimensions using an integrated set of structures whose development and evaluation formed the main strands of our research. These structures are among the core principles on which the framework is based; the other principle relates to a function for populating a subset of framework structures using data transferred from CASE tools. Work towards realisation of this principle was not possible within the confines of a Doctoral programme; instead we identified credible evidence from current literature to support its feasibility. It is hoped the framework presented will contribute to the furtherance of traceability, both generally and within the aerospace industry.
References

Airbus, 2001 Airbus Industrie - AM2085 : Method for Common Airbus Requirements Engineering (CARE)


Ascent, 1997 Ascent Logic - RDD-100 Marketing Material, Ascent Logic Corporation, Trinity Court, Batchworth Island, Church Street, Rickmansworth, Hertfordshire WD3 1RT, UK; obtained from http://www.alc.com


BAe, 1999 British Aerospace Defence Limited : Military Aircraft & Aerostructures - Top Level Requirements for a Mission Planning System for the Hawk Aircraft, BAe-BSY-DR-HWK-000194

Bailin et al., 1990 Bailin, S. C., Moore, J. M., Bentz, R. & Bewtra, M. - KAPTUR: Knowledge Acquisition for Preservation of Tradeoffs and Underlying Rationales, Proc. 5th Conf. on Knowledge Based Software Assistant, Liverpool NY, Sep.


Barbier, 1994 Barbier, F. - Traceability in the Object-Oriented Software Life Cycle, Proc. 13th Int'l Conf. on Technology of Object-Oriented Languages, Versailles, France, pp. 293-301

References

Deductive Object-Oriented Database System, Information Systems, 20(3), pp. 185-211

Barnes, 1996
Barnes, J. - Programming in Ada 95, Addison-Wesley

Barnes, 1997
Barnes, J. - High Integrity Ada: The Spark Approach, Addison Wesley

Belford & Taylor, 1976

Bell et al., 1977

Bellagamba et al., 1993

Bersoff & Davis, 1991

Bhattacharyya, 1992

Bigelow, 1988

Boehm, 1976

Boehm, 1981
Boehm, B. - Software Engineering Economics, Prentice-Hall

Bohner, 1995
Bohner, Shawn Anthony - A Graph Traceability Approach For Software Change Impact Analysis, George Mason University, USA, Ph.D. Thesis

Booch, 1994
Booch, G. - Object-Oriented Analysis and Design with Applications, Benjamin/Cummings

Booth, 1993

Börstler & Janning, 1992

Börstler, 1994

Börstler, 1996

Bosch, 1998
Bosch, Jan - Design Patterns as Language Constructs, Journal of Object-Oriented Programming, 11(2), pp. 18-32

Boyd, 1993

Brodie, 1984

R-2
<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Date</th>
<th>Title</th>
<th>Journal/Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buhr, R. J. A.</td>
<td>1995</td>
<td>Use Case Maps: A New Model to Bridge the Gap Between Requirements and Design, Contribution to the Use Case Workshop at OOPSLA '95</td>
<td>Austin, TX, USA, Oct.; obtained from <a href="http://www.sce.carleton.ca/ftp/pub/UseCaseMaps/">http://www.sce.carleton.ca/ftp/pub/UseCaseMaps/</a></td>
</tr>
<tr>
<td>Buzan, T.</td>
<td>1989</td>
<td>Use Your Head (Revised Edition)</td>
<td>BBC Books</td>
</tr>
<tr>
<td>Carré et al.</td>
<td>1990</td>
<td>SPARK - The Spade Ada Kernel - 3rd ed.</td>
<td>Program Validation Limited</td>
</tr>
<tr>
<td>Casemore, K. J.</td>
<td>1998</td>
<td>Study of a CASE Tool to Support Traceability for a Dependable Avionics System</td>
<td>Final Year Project Dissertation, Department of Computing Science, University of Newcastle upon Tyne</td>
</tr>
<tr>
<td>Chandrasekaran et al.</td>
<td>1993</td>
<td>Functional Representation as Design Rationale</td>
<td>IEEE Computer, Jan., pp. 48-56</td>
</tr>
<tr>
<td>Chen, S. et al.</td>
<td>1993</td>
<td>Database Requirements For A Software Engineering Environment</td>
<td>Information &amp; Software Technology, Mar., pp. 149-161</td>
</tr>
<tr>
<td>Choi &amp; Scacchi</td>
<td>1989</td>
<td>Assuring the Correctness of Configured Software Descriptions</td>
<td>Software Engineering Notes, Nov., pp. 66-75</td>
</tr>
<tr>
<td>Chung et al.</td>
<td>1995</td>
<td>Using Non-Functional Requirements to Systematically Support Change</td>
<td>Proc. 2nd Int'l Symposium on</td>
</tr>
</tbody>
</table>
References

Cimitile et al., 1992

Coallier, 1994

Cockram et al., 1998

Codd, 1970
Codd, E. F. - A Relational Model of Data for Large Shared Data Banks, Communications of the ACM, 13(6), Jun., pp. 377-387

Cogdell, 1999
Cogdell, J. R. - Foundations of Electric Circuits, Prentice Hall

Collofello & Vennergrund, 1987

Conklin & Yakemovic, 1991

Conklin, 1987

Conradi & Westfechtel (1998)

Cook & Daniels, 1994

Corriveau & Hayashi, 1994

Corriveau, 1996
Corriveau, Jean-Pierre - Traceability For Large Object-Oriented Projects, IEEE Computer, Sep., pp. 63-68

Coyne, 1993

Cross, 1996
Cross, Gary - Tracking the Changing Face of System Development, Real-Time Magazine, 96/1, pp. 10-12

Cullyer et al., 1991

Curran et al., 1994

Cybulski & Reed, 1992

Czejdo et al., 1992
Czejdo, B., Emsley, D. W. & Rusinkiewicz, M. - View Updates for an Extended Entity-Relationship Model, Information Sciences, No. 62, pp. 41-64

Dardenne et al., 1993
References

Date, 1995 - Date, C. J. - An Introduction to Database Systems, 6th edition, Addison-Wesley


Dobson & Strens, 1994 - Dobson, J. E. D. & Strens, R. - Enterprise Modelling as a Technique for Organisational Requirements Definition, Intelligent Systems Engineering, Spring, pp. 20-26


Dubois, 1994 - Dubois, E. - ALBERT at the Age of Two, Position Papers of the Dagstuhl Seminar on System Requirements Analysis: Management and Exploitation, Dagstuhl, Germany


Duvall, 1997 - Duvall, S. - Tool Supported Traceability for Development & Certification, Dpt. of Comp. Sci., University of Newcastle upon Tyne, MSc. Dissertation
References


EUROCAE, 1996a  European Organization for Civil Aviation Electronics - Certification Considerations For Highly-Integrated or Complex Aircraft Systems, EUROCAE document ARP 4754, Nov. (Issued in the United States by the Society of Automotive Engineers)


Evans, 1989  Evans, M. W. - The Software Factory, John Wiley & Sons


References

Fiksel & Hayes-Roth, 1993

FIPSP, 1993
Federal Information Processing Standards Publication - Announcing the Standard for Integration Definition for Information Modeling (IDEF1X); obtained from http://www.idef.com/

Fischer & Walker, 1979

Fischer, 1991

Flynn & Dorfman, 1990

Fragola & Spahn, 1973

Fraser et al., 1991

Freedman & Weinberg, 1981

Fyson & Boldyreff, 1998

Galle, 1996

Garcia, 1994

Gardner, 1994

Garg & Scacchi, 1989

Garg & Scacchi, 1990

Gieszl, 1992
Gieszl, Louis R. - Traceability For Integration, Proc. 2nd Int’l Conf. on Systems Integration, Morristown, New Jersey, USA, Jun., pp. 220-228

Gladden, 1982
Gladden, G. R. - Stop The Life-Cycle I Want To Get Off, Software Engineering Notes, Apr., pp. 35-39

Gogolla & Hohenstein, 1991

Gogolla, 1994
References

Goguen, 1996a  Goguen, J. A. - Formality and Informality in Requirements Engineering, Proc. 2nd Int'l Conf. on Requirements Engineering, Orlando, Florida, USA, Apr., pp. 2-10

Goguen, 1996b  Goguen, J. A. - Paramaterized Programming & Software Architecture, Proc. 4th Int'l Conf. on Software Reuse, Apr., pp. 2-11


Han, 1995  Han, Jun - A Document Based Approach to Software Engineering Environments, Proc. 5th Int'l CASE Symposium, Changsha, Hunan, China, Oct./Nov., pp. 128-133

Han, 1996  Han, Jun - Supporting Impact Analysis and Change Propagation In Software Engineering Environments, University of Monash, Victoria, Australia, TR-96-09, Oct.

Han, 1997a  Han, Jun - Designing For Increased Software Maintainability, Proc. Int'l Conf. on Software Maintenance, Bari, Italy, Sep./Oct., pp. 278-286

Han, 1997b  Han, Jun - Traceability and Consistency Support in Computer-Aided Design Environments, Proc. of 5th Int'l Conf. on Computer Aided Systems Theory & Technology, Las Palmas de Gran Canaria, Spain, Feb., pp. 11-15


R-8
References

Harris & Candy, 1999

Hatley & Pirbhai, 1987

Haveman & Pearson, 1997

Haveman et al., 1997

Hermens, 1991
Hermens, L. - A Software Specification Tool For Increased Requirements Traceability, University of Idaho, USA, MSc. Dissertation

Herzog & Scerri, 1998
Herzog, E. & Scerri, P. - The SEDRES Draft Standard /2 Data Model, SEDRES Deliverable W.4.6.1, May

Herzog & Törne, 1999

Hill, 1996
Hill, Mike - Parasitic Languages For Requirements, Proc. 2nd Int'l Conf. on Requirements Engineering, Orlando, Florida, USA, Apr., pp. 69-75

Hodge, 1994

Hoffman, 1990
Hoffman, Daniel - On Criteria For Module Interfaces, IEEE Trans. on Software Engineering,, May, pp. 537-542

Hoffnagle & Beregi, 1985

Horowitz & Williamson, 1986

Hsia et al., 1994

Hugge & Lang, 1995

Hughes & Martin, 1998

Hughes et al., 1995

Hull et al. (1991)

Humphrey, 1988
References

IEC, 1985

IEC, 1998

IEEE, 1977
Institute of Electrical & Electronics Engineers - Special Issue on Requirements Analysis and Requirements Tools, IEEE Trans. on Software Engineering, Jan., SE-3, No. 1

IEEE, 1987
Institute of Electrical & Electronics Engineers - Guide to Software Configuration Management, IEEE STD 1042-1987

IEEE, 1992

IEEE, 1993a
Institute of Electrical & Electronics Engineers - Standard For Software Maintenance, Jun.

IEEE, 1993b

Ihme et al., 1995
Ihme, T., Niemelä, E., Salmela, M. & Seppanen, V. - Object-Oriented Re-engineering of Embedded Software, Mechatronics, 5(1), pp. 73-86

Ince et al., 1993

Ince, 1994
Ince, Darrel - ISO 9001 and Software Quality Assurance, McGraw-Hill

INCOSE, 1996
International Council on Systems Engineering - Systems Engineering Capability Assessment Model (version 1.50), Jun

INCOSE, 1999
International Council on Systems Engineering - Results of Requirements Tools Survey (Revised); obtained from http://www.incose.org/workgrps/tools/rqsrvifo.html

Integrated, 1997
Integrated Chipware - Requirements Traceability Management, User Documentation

ITU-T, 1993
International Telecommunications Union - Recommendation Z.120: Message Sequence Chart (MSC)

Jackson & Renton, 1993

Jackson, 1986
Jackson, K. - Mascot 3 and Ada, Software Engineering Journal, May, pp. 121-135

Jackson, 1991
Jackson, Justin - A Keyphrase Based Traceability Scheme, Tools and Techniques For Maintaining Traceability During Design, IEE Colloquium, UK Digest Number: 1991/180, Dec., pp. 2/1-2/4

Jacobs & Kethers, 1994

Jacobson et al., 1993
Jacobson, I., Christerson, M., Jonsson, P. & Overgaard, G. - Object-Oriented Software Engineering - A Use Case Driven Approach, Addison-Wesley

Jarke et al., 1995
Jarke, M., Gellersdörfer, R., Jeusfeld, M., Staudt, M. & Eherer, S. - ConceptBase - A Deductive Object Base for Meta Data, Journal of
<table>
<thead>
<tr>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Jayaratna, 1994</strong></td>
</tr>
<tr>
<td><strong>Johnson, 1997</strong></td>
</tr>
<tr>
<td><strong>Kaindl, 1993</strong></td>
</tr>
<tr>
<td><strong>Kalinsky et al., 1989</strong></td>
</tr>
<tr>
<td><strong>Kelley, 1990</strong></td>
</tr>
</tbody>
</table>
References

Kenny, 1996
Kenny, Christy - Requirements Traceability; obtained from http://www.cs.usask.ca/grads/cab130/856/trace.ps

Keys, 1991

King et al., 2000

Klein, 1993a
Klein, Mark - Capturing Design Rationale In Concurrent Engineering Teams, IEEE Computer, Jan., pp. 39-47

Klein, 1993b

Klein, 1997a

Klein, 1997b

Kotonya & Sommerville, 1998
Kotonya, G. & Sommerville, I. - Requirements Engineering - Processes and Techniques, John Wiley & Sons Ltd.

Kunz & Rittel, 1970
Kunz, W. & Rittel, H. - Issues as Elements of Information Systems, Working Paper No. 131, University of Berkeley, California, USA, Center for Planning and Development Research

Kydd et al., 1994

Kyng, 1995

Laitinen, 1992

Landes & Studer, 1995

Lano, 1979

Lanobile & Visaggio, 1995

Laprie, 1989

Laubengayer & Spearman, 1994

Lee & Yen, 1993

Lee, 1991
Lee, J. - Extending the Potts and Bruns Model for Recording Design Rationale, Proc. 13th Int'l Conf. on Software Engineering, TX, USA, May.
References

Lefering, 1993
Lefering, Martin - An Incremental Integration Tool Between Requirements Engineering and Programming in the Large, Proc. 1st Int'l Symposium on Requirements Engineering, San Diego, California, USA, Jan., pp. 82-89

Lehman & Belady, 1985

Leite et al., 1997

Leveson & Harvey, 1983

Leveson & Reese, 1998

Leveson, 1995
Leveson, N. G. - Safeware: System Safety and Computers, Addison-Wesley

Lindsay & Traynor, 1998

Lindvall, 1997
Lindvall, M. - An Empirical Study of Requirements-Driven Impact Analysis in Object-Oriented Software Evolution, Linköping University, Sweden, Ph.D. Thesis

Liu et al., 1993

Lubars et al., 1993
Lubars, M., Potts, C. & Richter, C. A - Review of the State of the Art In Requirements Modelling, Proc. 1st Int'l Symposium on Requirements Engineering, California, USA, Jan., pp. 2-14

Lucey, 1992

Lueders, 1984

Luqi, 1990
Luqi - A Graph Model For Software Evolution, IEEE Trans. on Software Engineering, Aug., pp. 917-927

Macaulay, 1992
Macaulay, Linda - Requirements Capture As A Cooperative Activity, Proc. 1st Int'l Symposium on Requirements Engineering, San Diego, California, USA, pp. 174-181

MacLean et al., 1991
MacLean, A., Young, R., Bellotti, V. & Moran, T. - Questions, Options and Criteria: Elements of Design Space Analysis Human Computer Interaction, 6(3-4), pp. 201-250

Madhavji, 1992

Maier, 1993

Mair & Birdsall, 1992

Martin et al., 1993

R-13
<table>
<thead>
<tr>
<th>Reference</th>
<th>Author(s)</th>
<th>Title</th>
<th>Venue</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mason, 1996</td>
<td>Mason, P.</td>
<td>A Database Tool To Support Traceability For Dependable Avionic Systems</td>
<td>Department of Computing Science, University of Newcastle upon Tyne, MSc. Dissertation</td>
<td></td>
</tr>
<tr>
<td>Minker, 1988</td>
<td>Minker, J.</td>
<td>Perspectives in Deductive Databases</td>
<td>Logic Programming, No. 5, pp. 33-60</td>
<td></td>
</tr>
<tr>
<td>Morris et al., 1995</td>
<td>Morris, P., Masera, M., &amp; Wilikens, M.</td>
<td>Industrial Workshop on Requirements Engineering</td>
<td>Ispra, Italy, Oct.</td>
<td></td>
</tr>
</tbody>
</table>
| Mosley, 1992 | Mosley, Vicky | How To Assess Tools Efficiently and Quantitatively | IEEE
References

Muller, 1997
Muller, Pierre-Alain - Instant UML, Wrox Press Ltd.

Muller, 1979
Muller, G. - A Method for Controlled Requirements Specifications, Proc. International Conference on Software Engineering, Munich, Germany, pp. 126-135

Mylopoulos et al., 1990

Mylopoulos et al., 1992

NASA, 1993
National Aeronautics and Space Administration - Software Formal Inspection Standard (NASA-Std-2202),

Neely & Hartley, 1993

Nejmeh et al., 1989

Ni et al., 1994

Nixon et al., 1987

Oliver, 1994

Palmer & Evans, 1994

Palmer, 1997

Papaioannou & Theodoulidis, 1996

Parent et al., 1989

Patel et al., 1993

Paulk et al., 1993

Paulk, 1995
Paulk, M. - How ISO 9001 Compares with the CMM, IEEE Software, Jan., pp. 74-83
References

Paynter, 1995  

Paynter, 2000  

Pearson & Rowlands, 1996  

Pearson & Saeed, 1995  

Pearson & Saeed, 1996  

Pearson et al., 1998  

Pearson, 1996  

Peckham & Maryanski, 1988  

Pena-Mora, 1995  

Pierce, 1978  
Pierce, Robert A. - A Requirements Tracing Tool, ACM Software Engineering Notes, Nov., pp. 53-60

Pirnia & Hayek, 1981  

Plant & Tsoumpas, 1995  

Pohl & Haumer, 1995  

Pohl & Jacobs, 1994  

Pohl, 1996  
Pohl, K. - Process Centered Requirements Engineering, Research Studies Press

Polack, 1990  
Polack, A. - Practical Applications of CASE Tools on DoD Projects, ACM SIGSOFT Software Engineering Notes, Vol. 15, No. 1

Popov et al., 2001  

Potts & Bruns, 1988  
Potts, C. & Bruns, G. - Recording the Reasons for Design Decisions, Proc. 10th Int'l Conf. on Software Engineering, pp. 418-427
References

Potts et al., 1994

Potts, 1994
Potts, Colin - Inquiry Based Requirements Analysis, IEEE Software, Mar., pp. 21-32

Premerlani, 1994

Prowell & Poore, 1998

Pyle et al., 1993

QSS, 1998
Quality Systems & Software - DOORS Reference Manual, v4.0

Queille et al., 1994

Quillian, 1968

Ramamoorthy et al., 1986

Ramamoorthy et al., 1990
Ramamoorthy, C.V., Usuda, Y., Prakash, A. & Tsai, W. T. - The Evolution Support Environment System, IEEE Trans. on Software Engineering, 16(11), Nov., 1225-1234

Ramesh & Dhar, 1992

Ramesh & Edwards, 1993
Ramesh, B. & Edwards, M. - Issues In the Development of a Requirements Traceability Model, Proc. 1st Int'l Symposium on Requirements Engineering, San Diego, California, USA, Jan., pp. 256-259

Ramesh & Jarke, 1999

Ramesh et al., 1995

Ramesh, 1994

Ramsay & Bernsen, 1995

Rational Software Corporation, 1997a
Rational Software Corporation - The Unified Modelling Language, Version 1.3; obtained from http://www.rational.com

Rational Software Corporation, 1997b

Redden, 1999
Redden, L. E. - A Traceability Procedure to Support Project Management and the Development Process in Dependable Avionics Systems, Final Year Diss'n, Department of Comp. Sci., University of Newcastle upon Tyne

Regnell et al., 1996
<table>
<thead>
<tr>
<th>Reference</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sawyer et al., 1996</td>
<td>Sawyer, P., Sommerville, I. &amp; Viller, S. - PREview: Tackling the Real Concerns of Requirements Engineering, Lancaster University, Cooperative</td>
</tr>
</tbody>
</table>
References

Scalzo & Hugue, 1996

Schenk & Wilson, 1994
Schenk, D. & Wilson, P. - Information Modelling: The EXPRESS Way, Oxford University Press

Sciortino & Dunning, 1984

SEDRES, 1999
Systems Engineering Data Representation & Exchange - Capability 2 Data Model, SEDRES Deliverable, available from authors

SEI, 1995
Software Engineering Institute - Systems Engineering Capability Maturity Model (version 1.1), Carnegie Mellon University

Selic et al., 1994
Selic, B., Gullekson, G. & Ward, P. T. - Real-Time Object-Oriented Modelling, Wiley & Sons

Sheard, 1997
Sheard, A. - The Frameworks Quagmire, A Brief Look, Software Productivity Consortium; obtained from http://www.software.org

Shilling & Sweeney, 1989

Shipman, 1981

Silva & Agusta, 1998

Simpson, 1986

Simpson, 1994

Simpson, 2000a

Simpson, 2000b

Simpson, 2000c

Singh & Han, 1996
Singh, H. & Han, J. - Modelling Software Artifacts & Their Relationship in Software Engineering Environments, TR96-06, Peninsula School of Computing and Information Technology, Monash University, Melbourne, Australia, May

Smith & Smith, 1977

Smith et al., 1997

Smith, 1993
Smith, T. J. - READS: A Requirements Engineering Tool, Proc. 1st Int'l Symposium on Requirements Engineering, San Diego, California, USA, Jan., pp. 94-97

R-19
References

Sodhi, 1991  
Sodhi, Jag - Software Engineering: Methods, Management and CASE Tools, McGraw-Hill

Sommerville & Sawyer, 1997  

Sommerville et al., 1993  
Sommerville, I., Rodden, T., Sawyer, P., Bentley, R. & Twidle, M. - Integrating Ethnography into the Requirements Engineering Process, Proc. 1st Int'l Symposium on Requirements Engineering, California, USA, Jan., pp. 165-173

Spivey, 1989  
Spivey, J. M. - The Z Notation: A Reference Manual, Prentice-Hall

Srivastava et al. 1993  
Srivastava, D., Ramakrishnan, R., Seshadri, P., & Sudarshan, S. - Coral++: Adding Object-Orientation to a Logic Database Language, in Proc. Int. Conf. on Very Large Databases, Dublin, Ireland

Stallman & Sussman, 1977  

Stephenson, 1997  

Storey, 1996  
Storey, Neil - Safety-Critical Computer Systems, Addison-Wesley

Strens, 1995  
Strens, R. - Risk, Sensitivity and Impact Analysis and Their Application to Changing Requirements, Proteus Project Document

Su, 1986  

Sugden & Strens, 1996  

Sukamaran, 1999  
Sukamaran, S. - A Graphical User Interface For a Traceability Tool, Department of Computing Science, University of Newcastle upon Tyne, MSc. Dissertation

Sutcliffe et al., 1998  

Takahashi & Yamamoto, 1995  

Takahashi et al., 1996  
Takahashi, K., Potts, C. Kumar, V., Ota, K. & Smith, J. D. - Hypermedia Support for Collaboration In Requirements Analysis, Proc. 2nd Int'l Conf. on Requirements Engineering, Orlando, Florida, USA, Apr., pp. 31-40

Takeda et al., 1993  
Takeda, N., Shiomi, A., Kawai, K. & Ohiwa, H. - Requirements Analysis by the KJ Editor, Proc. 1st Int'l Symposium on Requirements Engineering, San Diego, California, USA, Jan., pp. 98-101

Tamanaha et al., 1989  

Teichroew & Sayani, 1971  
Teichroew, D. & Sayani, H. - Automation of System Building, Datamation, Aug., pp. 25-30

Telelogic, 2001  
Telelogic - DOORS Reference Manual, v5.1

Theriault, 1991  
Theriault, M. - The Requirements Traceability Tool on VSCS, Proc. 36th
References

Tiel, 1993

Tilbury, 1989

Toulmin, 1958
Toulmin, S. E. - The Uses of Argument, Cambridge University Press

Tran et al., 1997

Tryggeseth & Nytro, 1997

Turton et al., 1997

van Lamsweerde et al., 1995

Vesely et al., 1981

Villemeur, 1992

Warmer & Kleppe, 1999
Warmer, J. & Kleppe, A. - The Object Constraint Language: Precise Modelling with UML, Addison-Wesley

Watkins & Neal, 1994

Weidenhaupt et al., 1998

Weir, 2001

West, 1991
West, Martin - QFD In Software Development, IBM Systems Journal, Nov., pp. 5/1-5/7

Westfechtel, 1989
Westfechtel, B. - Revision Control In An Integrated Development Environment, Software Engineering Notes, 14(7), pp. 96-105

White, 1993

White, 1994a

White, 1994b
White, Stephanie - Traceability for Complex Systems Engineering, Proc. 4th Int’l Symposium on Systems Engineering, Sunnyvale, California, USA, Aug., pp. 49-55

White, 1997
White, Stephanie - Requirements Capture and Analysis Prior to Modelling, Proc. of Conference and Workshop on Engineering of Computer-Based Systems (ECBS ’97), Monterey; CA, Mar., pp. 10-17
<table>
<thead>
<tr>
<th>Reference</th>
<th>Author(s)</th>
<th>Title and Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whitgift, 1991</td>
<td>Whitgift, David</td>
<td>Software Configuration Management: Methods &amp; Tools, John Wiley and Sons</td>
</tr>
<tr>
<td>Wieringa, 1996</td>
<td>Wieringa, R. J.</td>
<td>Requirements Engineering: Frameworks for Understanding, Wiley</td>
</tr>
<tr>
<td>Wilson et al., 1997b</td>
<td>Wilson, W. M., Rosenberg, L. H. &amp; Hyatt, L. E.</td>
<td>Automated Analysis of Requirement Specifications, Proc. 20th Int'l Conf. on Software Engineering, Boston, USA, May</td>
</tr>
<tr>
<td>Yau &amp; Tsai, 1987</td>
<td>Yau, S. S. &amp; Tsai, J.</td>
<td>Knowledge Representation of Software Component Interconnection Information for Large Scale Software Modifications, IEEE Trans. on Software Engineering, 13(3), pp. 335-361</td>
</tr>
<tr>
<td>Zaniolo, 1983</td>
<td>Zaniolo, Carlo</td>
<td>The Database Language GEM, SIGMOD Conference, pp. 201-218</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Definition</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>2RARE</td>
<td>2 Real Applications for Requirements Engineering</td>
<td></td>
</tr>
<tr>
<td>ABE</td>
<td>Aerospace Build Entity</td>
<td></td>
</tr>
<tr>
<td>ADL</td>
<td>Activity Description Language</td>
<td></td>
</tr>
<tr>
<td>ADT</td>
<td>Abstract Data Type</td>
<td></td>
</tr>
<tr>
<td>AEA</td>
<td>Aerospace Engineering Association</td>
<td></td>
</tr>
<tr>
<td>AEE</td>
<td>Aerospace Engineering Entity</td>
<td></td>
</tr>
<tr>
<td>AEO</td>
<td>Aerospace Engineering Object</td>
<td></td>
</tr>
<tr>
<td>AEP</td>
<td>Aerospace Engineering Project</td>
<td></td>
</tr>
<tr>
<td>ALE</td>
<td>Aerospace Link Entity</td>
<td></td>
</tr>
<tr>
<td>AME</td>
<td>Aerospace Management Entity</td>
<td></td>
</tr>
<tr>
<td>ANSI</td>
<td>American National Standards Institute</td>
<td></td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
<td></td>
</tr>
<tr>
<td>ARP</td>
<td>Aerospace Recommended Practice</td>
<td></td>
</tr>
<tr>
<td>ASM</td>
<td>Activity State-Machine</td>
<td></td>
</tr>
<tr>
<td>ATE</td>
<td>Aerospace Traceability Entity</td>
<td></td>
</tr>
<tr>
<td>BCET</td>
<td>Best-Case Execution Time</td>
<td></td>
</tr>
<tr>
<td>BCS</td>
<td>BASE Control Software/British Computer Society</td>
<td></td>
</tr>
<tr>
<td>BE</td>
<td>Build Element</td>
<td></td>
</tr>
<tr>
<td>BNF</td>
<td>Backus-Naur Form</td>
<td></td>
</tr>
<tr>
<td>BSCU</td>
<td>Brake System Control Unit</td>
<td></td>
</tr>
<tr>
<td>CAD</td>
<td>Computer-Aided Design</td>
<td></td>
</tr>
<tr>
<td>CAoA</td>
<td>Corrected Angle of Attack</td>
<td></td>
</tr>
<tr>
<td>CARE</td>
<td>Common Airbus Requirements Engineering</td>
<td></td>
</tr>
<tr>
<td>CAS</td>
<td>Computed Air Speed</td>
<td></td>
</tr>
<tr>
<td>CASE</td>
<td>Computer-Aided Systems/Software Engineering</td>
<td></td>
</tr>
<tr>
<td>CBQL</td>
<td>ConceptBase Query Language</td>
<td></td>
</tr>
<tr>
<td>CCA</td>
<td>Common Cause Analysis</td>
<td></td>
</tr>
<tr>
<td>CMA</td>
<td>Common Mode Analysis</td>
<td></td>
</tr>
<tr>
<td>CMM</td>
<td>Capability Maturity Model</td>
<td></td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial Of The Shelf</td>
<td></td>
</tr>
<tr>
<td>CREWS</td>
<td>Co-operative Requirements Engineering With Scenarios</td>
<td></td>
</tr>
<tr>
<td>CSU</td>
<td>Command Sensor Unit</td>
<td></td>
</tr>
<tr>
<td>DCSC</td>
<td>Dependable Computing Systems Centre</td>
<td></td>
</tr>
<tr>
<td>DDS</td>
<td>Deductive Database System</td>
<td></td>
</tr>
<tr>
<td>DMS</td>
<td>Database Management System</td>
<td></td>
</tr>
<tr>
<td>DOODS</td>
<td>Deductive Object-Oriented Database System</td>
<td></td>
</tr>
<tr>
<td>DOORS</td>
<td>Dynamic Object-Oriented Requirements System</td>
<td></td>
</tr>
</tbody>
</table>
Glossary

DORIS  Data-Oriented Requirements Implementation Scheme
DRCS  Design Rationale Capture System
DRL  Decision Representation Language
DTC  Data Transfer Cartridge
DTM  Data Transfer Module
DTMIS  Data Transfer Module Interface Software
DXL  DOORS eXtension Language
ELAC  ELevator and Aileron Computer
ELH  Entity Life History
ESA  European Space Agency
EST  Earliest Start Time
EUROCAE  EUROpean organisation for Civil Aviation Electronics
FAA  Federal Aviation Authority
FBD  Function Block Diagram
FCS  Flight Control System
FHA  Functional Hazard Assessment
FMEA  Failure Modes and Effects Analysis
FPPU  Feedback Position Pickoff Unit
FTA  Fault Tree Analysis
GEM  General Entity Manipulator
gIBIS  graphical Issue-Based Information System
GSN  Goal Structure Notation
HAZOP  HAZard and OPerability studies
HTML  Hypertext Markup Language
IBIS  Issue-Based Information System
IDA  Intercommunication Data Area
IDEF  ICAM (Integrated Computer Aided Manufacturing) DEFinition method
IEEE  Institute of Electrical & Electronics Engineers
INCOSE  INternational Council on Systems Engineering
IP  Information Provision
IR  Information Request
ISO  International Standards Organisation
ISR  Inhibit Slat Retraction
ISRE  Immersive Scenario-based Requirements Engineering
JAR  Joint Airworthiness Requirements
KARE  Knowledge Acquisition in Requirements Engineering
LDL  Logical Data Language
LES D  Linguistic Engineering for Software Design
LST  Latest Start Time
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>MASCOT</td>
<td>Modular Approach to Software, Construction, Operation and Test</td>
</tr>
<tr>
<td>MATrA</td>
<td>Meta-modelling Approach to Traceability for Avionics</td>
</tr>
<tr>
<td>MCM</td>
<td>MATrA Configuration Model</td>
</tr>
<tr>
<td>MMSS</td>
<td>Modular Mission Support Software</td>
</tr>
<tr>
<td>MNLS</td>
<td>MATrA Natural Language Structure</td>
</tr>
<tr>
<td>MPS</td>
<td>Mission Planning System</td>
</tr>
<tr>
<td>MSC</td>
<td>Message Sequence Chart</td>
</tr>
<tr>
<td>NATURE</td>
<td>Novel Approaches to Theories Underlying Requirements Engineering</td>
</tr>
<tr>
<td>OCL</td>
<td>Object Constraint Language</td>
</tr>
<tr>
<td>OMT</td>
<td>Object Modelling Technique</td>
</tr>
<tr>
<td>OODS</td>
<td>Object-Oriented Database System</td>
</tr>
<tr>
<td>ORM</td>
<td>Object-Role Modelling</td>
</tr>
<tr>
<td>pCDS</td>
<td>populated CASE tool Data Structure</td>
</tr>
<tr>
<td>PDS</td>
<td>Product Data Synthesis</td>
</tr>
<tr>
<td>PERT</td>
<td>Programme Evaluation and Review Technique</td>
</tr>
<tr>
<td>PIM</td>
<td>Process Improvement Model</td>
</tr>
<tr>
<td>pNDS</td>
<td>populated Notation Dependent Structure</td>
</tr>
<tr>
<td>PRA</td>
<td>Particular Risk Analysis</td>
</tr>
<tr>
<td>PSSA</td>
<td>Preliminary System Safety Assessment</td>
</tr>
<tr>
<td>QFD</td>
<td>Quality Function Deployment</td>
</tr>
<tr>
<td>QOC</td>
<td>Questions, Options and Criteria</td>
</tr>
<tr>
<td>RDD</td>
<td>Requirements-Driven Design</td>
</tr>
<tr>
<td>RDMS</td>
<td>Relational Database Management System</td>
</tr>
<tr>
<td>RE</td>
<td>Requirements Engineering</td>
</tr>
<tr>
<td>REAIMS</td>
<td>Requirements Engineering Adaptation for IMprovement for Safety</td>
</tr>
<tr>
<td>RDL</td>
<td>Reference and IDEa Language</td>
</tr>
<tr>
<td>RTM</td>
<td>Requirements &amp; Traceability Management</td>
</tr>
<tr>
<td>RTN</td>
<td>Real-Time Network</td>
</tr>
<tr>
<td>RTN-SL</td>
<td>Real-Time Network Specification Language</td>
</tr>
<tr>
<td>RTN-SLg</td>
<td>Real-Time Network Specification Language graphical syntax</td>
</tr>
<tr>
<td>RTT</td>
<td>Real-Time Transaction</td>
</tr>
<tr>
<td>SAM</td>
<td>Safety Argument Manager</td>
</tr>
<tr>
<td>SAM*</td>
<td>Semantic Association Model</td>
</tr>
<tr>
<td>SDM</td>
<td>Semantic Database Model</td>
</tr>
<tr>
<td>SEC</td>
<td>Spoiler and Elevator Computer</td>
</tr>
<tr>
<td>SECAM</td>
<td>Systems Engineering Capability Assessment Model</td>
</tr>
<tr>
<td>SEDRES</td>
<td>System Engineering Data Representation &amp; Exchange Standardisation</td>
</tr>
<tr>
<td>SEI</td>
<td>Software Engineering Institute</td>
</tr>
<tr>
<td>SENLS</td>
<td>Scenario Event Natural Language Structure</td>
</tr>
</tbody>
</table>
Glossary

SENM Systems Engineering Notation Meta-class model
SFCC Slat and Flap Control Computer
SHM+ Semantic Hierarchy Model
SLATE System Level Automation Tool for Engineers
SML Safety Markup Language
SP Service Provision
SQL Structured Query Language
SR Service Request
SRS System/Software Requirements Specification
SSA System Safety Assessment
STEFFIE Systems Engineering Framework For Information and experience Exchange
STEP Standard for The Exchange of Product data
TCS Theatre Creation Software
UCRS User Centred Requirements Structure
UML Unified Modelling Language
uNDS unpopulated Notation Dependent Structure
VDM Vienna Development Method
WBS Wheel Braking System
WCET Worst-Case Execution Time
WCRT Worst-Case Response Time
ZSA Zonal Safety Analysis
This page deliberately left blank
Appendix A (Part 1) - Additional Rules & Constraints for Use Case View & Models

The following rules and constraints provide an addendum to 4.3.3.3.2.

i. Constraint to prevent ‘dangling’ Actor instances (i.e., not attached to an interaction).

**Actor invariant**

\[
\text{self.allInstances->forall(a I self.useCaseView->forall(v I self.useCaseView.ucv_interaction->exists(i I v.ucv_interaction->includes(i) and (i.interactor_1= a or i.interactor_2= a)))})
\]

ii. Constraint to prevent a UseCase from extending itself.

**Extends invariant**

\[
\text{self.allInstances->forall(e I e.extends_base <> e.extends_extend)}
\]

iii. Constraint to ensure that at most one «extends» association exists between two Use Cases.

**Extends invariant**

\[
\text{self.allInstances->forall(e1, e2 I not (e1 <> e2 and e1.extends_base = e2.extends_base and e1.extends_extend = e2.extends_extend))}
\]

iv. Constraint to ensure that two Use Cases cannot extend each other; i.e. not ‘use case a’ «extends» ‘b’ and ‘use case b’ «extends» ‘a’.

**Extends invariant**

\[
\text{self.allInstances->forall(e1, e2 I not (e1.extends_extend = e2.extends_base and e1.extends_base = e2.extends_extend))}
\]

v. Constraint to prevent cycles within «extends» associations.

First, we define a rule to determine the transitive closure of «extends» associations.

**UseCase**

\[
\text{self.allInstances->forall(u2, u1 I self.extends->exists(e1 I e1.extends_extend = u2 and e1.extends_base = u1) or self.allInstances->exists(u3 I self.extends->exists(e2 I e2.extends_extend = u2 and e2.extends_base = u3 and u3.transitive_extended_by->includes(u1)) implies u2.transitive_extended_by->includes(u1))}
\]

This rule allows us to specify the following constraint.

**UseCase invariant**

\[
\text{self.allInstances->forall(u I not (u.transitive_extended_by->includes(u))})
\]

vi. Rules to ensure that all elements associated with a particular model are also associated with the UseCaseView to which that model belongs.

a) Rule for deriving Actors.

**UseCaseView**

\[
\text{self.allInstances->forall (v I}
\]

A-1
b) Rule for deriving Interactions.

UseCaseView
self.allinstances->forall (v |
self.use_case_model.ucm_interaction->forall (i |
v.use_case_model.ucm_interaction->includes(i)))
implies
v.ucv_interaction->includes(i)

c) Rule for deriving «includes» associations.

UseCaseView
self.allinstances->forall (v |
self.use_case_model.ucm_includes->forall (i |
v.use_case_model.ucm_includes->includes(i)))
implies
v.ucv_includes->includes(i)

d) Rule for deriving «extends» associations.

UseCaseView
self.allinstances->forall (v |
self.use_case_model.ucm_extends->forall (e |
v.use_case_model.ucm_extends->includes(e)))
implies
v.ucv_extends->includes(e)

e) Rule for deriving Pre-conditions.

UseCaseView
self.allinstances->forall (v |
self.use_case_model.ucm_pre_condition->forall (p |
v.use_case_model.ucm_pre_condition->includes(p)))
implies
v.ucv_pre_condition->includes(p)

f) Rule for deriving Post-conditions.

UseCaseView
self.allinstances->forall (v |
self.use_case_model.ucm_post_condition->forall (p |
v.use_case_model.ucm_post_condition->includes(p)))
implies
v.ucv_post_condition->includes(p)

vii. Rules to derive «includes» use cases (a) and associations (b) where a UseCase appears in more than one model.

a) UseCaseModel
    self.allinstances->forall(m |
    self.useCaseView.ucv_includes->forall (i |
ix. The following rules are what we term ‘spine’ associations (e.g., ucm_interaction, ucm_includes, ucm_extends, etc.) that anchor elements to the UseCaseModel class.

a) Rule to populate ucm_interaction.

UseCaseModel
self.allInstances->forall (m l
self.ucm_use_case->forall(u l
self.ucm_actor->forall(a l
self.ucm_use_case.interaction->forall(i l
m.ucm_use_case->includes(u) and
m.ucm_actor->includes(a) and
(i.interactor_1->union(i.interactor_2))->includes(u) and
(i.interactor_1->union(i.interactor_2))->includes(a))))
implies
m.ucm_interaction->includes(i)
b) Rule to populate ucm_includes.

UseCaseModel
self.allInstances->forall (m)
self.ucm_use_case->forall(u1, u2)
self.ucm_use_case.includes->forall(i)
m.ucm_use_case->includes(u1) and
m.ucm_use_case->includes(u2) and
(i.includes_base->union(i.includes_include))->includes(u1) and
(i.includes_base->union(i.includes_include))->includes(u2))
implies
m.ucm_includes->includes(i)

c) Rule to populate ucm_extends.

UseCaseModel
self.allInstances->forall (m)
self.ucm_use_case->forall(u1, u2)
self.ucm_use_case.extends->forall(e)
m.ucm_use_case->includes(u1) and
m.ucm_use_case->includes(u2) and
(e.extends_base->union(e.extends_extend))->includes(u1) and
(e.extends_base->union(e.extends_extend))->includes(u2))
implies
m.ucm_extends->includes(e)
Appendix A (Part 2) - Additional Rules & Constraints for Interaction View & Models

The following rules and constraints provide an addendum to 4.3.3.4.2.

i. Potential ‘style rule’ (expressed over textual perspective) ensuring that Communication Event types are compatible with PDS InputOutput flow types; i.e., service request and service provision flows are not recorded as non-triggering in the PDS. Similarly information requests and information provision are not recorded as triggering. Note that PDS flow types (particularly at the requirements stage may be as yet “undetermined”). Note also that this rule requires addition of an enumerated effect attribute (enumerators triggering and non_triggering) to the InputOutput class in the PDS.

\[
\text{CommunicationEvent} \\
\text{self.allInstances} \rightarrow \forall (c \mid \text{not } \exists (\text{be}) \left( \text{c.tsn_communication_event.communication_description.tsn_message_node.bEelementAEO.build_element->not exists(be) c.tsn_communication_event.communication_description.tsn_message_node.message_name = be.flow_name and (be.effect="triggering" and (c.interaction_type = "IR" or c.interaction_type = "IP") or (be.effect = "non_triggering" and (c.interaction_type = "SR" or c.interaction_type = "SP") )))
\]

ii. Constraint to ensure that, for Message Sequence Chart perspectives, where a link_name is provided rather than using the default “unspecified” (a link is some form of connection between two instances), then there is a connection (Interface) of the same name specified in the PDS.

\[
\text{MscCommunication} \text{ invariant} \\
\text{self.allInstances} \rightarrow \forall (m \mid \text{not } (m.msc_message.bEelementAEO.build_element.sentTo.target.connection.interface->intersection (m.msc_message.bEelementAEO.build_element.receivedFrom.target.connection.interface) ->isEmpty and m.link_name = "unspecified") or (m.msc_message.bEelementAEO.build_element.sentTo.target.connection.interface->intersection (m.msc_message.bEelementAEO.build_element.receivedFrom.target.connection.interface) ->notEmpty and m.link_name = "unspecified") or (m.msc_message.bEelementAEO.build_element.sentTo.target.connection.interface->intersection (m.msc_message.bEelementAEO.build_element.receivedFrom.target.connection.interface) ->isEmpty and m.link_name = "unspecified") or (m.msc_message.bEelementAEO.build_element.sentTo.target.connection.interface.interface_name <> m.link_name and m.msc_message.bEelementAEO.build_element.receivedFrom.target.connection.interface.interface_name <> m.link_name)))
\]

iii. The following constraints ensure correct combination of TimingEvent constructs.

\[
\text{TimingEvent} \text{ invariant} \\
\text{self.allInstances} \rightarrow \forall (t \mid \text{not } (t.tsn_timing_event.timing_description.tsn_host_on_timeout_node->size = 1 and t.tsn_timing_event.timing_description.tsn_timer_duration->size = 1) or (t.msc_timing_event.msc_host_on_timeout_instance->size = 1 and t.msc_timing_event.msc_timer_duration->size = 1))
\]

TimingEvent invariant
self.allInstances->forall(t | (not (t.tsn_timing_event.timing_description.tsn_host_on_timeout_node->size = 1 and t.tsn_timing_event.timing_description.tsn_timer_duration->size = 1) or (t.msc_timing_event.msc_host_on_timeout_instance->size = 1 and t.msc_timing_event.msc_timer_duration->size = 1)))
Appendix A (Part 2)

(t. tsn_timing_event.timing_description.tsn_timer_set_node->size = 1 and
t. tsn_timing_event.timing_description.tsn_timer_duration->size = 0)
or
(t. msc_timing_event.msc_timer_set_instance->size = 1 and
t. msc_timing_event.msc_timer_duration->size = 0))

TimingEvent Invariant
self.allInstances->forall(t I
  (not
   (t. tsn_timing_event.timing_description.tsn_timer_reset_node->size = 1 and
    t. tsn_timing_event.timing_description.tsn_timer_duration->size = 1)
or
   (t. msc_timing_event.msc_timer_reset_instance->size = 1 and
    t. msc_timing_event.msc_timer_duration->size = 1))))

iv. This constraint (again expressed for the textual perspective) ensures that Instance elements in Timer
Events correspond to the target system - i.e., are the same as the subject_module (as stated by the
User Centred Requirements Structure).

TimingEvent Invariant
self.allInstances->forall(t I
  t. tsn_timing_event.timing_description.tsn_host_on_timeout_node.instance_name->union
  t. tsn_timing_event.timing_description.tsn_timer_set_node.instance_name->union
  t. tsn_timing_event.timing_description.tsn_timer_reset_node.instance_name->union
  t. interactionModel.interactionView.userCentredRequirementsStructure.subject_module->includes
  t. interactionModel.interactionView.userCentredRequirementsStructure.subject_module)

v. The following constraints apply to sequence numbers.

a) The set of events preceding an Event are recorded to ensure sequence numbers are correctly
maintained (e.g., following insert operations). In order to do this, we first define a rule for
population of transitive_follows_from.

Scenario
self.allInstances->forall(s I
  self.scenario_event->union(self.included_event->union(self.extension_event))->forall(e1, e2 I
    s.scenario_event->union(s.included_event->union(s.extension_event))->includes(e1) and
    s.scenario_event->union(s.included_event->union(s.extension_event))->includes(e2) and
    e1 <> e2 and
    e2.follows_from->includes(e1))
or
  self.scenario_event->union(self.included_event->union(self.extension_event))->exists(e' I
    s.scenario_event->union(s.included_event->union(s.extension_event))->includes(e') and
    e' <> e1 and e' <> e2 and
    e2.follows_from->includes(e') and
    e'.follows_from->includes(e1))
  )

implies
  e2.transitive_follows_from->includes(e1)

b) Constraint to ensure ‘normal sequence numbers’ are correct. Note that in order to prevent repetition,
an Event may in principle appear in all scenarios belonging to a use case and must therefore be
capable of having several sequence numbers (potentially up to the number of scenarios describing
that use case). These must be matched to the Scenario to which they apply.

Scenario Invariant
self.allInstances->forall(s I
  self.scenario_event->forall(e I
Appendix A (Part 2)

c) Constraint to ensure ‘included sequence numbers’ are correct. Again these must be matched to the Scenario to which they apply, given that an event can be included in n > 1 scenarios and therefore has n > 1 ‘included sequence numbers’.

Scenario Invariant
self.allInstances->forall(s)
self.included_event->forall(e)
self.included_event.included_seq_no->intersection(self.scn_included_seq_no)
->not exists(n)
(s.included_event->includes(e) and
e.included_seq_no->intersection(s.scn_included_seq_no)->includes(n) and
n.sequence_no <>
(s.scenario_event->union(s.included_event->union(s.extension_event))
->intersection (e.transitive_follows_from))->size+1)))))

d) And similarly with ‘extension sequence numbers’.

Scenario Invariant
self.allInstances->forall(s)
self.extension_event->forall(e)
self.extension_event.extension_seq_no->intersection(self.scn_extension_seq_no)
->not exists(n)
(s.extension_event->includes(e) and
e.extension_seq_no->intersection(s.scn_extension_seq_no)->includes(n) and
n.sequence_no <>
(s.scenario_event->union(s.included_event->union(s.extension_event))
->intersection (e.transitive_follows_from))->size+1)))))

e) The following constraint ensures that each Event has only one included or ‘extension sequence number’ per Scenario for which it is included in or extends.

Scenario Invariant
self.allInstances->forall(s)
self.included_event->forall(e)
self.extension_event->forall(x)
not(
(s.included_event->includes(e) and
s.scn_included_seq_no->intersection(e.included_seq_no)->size <>1) or
(s.extension_event->includes(x) and
s.scn_extension_seq_no->intersection(x.extension_seq_no)->size <>1)))))

vi. Constraint to ensure 'extends scenarios' are paths through different use cases.

InteractionModel Invariant
self.allInstances->forall(m1, m2)
self.inm_scenario->forall(s1, s2)
not (m1.inm_scenario->includes(s1) and m2.inm_scenario->includes(s2) and
s1.extended_by_scenario->includes(s2) and m1 = m2))
vii. The following rules populate associations for Scenario and Event elements of Interaction Models. They arise from the presence of «extends» associations in the Use Case Models they describe. Note similar rules may be defined for «includes».

a) To derive extension scenarios of a model (inm_extension_scenario), it is first necessary to define a rule for population of transitive_extended_by.

\[
\text{InteractionModel}
\]
\[
\text{self.allInstances->forall(m1, m2) I}
\text{self.inm_scenario->forall(s1, s2) I}
\text{m1.inm_scenario->includes(s1) and}
\text{m2.inm_scenario->includes(s2) and}
\text{s1.extended_by_scenario->includes(s2)}
\]
or
\[
\text{self.allInstances->exists(m' I}
\text{self.inm_scenario->exists(s' I}
\text{m'.inm_scenario->includes(s') and}
\text{(s1.extended_by_scenario->includes(s') or s1.includes_scenario->includes(s')) and}
\text{s'.transitive_extended_by_scenario->includes(s2)))}
\]
implies
\[
\text{s1.transitive_extended_by_scenario->includes(s2)}
\]

b) We can now define a rule for deriving «extends» scenarios.

\[
\text{InteractionModel}
\]
\[
\text{self.allInstances->forall(m1, m2) I}
\text{self.inm_scenario->forall(s1, s2) I}
\text{m1.inm_scenario->includes(s1) and}
\text{m2.inm_scenario->includes(s2) and}
\text{s1.extended_by_scenario->includes(s2) or}
\text{s1.transitive_extended_by_scenario->includes(s2))}
\]
implies
\[
\text{m1.inm_extension_scenario->includes(s2)}
\]

c) In addition, the following populate the extension_event association between Scenario and Event.

\[
\text{Scenario}
\]
\[
\text{self.allInstances->forall(s) I}
\text{self.transitive_extended_by_scenario->forall(e I}
\text{s.transitive_extended_by_scenario->includes(e)))}
\]
implies
\[
\text{s.extension_event->includesAll(e.scenario_event)}
\]

d) Also, the following populate the extension_event_group association between Scenario and EventGroup.

\[
\text{Scenario}
\]
\[
\text{self.allInstances->forall(s l}
\text{self.transitive_extended_by_scenario->forall(e l}
\text{s.transitive_extended_by_scenario->includes(e) and}
\text{e.event_group->size > 0))}
\]
implies
\[
\text{s.extension_event_group->includesAll(e.event_group)}
\]

viii. Constraint to ensure that for two Interaction Models (m1, m2) where the Use Case View contains an «extends» association in which the use case described in m1 is the base of this association and
the use case described in m2 is the extended part, then at least one Scenario forming m1 should appear in the extended scenarios of any Scenario in m2.

**InteractionModel Invariant**

```plaintext
self.allInstances->forall(m1, m2 | self.interactionView.userCentredRequirementsStructure.useCaseView.ucv_extends ->not exists(e | m1.describes_use_case = e.extends_base.use_case_name and m2.describes_use_case = e.extends_extend.use_case_name and (m2.inm_scenario.extended_by_scenario->intersection(m1.inm_scenario))->isEmpty))
```

ix. The following constraints enforce uniqueness of Message (a), Action (b) and Timer (c) primitives throughout an InteractionView.

a) **InteractionView invariant**

```plaintext
self.allInstances->forall(v | self.interaction_model.inm_message->forall(m1, m2 | not(v.interaction_model.inm_message->includes(m1) and v.interaction_model.inm_message->includes(m2) and m1 <> m2 and m1.message_name = m2.message_name)))
```

b) **InteractionView invariant**

```plaintext
self.allInstances->forall(v | self.interaction_model.inm_action->forall(a1, a2 | not(v.interaction_model.inm_action->includes(a1) and v.interaction_model.inm_action->includes(a2) and a1 <> a2 and a1.action_name = a2.action_name)))
```

c) **InteractionView invariant**

```plaintext
self.allInstances->forall(v | self.interaction_model.inm_timer->forall(t1, t2 | not(v.interaction_model.inm_timer->includes(t1) and v.interaction_model.inm_timer->includes(t2) and t1 <> t2 and t1.timer_name = t2.timer_name)))
```

x. Rules to derive textual event representations from Message Sequence Charts (note derives primitives only, manual intervention is required to produce meaningful event statements).

a) **Rule to derive Communication Events.**

```plaintext
CommunicationEvent self.allInstances->forall(c | self.tsn_communication_event->exists(t | c.msc_communication_event->notEmpty and c.tsn_communication_event->includes(t)) implies t.communication_description.tsn_sender_node->includes(c.msc_communication_event.msc_sender_instance) and t.communication_description.tsn_receiver_node->includes(c.msc_communication_event.msc_receiver_instance) and t.communication_description.tsn_message_node->includes(c.msc_communication_event.msc_message)
```
b) Rule to derive Internal Action Events.

\[
\text{InternalActionEvent}
\]
\[
\text{self.allInstances->forall(a I}
\text{self.tsn_action_event->exists(t I}
\text{a.msc_action_event->notEmpty}
\text{and}
\text{a.tsn_action_event->includes(t)))}
\text{implies}
\text{t.action_description.tsn_sdr.rcr_node->includes(a.msc_action_event.msc_sdr.rcr_instance) and}
\text{t.action_description.tsn_action_node->includes(a.msc_action_event.msc_system_action)}
\]

c) Rule to derive Timing Events.

\[
\text{TimingEvent}
\]
\[
\text{self.allInstances->forall(e I}
\text{self.tsn_timing_event->exists(t I}
\text{e.msc_timing_event->notEmpty and}
\text{e.tsn_timing_event->includes(t)))}
\text{implies}
\text{t.timing_description.tsn_host_on_timeout_node->union(t.timing_description.tsn_timer_set_node->union(t.timing_description.tsn_timer_reset_node))->includes(e.msc_timing_event.msc_host_on_timeout_instance->union(e.msc_timing_event.msc_timer_set_instance->union(e.msc_timing_event.msc_timer_reset_instance))) and}
\text{t.timing_description.tsn_timer_instance_node->includes(e.msc_timing_event.msc_timer_instance) and}
\text{t.timing_description.tsn_timer_duration->includes(e.msc_timing_event.msc_timer_duration)}
\]

xi. For more direct navigation, in particular between the Use Case and Interaction Views, a number of rules are used to populate InteractionModel 'spine' associations (e.g., inm_tsn_viewpoint, inm_tsn_action, etc.). A selection of these are defined as follows:-

a) Rule for population of Textual Viewpoints (a similar rule for MSC Viewpoints can also be defined).

\[
\text{InteractionModel}
\]
\[
\text{self.allInstances->forall(m I}
\text{self.inm_scenario.tsn_viewpoint->forall(t I}
\text{m.inm_scenario.tsn_viewpoint->includes(t))}
\text{implies}
\text{m.inm_tsn_viewpoint->includes(t)}
\]

b) Rule for population of Textual Viewpoints for each event type.

\[
\text{InteractionModel}
\]
\[
\text{self.allInstances->forall(m I}
\text{self.inm_tsn_viewpoint.tvp_tsn_comm->forall(c I}
\text{m.inm_tsn_viewpoint.tvp_tsn_comm->includes(c))}
\text{implies}
\text{m.inm_tsn_communication->includes(c)}
\]

\[
\text{InteractionModel}
\]
\[
\text{self.allInstances->forall(m I}
\text{self.inm_tsn_viewpoint.tvp_tsn_act->forall(a I}
\text{m.inm_tsn_viewpoint.tvp_tsn_act->includes(a))}
\text{implies}
\text{m.inm_tsn_action->includes(a)}
\]
\text{Interaction Model}
\text{self.allInstances->forall(m)}
\text{self.inm_tsn_viewpoint.tvp_tsn_tim->forall(t)}
\text{m.inm_tsn_viewpoint.tvp_tsn_tim->includes(t))}
\text{implies}
\text{m.inm_tsn_timing->includes(t)}

c) Rule for population of the three event types.

\text{Interaction Model}
\text{self.allInstances->forall(m)}
\text{self.inm_scenario.scenario_event->forall(c)}
\text{m.inm_scenario.scenario_event->includes(c) and m.oclType = CommunicationEvent)}
\text{implies}
\text{m.inm_communication_event->includes(c)}

\text{Interaction Model}
\text{self.allInstances->forall(m)}
\text{self.inm_scenario.scenario_event->forall(a)}
\text{m.inm_scenario.scenario_event->includes(a) and a.oclType = InternalActionEvent)}
\text{implies}
\text{m.inm_action_event->includes(a)}

\text{Interaction Model}
\text{self.allInstances->forall(m)}
\text{self.inm_scenario.scenario_event->forall(t)}
\text{m.inm_scenario.scenario_event->includes(t) and t.oclType = TimingEvent)}
\text{implies}
\text{m.inm_timing_event->includes(t)}
This page deliberately left blank
Appendix B

Appendix - B
This page deliberately left blank
Appendix B (Part 1)

'BATrA Framework (Revised)'
Appendix B (Part 2)

AerospaceTraceabilityEntity (abstract)

<<Structure Element>>
MoreImportantThan

<<Structure Element>>
RaisesIssue

<<Structure Element>>
ResolvesIssue

<<Structure Element>>
Qualifies (abstract)

<<Structure Element>>
Issue

<<Structure Element>>
Supports

<<Structure Element>>
Denies

<<Structure Element>>
Presupposes

'Argumentation Meta-model'
Complete O-Telos Representations for MATrA Configuration Model

Worked Example

Instantiation of elements in figure 5.12

A320 in AerospaceEngineeringProject, Token with
  project_title
  projectTitle: "A320 Project"
  has_pds
  hasPDS: A320PDS
end

A320PDS in ProductDataSynthesis, Token with
  build_entity
    buildEntity1: AirbusA320;
    buildEntity2: GrossWingArea;
    buildEntity3: LandingDistance;
    buildEntity4: WingLoading;
    buildEntity5: FuelCapacity;
    buildEntity6: MaximumLiftCoefficient;
    buildEntity7: Payload;
    buildEntity8: Range;
    buildEntity9: TouchDownSpeed;
    buildEntity10: ApproachSpeed;
    buildEntity11: Stall ingSpeed;
    buildEntity12: MTOW
end

AirbusA320 in Module, Token with
  moduleName
  moduleName: "A320"
end

A320HasVer100 in HasRevision, Token with
  bd_source
  bdSource: AirbusA320
  bd_target
  bdTarget: Dash100
end

Dash100 in Version, Token with
  version_name
  versionName: "A320-100"
end

Dash100HasPropertyWingLoading in HasProperty, Token with
  bd_source
    bdSource: Dash100
  bd_target
    bdTarget: WingLoading
end

WingLoading in Property, Token with
  property_name
    propertyName: "Wing Loading"
end

Dash100HasPropertyFuelCapacity in HasProperty, Token with
  bd_source
    bdSource: Dash100
  bd_target
    bdTarget: FuelCapacity
end

FuelCapacity in Property, Token with
  property_name
    propertyName: "Fuel Capacity"
end

Dash100HasPropertyMaximumLiftCoefficient in HasProperty, Token with
  bd_source
    bdSource: Dash100
  bd_target
    bdTarget: MaximumLiftCoefficient
end

MaximumLiftCoefficient in Property, Token with
  property_name
    propertyName: "Maximum Lift Coefficient"
end

Dash100HasPropertyPayload in HasProperty, Token with
  bd_source
    bdSource: Dash100
  bd_target
    bdTarget: Payload
end

Payload in Property, Token with
  property_name
    propertyName: "Payload"
end

Dash100HasPropertyGrossWingArea in HasProperty, Token with
  bd_source
    bdSource: Dash100
  bd_target
    bdTarget: GrossWingArea
end

GrossWingArea in Property, Token with
  property_name
    propertyName: "Gross Wing Area"
end

Dash100HasPropertyLandingDistance in HasProperty, Token with
  bd_source
    bdSource: Dash100
  bd_target
    bdTarget: LandingDistance
end

LandingDistance in Property, Token with
  property_name
    propertyName: "Landing Distance"
end

Dash100HasPropertyRange in HasProperty, Token with
  bd_source
    bdSource: Dash100
  bd_target
    bdTarget: Range
end

Range in Property, Token with
  property_name
    propertyName: "Range"
end

Dash100HasPropertyTouchDownSpeed in HasProperty, Token with
  bd_source
    bdSource: Dash100
Appendix B (Part 3)

bd_target
bdTarget : TouchDownSpeed
end

TouchDownSpeed in Property, Token with
property_name
  propertyName : "Touch Down Speed"
end

Dash100HasPropertyApproachSpeed in
HasProperty, Token with
  bd_source
    bdSource : Dash100
  bd_target
    bdTarget : ApproachSpeed
end

ApproachSpeed in Property, Token with
property_name
  propertyName : "Approach Speed"
end

Dash100HasPropertyStallingSpeed in
HasProperty, Token with
  bd_source
    bdSource : Dash100
  bd_target
    bdTarget : StallingSpeed
end

StallingSpeed in Property, Token with
property_name
  propertyName : "Stalling Speed"
end

Dash100HasPropertyMTOW in HasProperty,
Token with
  bd_source
    bdSource : Dash100
  bd_target
    bdTarget : MTOW
end

MTOW in Property, Token with
property_name
  propertyName : "Maximum Takeoff
Weight"
end

MTOWHasSpecificationSpec1 in
HasSpecification, Token with
  bd_source
    bdSource : MTOW
  bd_target
    bdTarget : MTOWSpec1
end

MTOWSpec1 in Specification, Token with
value_specification
  valueSpecification : "75 000 kg"
end

MTOWHasSpecificationSpec2 in
HasSpecification, Token with
value_specification
  valueSpecification : "75 500 kg"
end

MTOWHasSpecificationSpec3 in
HasSpecification, Token with
value_specification
  valueSpecification : "76 000 kg"
end

MTOWSpecSucceeds1 in Succeeds, Token with
  bd_source
    bdSource : MTOWSpec2
  bd_target
    bdTarget : MTOWSpec1
end

MTOWSpecSucceeds2 in Succeeds, Token with
  bd_source
    bdSource : MTOWSpec3
  bd_target
    bdTarget : MTOWSpec2
end

Instantiation of elements in figure 5.13

FlightControlSystem in Module, Token with
  moduleName : "FCS"
end

FCSHasVerDash100FCS in HasRevision, Token with
  bd_source
    bdSource : FlightControlSystem
  bd_target
    bdTarget : Dash100FCS
end

Dash100FCS in Version, Token with
  version_name
    versionName : "-100 FCS"
end

Dash100FCSHasFTsubmoduleSEC1 in
HasFTSubmodule, Token with
  bd_source
    bdSource : Dash100FCS
  bd_target
    bdTarget : SEC1
end

SEC1 in Module, Token with
  moduleName : "SEC1"
end

Dash100FCSHasFTsubmoduleSEC2 in
HasFTSubmodule, Token with
  bd_source
    bdSource : Dash100FCS
  bd_target
    bdTarget : SEC2
end

SEC2 in Module, Token with
  moduleName : "SEC2"
end

Dash100FCSHasFTsubmoduleELAC1 in
HasFTSubmodule, Token with
  bd_source
    bdSource : Dash100FCS
  bd_target
    bdTarget : ELAC1
end

ELAC1 in Module, Token with
  moduleName : "ELAC1"
Appendix B (Part 3)

Dash100FCSHasFTsubmoduleELAC2 in HasFTSubmodule, Token with
  bd_source : Dash100FCS
  bd_target : ELAC2
end

ELAC2 in Module, Token with
  moduleName : "ELAC2"
end

Dash100FCSHasFTsubmoduleSFCC1 in HasFTSubmodule, Token with
  bd_source : Dash100FCS
  bd_target : SFCC1
end

SFCC1 in Module, Token with
  moduleName : "SFCC1"
end

Dash100FCSHasFTsubmoduleSFCC2 in HasFTSubmodule, Token with
  bd_source : Dash100FCS
  bd_target : SFCC2
end

SFCC2 in Module, Token with
  moduleName : "SFCC2"
end

Dash100FCSHasVerDash100SFCC1 in HasRevision, Token with
  bd_source : Dash100FCS
  bd_target : Dash100SFCC1
end

Dash100SFCC1 in Version, Token with
  versionName : "Dash100SFCC1"
end

Dash100SFCC1HasSubmoduleFlapChannel in HasSubmodule, Token with
  bd_source : Dash100SFCC1
  bd_target : FlapChannel
end

FlapChannel in Module, Token with
  moduleName : "Flap Channel"
end

Dash100SFCC1HasSubmoduleSlatChannel in HasSubmodule, Token with
  bd_source : Dash100SFCC1
  bd_target : SlatChannel
end

SlatChannel in Module, Token with
  moduleName : "Slat Channel"
end

SlatChannelHasVerSlatChannel1 in HasRevision, Token with
  bd_source : SlatChannel
  bd_target : SlatChannel1
end

SlatChannel1 in Version, Token with
  versionName : "Slat Channel-1"
end

SlatChannel1HasFTsubmoduleSCLanel in HasFTSubmodule, Token with
  bd_source : SlatChannel1
  bd_target : SlatChannelLanel
end

SlatChannelLanel in Module, Token with
  moduleName : "SCLanel"
end

SlatChannelLanelHasVerSlatChannelLanel1 in HasRevision, Token with
  bd_source : SlatChannelLanel
  bd_target : SlatChannelLanel1
end

SlatChannelLanel1 in Version, Token with
  versionName : "SCLanel-1"
end

SlatChannelLanel1EncapsulatesInhibitSlatRetraction in Encapsulates, Token with
  bd_source : SlatChannelLanel1
  bd_target : InhibitSlatRetraction
end

InhibitSlatRetraction in Function, Token with
  functionName : "Inhibit Slat Retraction"
end

InhibitSlatRetractionHasVerInhibitSlatRetraction1 in HasRevision, Token with
  bd_source : InhibitSlatRetraction1
  bd_target : InhibitSlatRetraction
end

InhibitSlatRetraction1 in Version, Token with
  versionName : "Inhibit Slat Retraction-1"
end

InhibitSlatRetraction1HasConditionDisable in HasCondition, Token with
  bd_source :
Appendix B (Part 3)

```plaintext
bdSource : InhibitSlatRetractioni
bd_target
bdTarget : Disabled
end

Disabled in Condition, Token with
condition_label
conditionLabel : "Disabled"
end

InhibitSlatRetractioniHasConditionNotEngaged in HasCondition, Token with
bd_source
bdSource : InhibitSlatRetractioni
bd_target
bdTarget : NotEngaged
end

NotEngaged in Condition, Token with
condition_label
conditionLabel : "Not-Engaged"
end

InhibitSlatRetractioniHasConditionSpeedBaulk in HasCondition, Token with
bd_source
bdSource : InhibitSlatRetractioni
bd_target
bdTarget : SpeedBaulk
end

SpeedBaulk in Condition, Token with
condition_label
conditionLabel : "Speed-Baulk"
end

InhibitSlatRetractioniHasConditionAlphaLock in HasCondition, Token with
bd_source
bdSource : InhibitSlatRetractioni
bd_target
bdTarget : AlphaLock
end

AlphaLock in Condition, Token with
condition_label
conditionLabel : "Alpha-Lock"
end

InhibitSlatRetractioniHasSubfunctionISRLowSpeed in HasSubfunction, Token with
bd_source
bdSource : InhibitSlatRetractioni
bd_target
bdTarget : ISRLowSpeed
end

ISRLowSpeed in Function, Token with
function_name
functionName : "ISR-Low Speed"
end

InhibitSlatRetractioniHasSubfunctionISRHighAoA in HasSubfunction, Token with
bd_source
bdSource : InhibitSlatRetractioni
bd_target
bdTarget : ISRHighAoA
end

ISRHighAoA in Function, Token with
function_name
functionName : "ISR-High Angle of
Attack"
end

Instantiation of elements in figure 5.14

ISRHighAoAHasVerISRHighAoAi in
HasRevision, Token with

bdSource : ISRHighAoA
bd_target
bdTarget : ISRHighAoAi
end

ISRHighAoAi in Version, Token with
version_name
versionName : "ISR-High Angle of Attack-i"
end

AlphaLockHasVerAlphaLocki in HasRevision, Token with
bd_source
bdSource : AlphaLock
bd_target
bdTarget : AlphaLocki
end

AlphaLocki in Version, Token with
version_name
versionName : "Alpha-Lock-i"
end

AlphaLockiHasSubconditionEngagedAlphaLock in HasSubcondition, Token with
bd_source
bdSource : AlphaLocki
bd_target
bdTarget : EngagedAlphaLock
end

EngagedAlphaLock in Condition, Token with
condition_label
conditionLabel : "Engaged Alpha-Lock"
end

EngagedAlphaLockHasVerEngagedAlphaLocki in HasRevision, Token with
bd_source
bdSource : EngagedAlphaLock
bd_target
bdTarget : EngagedAlphaLocki
end

EngagedAlphaLocki in Version, Token with
version_name
versionName : "Engaged Alpha-Lock-i"
end

EngagedAlphaLockiHasSubconditionReleaseAlphaLock in HasSubcondition, Token with
bd_source
bdSource : EngagedAlphaLocki
bd_target
bdTarget : ReleaseAlphaLock
end

ReleaseAlphaLock in Condition, Token with
condition_label
conditionLabel : "Release Alpha-Lock"
end

ReleaseAlphaLockHasVerReleaseAlphaLocki in HasRevision, Token with
bd_source
bdSource : ReleaseAlphaLock
bd_target
bdTarget : ReleaseAlphaLocki
end

ReleaseAlphaLocki in Version, Token with
version_name
versionName : "Release Alpha-Lock-i"
end

ISRHighAoAiConsumesCAoA in
ConsumesExternalIO, Token with
bd_source
bdSource : ISRHighAoAi
```
Appendix B (Part 3)

bd_target
  bdTarget : CAoA
end
CAoA in InputOutput, Token with
  flow_name
  flowName : "CAoA"
end
CAoAHasVerCAoAi in HasRevision, Token with
  bd_source
  bdSource : CAoA
  bd_target
  bdTarget : CAoAi
end
CAoAi in Version, Token with
  version_name
  versionName "CAoAi"
end
CAoAICarriesConditionCAoA7PtlEvent in
  CarriesCondition, Token with
  bd_source
  bdSource : CAoAi
  bd_target
  bdTarget : CAoA7PtlEvent
end
CAoA7PtlEvent in Condition, Token with
  condition_label
  conditionLabel : "CAoA < 7.1 degrees"
end
CAoA7PtlEventOccurringInEngagedAlphaLock in
  OccurringIn, Token with
  bd_source
  bdSource : CAoA7PtlEvent
  bd_target
  bdTarget : EngagedAlphaLock
end
CAoA7PtlEventLeadsToReleaseAlphaLock in
  LeadsTo, Token with
  bd_source
  bdSource : CAoA7PtlEvent
  bd_target
  bdTarget : ReleaseAlphaLock
end

Instantiation of elements in figure 5.15

ISRLowSpeedHasVerISRLowSpeedi in
  HasRevision, Token with
  bd_source
  bdSource : ISRLowSpeed
  bd_target
  bdTarget : ISRLowSpeedi
end
ISRLowSpeedi in Version, Token with
  version_name
  versionName "ISRLow Speed-i"
end
SpeedBaulkHasVerSpeedBaulki in
  HasRevision, Token with
  bd_source
  bdSource : SpeedBaulk
  bd_target
  bdTarget : SpeedBaulki
end
SpeedBaulki in Version, Token with
  version_name
  versionName : "Speed-Baulk-i"
end
SpeedBaulkiHasSubconditionEngagedSpeedBaulk in
  HasSubcondition, Token with
  bd_source
  bdSource : SpeedBaulki
  bd_target
  bdTarget : EngagedSpeedBaulk
end
EngagedSpeedBaulk in Condition, Token with
  condition_label
  conditionLabel : "Engaged Speed-Baulk"
end
EngagedSpeedBaulkHasVerEngagedSpeedBaulki in
  HasRevision, Token with
  bd_source
  bdSource : EngagedSpeedBaulk
  bd_target
  bdTarget : EngagedSpeedBaulki
end
EngagedSpeedBaulki in Version, Token with
  version_name
  versionName : "Engaged Speed-Baulk-i"
end
EngagedSpeedBaulkiHasSubconditionReleaseSpeedBaulk in
  HasSubcondition, Token with
  bd_source
  bdSource : EngagedSpeedBaulk
  bd_target
  bdTarget : ReleaseSpeedBaulk
end
ReleaseSpeedBaulk in Condition, Token with
  condition_label
  conditionLabel : "Release Speed-Baulk"
end
ReleaseSpeedBaulkHasVerReleaseSpeedBaulki in
  HasRevision, Token with
  bd_source
  bdSource : ReleaseSpeedBaulk
  bd_target
  bdTarget : ReleaseSpeedBaulki
end
ReleaseSpeedBaulki in Version, Token with
  version_name
  versionName : "Release Speed-Baulk-i"
end
ReleaseSpeedBaulkiConsumesCAS in
  ConsumesExternalIO, Token with
  bd_source
  bdSource : ReleaseSpeedBaulki
  bd_target
  bdTarget : CAS
end
CAS in InputOutput, Token with
  flow_name
  flowName : "CAS"
end
CASHasVerCASi in HasRevision, Token with
  bd_source
  bdSource : CAS
  bd_target
  bdTarget : CASi
end

B-7
Appendix B (Part 3)

Instantiation of elements in figure 5.14

ISRHighAoA HasRevision, Token with
version_name
versionName : "ISR-High Angle of Attack-i"
end

AlphaLock HasRevision, Token with
version_name
versionName : "Alpha-Lock-i"
end

ISRLowSpeed HasRevision, Token with
version_name
versionName : "ISR-Low Speed"
end

ISRHighAoA HasRevision, Token with
version_name
versionName : "ISR-High Angle of Attack"
end

ReleaseAlphaLock HasRevision, Token with
version_name
versionName : "Release Alpha-Lock"
end

ISRHighAoA ConsumesExternalIO, Token with
bd_source
bdSource : ISRHighAoA

ISRHighAoAi ConsumesExternalIO, Token with
bd_source
bdSource : ISRHighAoAi

ISRHighAoA ConsumesExternalIO, Token with
bd_source
bdSource : ISRHighAoAi

ISRHighAoAi ConsumesExternalIO, Token with
bd_source
bdSource : ISRHighAoAi
Instantiation of elements in figure 5.15

Appendix B (Part 3)

bd_target
  bdTarget : CAoA
end

CAoA in InputOutput, Token with
  flow_name : "CAoA"
end

CAoAHasVerCAoAi in HasRevision, Token with
  bd_source
    bdSource : CAoA
  bd_target
    bdTarget : CAoAi
end

CAoAi in Version, Token with
  version_name : "CAoAi"
end

CAoAiCarriesConditionCAoA7Pt1Event in
  CarriesCondition, Token with
    bd_source
      bdSource : CAoAi
    bd_target
      bdTarget : CAoA7Pt1Event
end

CAoA7Pt1Event in Condition, Token with
  condition_label : "CAoA < 7.1
  degrees"
end

CAoA7Pt1EventOccurringInEngagedAlphaLock in
  OccurringIn, Token with
    bd_source
      bdSource : CAoA7Pt1Event
    bd_target
      bdTarget : EngagedAlphaLock
end

CAoA7Pt1EventLeadsToReleaseAlphaLock in
  LeadsTo, Token with
    bd_source
      bdSource : CAoA7Pt1Event
    bd_target
      bdTarget : ReleaseAlphaLock
end

ISRLowSpeedHasVerISRLowSpeedi in
  HasRevision, Token with
    bd_source
      bdSource : ISRLowSpeed
    bd_target
      bdTarget : ISRLowSpeedi
end

ISRLowSpeedi in Version, Token with
  version_name : "ISRLow-Speed-i"
end

SpeedBaulkHasVerSpeedBaulki in
  HasRevision, Token with
    bd_source
      bdSource : SpeedBaulk
    bd_target
      bdTarget : SpeedBaulki
end

SpeedBaulki in Version, Token with
  version_name
  StringBuffer
    versionName : "Speed-Baulk-i"
end

SpeedBaulkiHasSubconditionEngagedSpeedBaulk in
  HasSubcondition, Token with
    bd_source
      bdSource : SpeedBaulki
    bd_target
      bdTarget : EngagedSpeedBaulk
end

EngagedSpeedBaulk in Condition, Token with
  condition_label : "Engaged Speed-
  Baulk"
end

EngagedSpeedBaulkHasVerEngagedSpeedBaulki in
  HasRevision, Token with
    bd_source
      bdSource : EngagedSpeedBaulk
    bd_target
      bdTarget : EngagedSpeedBaulki
end

EngagedSpeedBaulki in Version, Token with
  version_name : "Engaged Speed-Baulk-i"
end

EngagedSpeedBaulkiHasSubconditionReleaseSpeedBaulk in
  HasSubcondition, Token with
    bd_source
      bdSource : SpeedBaulk
    bd_target
      bdTarget : ReleaseSpeedBaulk
end

ReleaseSpeedBaulk in Condition, Token with
  condition_label : "Release Speed-
  Baulk"
end

ReleaseSpeedBaulkHasVerReleaseSpeedBaulki in
  HasRevision, Token with
    bd_source
      bdSource : ReleaseSpeedBaulk
    bd_target
      bdTarget : ReleaseSpeedBaulki
end

ReleaseSpeedBaulki in Version, Token with
  version_name
  StringBuffer
    versionName : "Release Speed-Baulk-i"
end

ISRLowSpeediConsumesCAS in
  ConsumesExternalIO, Token with
    bd_source
      bdSource : ISRLowSpeedi
    bd_target
      bdTarget : CAS
end

CAS in InputOutput, Token with
  flow_name : "CAS"
end

CASHasVerCASI in HasRevision, Token with
  bd_source
    bdSource : CAS
  bd_target
    bdTarget : CASI
end

CASI in Version, Token with
  version_name
  StringBuffer
    versionName : "CASI"
end
Appendix B (Part 3)

CASi in Version, Token with
  version_name
    versionName "CASi"
end

CASiCarriesConditionCAS154KtsEvent in
  CarriesCondition, Token with
    bd_source
      bdSource : CASi
    bd_target
      bdTarget : CAS154KtsEvent
end

CAS154KtsEvent in Condition, Token with
  condition_label
    conditionLabel : "CAS > 154 knots"
end

CAS154KtsEventOccurringInEngagedSpeedBaulk in
  OccurringIn, Token with
    bd_source
      bdSource : CAS154KtsEvent
    bd_target
      bdTarget : EngagedSpeedBaulk
end

CAS154KtsEventLeadsToReleaseSpeedBaulk in
  LeadsTo, Token with
    bd_source
      bdSource : CAS154KtsEvent
    bd_target
      bdTarget : ReleaseSpeedBaulk
end
Statechart Meta-model: UML and O-Telos Base Classes

Event in StructureElement, SimpleClass isA AerospaceEngineeringObject with
  has_property
    event_description : String
  end

StateVertex in StructureElement, SimpleClass isA AerospaceEngineeringObject end

Transition in StructureElement, SimpleClass isA AerospaceEngineeringObject with
  has_part
    source : StateVertex;
    target : StateVertex;
    trigger : Event
  end

State in StructureElement, SimpleClass isA StateVertex with
  has_property
    state_name : String
  end

InitialState in StructureElement, SimpleClass isA State end

SubMachineState in StructureElement, SimpleClass isA State with
  has_structure
    sub_machne : StateMachine
  end

StateMachine in DevelopmentStructure, SimpleClass isA AerospaceEngineeringObject with
  has_property
    subject : ModelElement
    has_element
      state : State;
      transition : Transition
  end

ModelElement in ArtifactProperty, SimpleClass isA String end

'State Machine Meta-model'
This page deliberately left blank
Appendix C (Part 1) - Case Study I: A Hypothetical Mission Planning System for the Hawk 100 and 200 Series Aircraft (Complete O-Telos Representation of Artifacts from 6.2)

'Erase Cartridge Use Case Diagram'

'Retrieve From Cartridge Use Case Diagram'

'Choose Mission and Aircraft Use Case Diagram'
I. Instantiation of Use Case View for Erase Cartridge, Retrieve from Cartridge and Choose Mission & Aircraft

Instantiation of UseCaseModel Elements

- Actor Instances

  Pilot in Actor, Token with
  actor_name
  actorName : "Pilot"
end

  Cartridge in Actor, Token with
  actor_name
  actorName : "Cartridge"
end

  MPS Printer in Actor, Token with
  actor_name
  actorName : "MPS Printer"
end

  Mission Plan in Actor, Token with
  actor_name
  actorName : "Mission Plan"
end

- Text Nodes (for specifying Pre and Post Conditions)

  Cartridge in ModuleNode, Token with
  module_name
  moduleName : "Cartridge"
end

  Mission Planning System in ModuleNode, Token with
  module_name
  moduleName : "MPS"
end

  Pilot in ModuleNode, Token with
  module_name
  moduleName : "Pilot"
end

  MPS Printer in ModuleNode, Token with
  module_name
  moduleName : "MPS Printer"
end

  Mission Plan in ModuleNode, Token with
  module_name
  moduleName : "Mission Plan"
end

- Use Cases (including Pre & Post Conditions)

  Erase Cartridge

  Erase Cartridge in UseCase, Token with
  use_case_name
  useCaseName : "Erase Cartridge"
sub_case
  subCase : False
post_condition
    postCondition1 : ECPost1NLS
end

  Post Condition for Erase Cartridge

  ECPost1NLS in
  MatraNaturalLanguageStructure, Token with
  mnlS_composite
    mnlSComposite1 : ECPost1C
    mnlS_plain_text
      mnlSPlainText1 : ECPostPT1;
      mnlSPlainText2 : ECPostPT2
    mnlS_module
      mnlSModule1 : CartridgeNode
end

  ECPost1C in MatraNLSComposite, Token with
  preceding_fragment
    precedingFragment : ECPostPT1
  subject_node
    subjectNode : CartridgeNode
  following_fragment
    followingFragment : ECPostPT2
end

  ECPostPT1 in PlainTextNode, Token with
  mnlS_text
    mnlSText : "The data on the "
end

  ECPostPT2 in PlainTextNode, Token with
  mnlS_text
    mnlSText : " has been erased."
end

  Check for Cartridge

  Check for Cartridge in UseCase, Token with
  use_case_name
    useCaseName : "Check for Cartridge"
sub_case
    subCase : True
post_condition
    postCondition1 : CfCPost1NLS
end

  Post Condition for Check for Cartridge

  CfCPost1NLS in
  MatraNaturalLanguageStructure, Token with
  mnlS_composite
    mnlSComposite1 : CfCPostC1;
    mnlSComposite2 : CfCPostC2
    mnlS_plain_text
      mnlSPlainText1 : CfCPostPT1;
      mnlSPlainText2 : CfCPostPT2;
      mnlSPlainText3 : CfCPostPT3
    mnlS_module
      mnlSModule1 : MissionPlanningSystemNode;
      mnlSModule2 : CartridgeNode
end

  CfCPostC1 in MatraNLSComposite, Token with
  preceding_fragment
    precedingFragment : CfCPostPT1
  subject_node
    subjectNode : MissionPlanningSystemNode
  following_fragment
    followingFragment : CfCPostC2
Appendix C (Part 1)

CfCPostPT1 in PlainTextNode, Token with
mlns_text : "The ".
end

CfCPostC2 in MatraNLSComposite, Token with
preceding_fragment : CfCPostPT2
subject_node
subjectNode : CartridgeNode
following_fragment : CfCPostPT3
end

CfCPostPT2 in PlainTextNode, Token with
mlns_text : " has been informed that there is a ".
end

CfCPostPT3 in PlainTextNode, Token with
mlns_text : " present in the hardware."
end

Check Cartridge for Data

CheckCartridgeforData in UseCase, Token with
use_case_name : "Check Cartridge for Data"
sub_case : True
pre_condition : CfDPre1NLS
post_condition : CfDPost1NLS
end

PreCondition for Check Cartridge for Data

CfDPre1NLS in MatraNaturalLanguageStructure, Token with
mlns_composite
mlnsComposite1 : CfDPreC1
mlnsPlain_text
mlnsPlainText1 : CfDPrePT1;
mlnsPlainText2 : CfDPrePT2
mlns_module
mlnsModule1 : CartridgeNode
end

CfDPreC1 in MatraNLSComposite, Token with
preceding_fragment : CfDPrePT1
subject_node
subjectNode : CartridgeNode
following_fragment : CfDPrePT2
end

CfDPrePT1 in PlainTextNode, Token with
mlns_text : "There is a ".
end

CfDPrePT2 in PlainTextNode, Token with
mlns_text : " present in the hardware."
end

Post Condition for Check Cartridge for Data

CfDPost1NLS in MatraNaturalLanguageStructure, Token with
mlns_composite
mlnsComposite1 : CfDPostC1;
mlnsComposite2 : CfDPostC2
mlnsPlain_text
mlnsPlainText1 : CfDPostPT1;
mlnsPlainText2 : CfDPostPT2;
mlnsPlainText3 : CfDPostPT3
mlns_module
mlnsModule1 : MissionPlanningSystemNode;
mlnsModule2 : CartridgeNode
end

CfDPostC1 in MatraNLSComposite, Token with
precedingFragment : CfDPostPT1
subject_node
subjectNode : MissionPlanningSystemNode
following_fragment : CfDPostPT3
end

CfDPostPT1 in PlainTextNode, Token with
mlns_text : "The ".
end

CfDPostC2 in MatraNLSComposite, Token with
precedingFragment : CfDPostPT2
subject_node
subjectNode : CartridgeNode
following_fragment : CfDPostPT3
end

CfDPostPT2 in PlainTextNode, Token with
mlns_text : " has been informed whether or not there is data loaded on
the ".
end

CartridgeNode in ModuleNode, Token with
module_name : "Cartridge"
end

CfDPrePT3 in PlainTextNode, Token with
mlns_text : "."
end

Erase Data from Cartridge

EraseDatafromCartridge in UseCase, Token with
use_case_name : "Erase Data from Cartridge"
sub_case : True
pre_condition : EdfCPre1NLS;
post_condition : EdfCPost1NLS
end

C-3
Pre Conditions for Erase Data from Cartridge

Pre Condition 1

EDfCP1NLS in MatraNaturalLanguageStructure, Token with
mnls_composite
  mnlsComposite1 : EDfCP1C1;
  mnlsComposite2 : EDfCP1C2
mnls_plain_text
  mnlsPlainText1 : EDfCP1PT1;
  mnlsPlainText2 : EDfCP1PT2;
  mnlsPlainText3 : EDfCP1PT3
mnls_module
  mnlsModule1 : PilotNode;
  mnlsModule2 : CartridgeNode
end

EDfCP1C1 in MatraNLSComposite, Token with
preceeding_fragment : EDfCP1PT1
subject_node
  subjectNode : PilotNode
following_fragment : EDfCP1C2
end

EDfCP1PT1 in PlainTextNode, Token with
mnls_text
  mnlsText : "The 
end

EDfCP1C2 in MatraNLSComposite, Token with
preceeding_fragment : EDfCP1PT2
subject_node
  subjectNode : CartridgeNode
following_fragment : EDfCP1PT3
end

EDfCP1PT2 in PlainTextNode, Token with
mnls_text
  mnlsText : " has confirmed that he 
end

EDfCP1PT3 in PlainTextNode, Token with
mnls_text
  mnlsText : "." 
end

Pre Condition 2

EDfCP2NLS in MatraNaturalLanguageStructure, Token with
mnls_composite
  mnlsComposite1 : EDfCP2C1
mnls_plain_text
  mnlsPlainText1 : EDfCP2PT1;
  mnlsPlainText2 : EDfCP2PT2
mnls_module
  mnlsModule1 : CartridgeNode
end

EDfCP2C1 in MatraNLSComposite, Token with
preceeding_fragment : EDfCP2PT1
subject_node
  subjectNode : CartridgeNode
following_fragment : EDfCP2PT2
end

EDfCP2PT1 in PlainTextNode, Token with
mnls_text
  mnlsText : "The 
end

EDfCP2PT2 in PlainTextNode, Token with
mnls_text
  mnlsText : " has to contain data." 
end

Post Condition for Erase Data from Cartridge

EDfCP2NLS in MatraNaturalLanguageStructure, Token with
mnls_composite
  mnlsComposite1 : EDfCP2C1
mnls_plain_text
  mnlsPlainText1 : EDfCP2PT1;
  mnlsPlainText2 : EDfCP2PT2
mnls_module
  mnlsModule1 : PilotNode;
end

EDfCP2C1 in MatraNLSComposite, Token with
preceeding_fragment : EDfCP2PT1
subject_node
  subjectNode : CartridgeNode
following_fragment : EDfCP2PT2
end

EDfCP2PT1 in PlainTextNode, Token with
mnls_text
  mnlsText : "has confirmed that he 
end

EDfCP2PT2 in PlainTextNode, Token with
mnls_text
  mnlsText : "." 
end

Retrieve from Cartridge

RetrievefromCartridge in UseCase, Token with
use_case_name
  useCaseName : "Retrieve from Cartridge"
sub_case
  subCase : False
post_condition
  postCondition1 : RfCPost1NLS;
  postCondition2 : RfCPost2NLS
end

Post Conditions for Retrieve from Cartridge

Post Condition 1

RfCPost1NLS in MatraNaturalLanguageStructure, Token with
mnls_composite
  mnlsComposite1 : RfCPost1C1;
  mnlsComposite2 : RfCPost1C2
mnls_plain_text
  mnlsPlainText1 : RfCPost1PT1;
  mnlsPlainText2 : RfCPost1PT2;
  mnlsPlaintext3 : RfCPost1PT3
mnls_module
  mnlsModule1 : CartridgeNode;
  mnlsModule2 : MissionPlanningSystemNode
end

RfCPost1C1 in MatraNLSComposite, Token with
Appendix C (Part 1)

Post Condition for Select Data to Retrieve

SDtRPost1NLS in MatraNaturalLanguageStructure, Token with
  mnsPlainText1: SDtRPostPT1
end

SDLRPostPT1 in PlainTextNode, Token with
  mnsText: "The data types to be retrieved have been defined."
end

Retrieve Data from Cartridge

RetrieveDataFromCartridge in UseCase, Token with
  use_case_name: "Retrieve Data from Cartridge"
  sub_case: True
  pre_condition: RfCPre1NLS
  post_condition: RfCPost1NLS
end

Pre Condition for Retrieve Data from Cartridge

RfCPre1NLS in MatraNaturalLanguageStructure, Token with
  mns_composite
    mnsComposite1: RfCPreC1
    mnsPlainText1: RfCPrePT1;
    mnsPlainText2: RfCPrePT2
    mnsModule: mnsModule1: CartridgeNode
end

RfCPreC1 in MatraNLSComposite, Token with
  preceding_fragment
    precedingFragment: RfCPrePT1
    subject_node: CartridgeNode
  following_fragment
    followingFragment: RfCPrePT2
end

RfCPrePT1 in PlainTextNode, Token with
  mnsText: "A containing data is present in the hardware."
end

RfCPrePT2 in PlainTextNode, Token with
  mnsText: "A"
end

Post Condition for Retrieve Data from Cartridge

RfCPost1NLS in MatraNaturalLanguageStructure, Token with
  mns_composite
    mnsComposite1: RfCPostC1;
    mnsComposite2 : RfCPostC2
end

Select Data to Retrieve

SelectDataToRetrieve in UseCase, Token with
  use_case_name: "Select Data to Retrieve"
  sub_case: True
  post_condition: SdtrPost1NLS
end
Display Cartridge Data

DisplayCartridgeData in UseCase, Token with
use_case_name : "Display Cartridge Data"
sub_case : True
pre_condition
preCondition1 : DCDPre1NLS
post_condition
postCondition1 : DCDPost1NLS
end

Pre Condition for Display Cartridge Data

DCDPre1NLS in MatraNaturalLanguageStructure, Token with
mnlsexp_composite
mnlsexpComposite1 : DCDPreC1;
mnlsexpComposite2 : DCDPreC2
mnlsexp_plain_text
mnlsexpPlainText1 : DCDPrePT1;
mnlsexpPlainText2 : DCDPrePT2;
mnlsexpPlainText3 : DCDPrePT3
mnlsexp_module
mnlsexpModule1 : CartridgeNode;
mnlsexpModule2 : MissionPlanningSystemNode
end

DCDPreC1 in MatraNLSComposite, Token with
preceeding_fragment
preceedingFragment : DCDPrePT1
subject_node
subjectNode : CartridgeNode
followingFragment
followingFragment : DCDPreC2
end

DCDPrePT1 in MatraTextNode, Token with
mnlsexp_text
mnlsexpText : "Data has been retrieved from the "
end

DCDPreC2 in MatraNLSComposite, Token with
preceeding_fragment
preceedingFragment : DCDPrePT2
subject_node
subjectNode : MissionPlanningSystemNode
following_fragment
followingFragment : DCDPreC3
end

DCDPrePT2 in MatraTextNode, Token with
mnlsexp_text
mnlsexpText : " and stored on the "
end

DCDPrePT3 in MatraTextNode, Token with
mnlsexp_text
mnlsexpText : ".
end

Post Condition for Display Cartridge Data

DCDPost1NLS in MatraNaturalLanguageStructure, Token with
mnlsexp_composite
mnlsexpComposite1 : DCDPostC1;
mnlsexpComposite2 : DCDPostC2
mnlsexp_plain_text
mnlsexpPlainText1 : DCDPostPT1;
mnlsexpPlainText2 : DCDPostPT2;
mnlsexpPlainText3 : DCDPostPT3
mnlsexp_module
mnlsexpModule1 : CartridgeNode;
mnlsexpModule2 : PilotNode
end

DCDPostC1 in MatraNLSComposite, Token with
preceeding_fragment
preceedingFragment : DCDPostPT1
subject_node
subjectNode : CartridgeNode
followingFragment
followingFragment : DCDPostC2
end

DCDPostPT1 in MatraTextNode, Token with
mnlsexp_text
mnlsexpText : "The data that is currently loaded onto the "
end

DCDPostC2 in MatraNLSComposite, Token with
preceeding_fragment
preceedingFragment : DCDPostPT2
subject_node
subjectNode : PilotNode
following_fragment
followingFragment : DCDPostC3
end

DCDPostPT2 in MatraTextNode, Token with
mnlsexp_text
mnlsexpText : " is displayed for the "
end

DCDPostPT3 in MatraTextNode, Token with
mnlsexp_text
mnlsexpText : ".
end
Appendix C (Part 1)

Print Data

PrintData in UseCase, Token with
use_case_name
  useCaseName : "Print Data"
sub_case
  subCase : True
pre_condition
    preCondition1 : PDPre1NLS
post_condition
    postCondition1 : PDPost1NLS
end

Pre Condition for Print Data

PDPre1NLS in
MatraNaturalLanguageStructure, Token with
  mnl_composite
    mnlCS1
      mnlPlainText1 : PDPrePT1
    mnlPlainText2 : PDPrePT2
  mnl_module
    mnlModule1 : MPSPrinterNode
end

PDPrePT1 in PlainTextNode, Token with
mnl_text
  mnlText : "The data to be printed is being displayed."
end

Post Condition for Print Data

PDPost1NLS in
MatraNaturalLanguageStructure, Token with
  mnl_composite
    mnlCS1
      mnlPlainText1 : PDPostPT1;
      mnlPlainText2 : PDPostPT2
  mnl_module
    mnlModule1 : MPSPrinterNode
end

PDPostPT1 in PlainTextNode, Token with
mnl_text
  mnlText : "The data item has been printed by the"
end

Post Condition 1

CMAPOst1NLS in
MatraNaturalLanguageStructure, Token with
  mnl_composite
    mnlCS1
      mnlPlainText1 : CMAPOstPT1;
end

Post Condition 2

CMAPOst2NLS in
MatraNaturalLanguageStructure, Token with
  mnl_composite
    mnlCS1
      mnlPlainText1 : CMAPOstPT1;
      mnlPlainText2 : CMAPOstPT2;
      mnlPlainText3 : CMAPOstPT3
  mnl_module
    mnlModule1 : MissionPlanNode;
    mnlModule2 : MissionPlanningSystemNode
end

Choose Mission and Aircraft

ChooseMissionandAircraft in UseCase, Token with
use_case_name
  useCaseName : "Choose Mission and Aircraft"
sub_case
  subCase : False
pre_condition
  preCondition1 : CMAPOst1NLS
post_condition
  postCondition1 : CMAPOst1NLS;
    postCondition2 : CMAPOst2NLS
end

Pre Condition for Choose Mission and Aircraft

CMAPOst1NLS in
MatraNaturalLanguageStructure, Token with
  mnl_composite
    mnlCS1
      mnlPlainText1 : CMAPOstPT1;
end

Post Condition 1

CMAPOst1PT1 in PlainTextNode, Token with
mnl_text
  mnlText : "A database must be attached to the"
end

Post Condition 2

CMAPOst2PT1 in PlainTextNode, Token with
mnl_text
  mnlText : "A mission and aircraft have been selected."
end

Choose Mission and Aircraft
Retrieve from Mission Plan

RetrievefromMissionPlan in UseCase, Token with
use_case_name

useCaseName : "Retrieve from Mission Plan"
sub_case
subCase : True
pre_condition
preCondition1 : RfMPPre1NLS
post_condition
postCondition1 : RFMPPost1NLS
end

Pre Condition for Retrieve from Mission Plan

RfMPPre1NLS in MatraNaturalLanguageStructure, Token with
mnls_plain_text
mnlsPlainText1 : RfMPPrePT1;
end

RfMPPrePT1 in PlainTextNode, Token with
mnls_text
mnlsText : "A mission and aircraft have been defined."
end

Post Condition for Retrieve from Mission Plan

RFMPPost1NLS in MatraNaturalLanguageStructure, Token with
mnls_composite

mnlsComposite1 : RFMPPostC1;

mnlsModule1 : MissionPlanNode;

mnlsModule2 : MissionPlanningSystemNode
end

RFMPPostC1 in MatraNLSComposite, Token with
precedes Fragment

precedingFragment : RFMPPostPT1
subject_node
subjectNode : MissionPlanNode
following_fragment
followingFragment : RFMPPostC2
end

RFMPPostPT1 in PlainTextNode, Token with
mnls_text
mnlsText : "Data for the selected "
end

RFMPPostC2 in MatraNLSComposite, Token with
precedes Fragment

precedingFragment : RFMPPostPT2
subject_node
subjectNode : MissionPlanNode
following_fragment
followingFragment : RFMPPostC3
end

RFMPPostPT2 in PlainTextNode, Token with
mnls_text
mnlsText : "The data on the "
end

RFMPPostC3 in MatraNLSComposite, Token with
precedes Fragment

precedingFragment : RFMPPostPT3
subject_node
subjectNode : MissionPlanningSystemNode
following_fragment
followingFragment : RFMPPostC4
end

RFMPPostPT3 in PlainTextNode, Token with
mnls_text
mnlsText : " for the selected mission and aircraft has been copied onto the "
end

RFMPPostPT4 in PlainTextNode, Token with
mnls_text
mnlsText : "."
interactor_2
   interactor2 : RetrievefromCartridge
end
CartridgeRetrieveDatafromCartridge in Interaction, Token with interactor_1
   interactor1 : Cartridge
   interactor2 : RetrievefromCartridge
end
PilotSelectDatatoRetrieve in Interaction, Token with interactor_1
   interactor1 : Pilot
   interactor2 : SelectDatatoRetrieve
end
MPSPrinterPrintData in Interaction, Token with interactor_1
   interactor1 : MPSPrinter
   interactor2 : PrintData
end
MissionPlanRetrievefromMissionPlan in Interaction, Token with interactor_1
   interactor1 : MissionPlan
   interactor2 : RetrievefromMissionPlan
end
MissionPlanChooseMissionandAircraft in Interaction, Token with interactor_1
   interactor1 : MissionPlan
   interactor2 : ChooseMissionandAircraft
end
PrintDataDisplayCartridgeData in Extends, Token with extends_base
   extendsBase : PrintData
   extendsExtend : DisplayCartridgeData
end
SelectDatatoRetrieveRetrievefromCartridge in Extends, Token with extends_base
   extendsBase : SelectDatatoRetrieve
   extendsExtend : RetrievefromCartridge
end
EraseCartridgeUse Case Model
EraseCartridgeModel in UseCaseModel, Token with model_name
   modelName : "Erase Cartridge Model"
   ucLcomments
   ucmComments : EraseCartridgeComments
   ucm_use_case
   ucmUseCase1 : EraseCartridge;
   ucmUseCase2 : CheckforCartridge;
   ucmUseCase3 : CheckCartridgeforData;
   ucmUseCase4 : EraseDatafromCartridge
Retrieve from Cartridge Use Case Model (Partial - Derived Elements not Shown)

RetrievefromCartridgeModel in UseCaseModel, Token with model_name
modelName : "Retrieve from Cartridge Model"

ucm_actor
ucmActor1 : Pilot;
ucmActor2 : Cartridge
ucm_interaction
ucmInteraction1 : PilotRetrievefromCartridge;
ucmInteraction2 : CartridgeRetrieveDatafromCartridge;
ucmInteraction3 : PilotSelectDatatoRetrieve;
ucmInteraction4 : MPPrinterPrintData
ucm_includes
ucmIncludes1 : RetrievefromCartridgeCheckforCartridge;
ucmIncludes2 : RetrievefromCartridgeCheckCartridgeforData;
ucmIncludes3 : RetrievefromCartridgeEraseDatafromCartridge
ucm_pre_condition
ucmPreCondition1 : CCfDPre1NLS;
ucmPreCondition2 : EDfCPre1NLS;
ucmPreCondition3 : EDfCPre2NLS
ucm_post_condition
ucmPostCondition1 : ECPost1NLS;
ucmPostCondition2 : CFPost1NLS;
ucmPostCondition3 : CCfDPost1NLS;
ucmPostCondition4 : EDfCPost1NLS
end

EraseCartridgeComments in MatraNaturalLanguageStructure, Token with mnls_composite
mnlsComposite1 : ECCC1;
mnlsComposite2 : ECCC2;
mnlsComposite3 : ECCC3
mnls_plain_text
mnlsPlainText1 : ECCPT1;
mnlsPlainText2 : ECCPT2;
mnlsPlainText3 : ECCPT3;
mnlsPlainText4 : ECCPT4
mnls_module
mnlsModule1 : CartridgeNode
mnls_transaction
mnlsTransaction1 : CheckforCartridgeNode;
mnlsTransaction2 : CheckCartridgeforDataNode
end

ECCC1 in MatraNLSComposite, Token with preceding_fragment
precedingFragment : ECCPT1
subject_node
subjectNode : CartridgeNode
following_fragment
followingFragment : ECCC2
end

ECCPT1 in PlainTextNode, Token with mnls_text
mnlsText : "The purpose of this diagram is to model the erasing of data from a"
end

ECCC2 in MatraNLSComposite, Token with preceding_fragment
precedingFragment : ECCPT2
subject_node
subjectNode : CheckforCartridgeNode
following_fragment
followingFragment : ECCC3
end

ECCPT2 in PlainTextNode, Token with mnls_text
mnlsText : ", Checks are made to ensure that a cartridge is physically present, using ",
end
Appendix C (Part 1)

ucmPostCondition4 : FDPost1NLS;
ucmPostCondition5 : DCPost1NLS;
ucmPostCondition6 : SDLRPost1NLS
end

RetrievefromCartridgeComments in
MatraNaturalLanguageStructure, Token with

mnls_composite
mnlsComposite1 : RfCC1;
mnlsComposite2 : RfCC2

mnls_plain_text
mnlsPlainText1 : RfCPT1;
mnlsPlainText2 : RfCPT2;
mnlsPlainText3 : RfCPT3

mnls_module
mnlsModule1 : CartridgeNode;
mnlsModule2 : MissionPlanningSystemNode
end

RfCC1 in MatraNLSComposite, Token with
preceding_fragment
precedingFragment : RfCPT1
subject_node
subjectNode : CartridgeNode
following_fragment
followingFragment : RfCC2
end

RfCPT1 in PlainTextNode, Token with
mnls_text
mnlsText : "The purpose of this
diagram is to model retrieval of data
from the *
end

RfCPT2 in MatraNLSComposite, Token with
preceding_fragment
precedingFragment : RfCPT2
subject_node
subjectNode : MissionPlanningSystemNode
following_fragment
followingFragment : RfCPT3
end

RfCPT2 in PlainTextNode, Token with
mnls_text
mnlsText : " and its storage on the *
end

RfCPT3 in PlainTextNode, Token with
mnls_text
mnlsText : ".*
end

Choose Mission and Aircraft Use Case
Model (Partial - Derived Elements not
Shown)

ChooseMissionandAircraftModel in
UseCaseModel, Token with model_name
modelName : "Choose Mission and
Aircraft Model"

ucm_use_case
ucmUseCase1 : ChooseMissionandAircraft;

ucm_actor
ucmActor1 : Pilot;
ucmActor2 : MissionPlan

ucm_interaction
ucmInteraction1 : PilotChooseMissionandAircraft;

ucmInteraction2 : MissionPlanRetrievefromMissionPlan;

ucmPostCondition4 : RfMPPost1NLS
end

ucmIncludes1 :
ChooseMissionandAircraftRetrievefromMissionPlan
ucm_pre_condition
ucmPreCondition1 : CMAPre1NLS;
ucmPreCondition2 : RMPPre1NLS
ucmPostCondition
ucmPostCondition1 : CMAPost1NLS;
ucmPostCondition2 : CMAPost2NLS;
ucmPostCondition3 : RMPPost1NLS
end

* Service Instances

MissionAdministration in Service, Token
with

service_name : "Mission
Administration"

service_use_case
serviceUseCase1 : RetrievefromCartridge;

serviceUseCase2 : ChooseMissionandAircraft
end

CartridgeAdministration in Service, Token
with

service_name : "Cartridge
Administration"

service_use_case
serviceUseCase1 : EraseCartridge
end

* Hawk MPS Use Case View (Partial)

HawkMPSUseCaseView in UseCaseView, Token
with

ucv_service
ucvService1 : MissionAdministration;

ucvService2 : CartridgeAdministration

use_case_model
useCaseModel1 : EraseCartridgeModel;

useCaseModel2 : RetrievefromCartridgeModel;

useCaseModel3 : ChooseMissionandAircraftModel
end

* Hawk MPS User Centred Requirements
Structure

HawkMPSUCRS in
UserCentredRequirementsStructure, Token
with

subject_module
subjectModule : "Mission Planning
System"

use_case_view
useCaseView : HawkMPSUseCaseView

interaction_view
interactionView : HawkMPSInteractionView
end

C-11
Scenario Name: Erase Cartridge - Normal Path

2. C[IR] The [09 MPS] asks the [09 Cartridge] whether there [09 is a cartridge present in the hardware ] Request Confirm Cartridge.
3. C[IP] The [09 Cartridge] responds to the [09 MPS] that [09 there is a cartridge present in the hardware ] Confirm Cartridge.
4. C[IR] The [09 MPS] asks the [09 Cartridge] whether there [09 is data on the cartridge ] Request Confirm Cartridge Data.
5. C[IP] The [09 Cartridge] responds to the [09 MPS] that [09 there is data on the cartridge ] Confirm Cartridge Data.
6. C[IP] The [09 Cartridge] informs the [09 Pilot] that [09 there is data on the cartridge ] Cartridge Data Found Notification.
7. C[SR] The [09 Pilot] indicates to the [09 MPS] that it is [09 OK to proceed with Cartridge Erase : Proceed Erase Cartridge].
8. C[SR] The [09 MPS] tells the [09 Cartridge] to [09 erase the current data : Erase All].
9. C[SP] The [09 Cartridge] indicates to the [09 MPS] that [09 all current data has been deleted : Cartridge Erased].
10. C[SP] The [09 MPS] indicates to the [09 Pilot] that [09 all current data has been deleted : Cartridge Erased Notification].
12. C[IP] The [09 MPS] displays to the [09 Pilot] the [09 Main Screen Main Screen 1].

'MSC: Erase Cartridge - Normal Path'
Scenario Name: Erase Cartridge - No Cartridge Present


2. **C[IR]** The [C[IR] MPS] asks the [C[IR] Cartridge] whether there [C[IR] is a cartridge present in the hardware? Request Confirm Cartridge].

3. **C[IP]** The [C[IP] Cartridge] responds to the [C[IP] MPS] that [C[IP] there is no cartridge present in the hardware: No Data].

4. **C[IP]** The [C[IP] MPS] informs the [C[IP] Pilot] that there is [C[IP] no cartridge present in the hardware: No Cartridge Notification].


---

**MSC: Erase Cartridge - No Cartridge**
Scenario Name: Erase Cartridge - No Data on Cartridge

1. **C[SR]** The [Pilot] selects the [Erase Cartridge option on the MPS Main Screen.

2. **C[IR]** The [MPS] asks the [Cartridge] whether there is a cartridge present in the hardware?

3. **C[IP]** The [Cartridge] responds to the [MPS] that there is a cartridge present in the hardware.

4. **C[IR]** The [MPS] asks the [Cartridge] whether there is data on the cartridge?

5. **C[IP]** The [Cartridge] responds to the [MPS] that there is no data on the cartridge.

6. **C[IP]** The [MPS] informs the [Pilot] that there is no data on the cartridge.

7. **C[IP]** The [Pilot] sends an acknowledgment to the [MPS].

8. **C[IP]** The [MPS] displays to the [Pilot] the Main Screen.
Scenario Name: Erase Cartridge - Pilot Chooses Not to Erase Data

2. C[IR] The [CIP] MPS] asks the [Cartridge] whether there is a cartridge present in the hardware: Request Confirm Cartridge.
3. C[IP] The [CIP] Cartridge] responds to the [MPS] that there is a cartridge present in the hardware: Confirm Cartridge.
5. C[IP] The [CIP] Cartridge] responds to the [MPS] that there is data on the cartridge: Confirm Cartridge Data.

"MSC: Erase Cartridge - Pilot Chooses Not To Erase Data"
Retrieve From Cartridge - Normal Path (Fragment)

11. **T** The \( \text{MPS} \) sets the \( \text{DataTimer} \) to 10 seconds.

12. **C[SP]** The \( \text{MPS} \) informs the \( \text{Pilot} \) that \( \text{Type}_1 \) data has been retrieved: \( \text{Type}_1 \) Retrieved Notification.

13. **T** The \( \text{MPS} \) \( \text{DataTimer} \) times-out.

14. **A** The \( \text{MPS} \) \( \text{Type}_1 \) stores \( \text{Type}_1 \) data.

---

Choose Mission and Aircraft: New Mission From Open Missions (Fragment)

**ITERATION:** Lower Bound = 1 : Upper Bound = card data types \( \text{MissionPlan}() \)

11. **C[IR]** The \( \text{MPS} \) asks the \( \text{Mission Plan} \) to \( \text{Supply a data item: Request Data Item} \).

12. **C[IP]** The \( \text{Mission Plan} \) supplies the \( \text{data item for the selected Mission and Aircraft: Mission\&AC Data item} \) to the \( \text{MPS} \).

13. **A** The \( \text{MPS} \) \( \text{stores the data item for the selected Mission and Aircraft: Store Selected Mission\&AC Data} \).

---

"Retrieve from Cartridge - Normal Path (Timing Fragment)"

"Choose Mission and Aircraft - New Mission From Open Missions (Event Group Fragment)"
I. Instantiation of Interaction View for Erase Cartridge Scenarios

- Instantiation of Interaction View

HawkMPSInteractionView in InteractionView, Token with
interaction_model
interactionModel1 : CheckforCartridgeModel;
interactionModel2 : CheckCartridgeForDataModel;
interactionModel3 : EraseDatafromCartridgeModel;
interactionModel4 : EraseCartridgeModel
end

- Instantiation of Instance Definitions

MPSInstance in Instance, Token with
instance_name
instanceName : "MPS"
end

CartridgeInstance in Instance, Token with
instance_name
instanceName : "Cartridge"
end

PilotInstance in Instance, Token with
instance_name
instanceName : "Pilot"
end

MissionPlanInstance in Instance, Token with
instance_name
instanceName : "Mission Plan"
end

- Instantiation of Interaction View Elements

Instantiation of Check For Cartridge Interaction Model

CheckforCartridgeModel in InteractionModel, Token with
model_name
modelName : "Check For Cartridge Descriptions"
describes_use_case
describesUseCase : "Check for Cartridge"
imm_scenario
immScenario1 : CheckforCartridge_NormalPath;
immScenario2 : CheckforCartridge_NoCartridge
describesUseCase : "Check for Cartridge - Normal Path"
end

Instantiation of Check for Cartridge : Normal Path

CheckforCartridge_NormalPath in Scenario, Token with
scenario_title
scenarioTitle : "Check for Cartridge - Normal Path"
is_exception
scenario_event
scenarioEvent1 :
CheckforCartridgeRequest;
scenarioEvent2 :
ConfirmCartridgePresent
scn_seq_no
scnSeqNo1 :
CheckforCartridgeRequest_NormalPathSSN;
scnSeqNo2 :
ConfirmCartridgePresentSSN
tsn_viewpoint
tsnViewpoint :
CheckforCartridge_NormalPathTSV
msc_viewpoint
mscViewpoint :
CheckforCartridge_NormalPathMSV
end

CheckforCartridgeRequest_NormalPathSSN in SequenceNumber, Token with
sequence_no
sequenceNo : 1
end

CheckforCartridgeRequest_NormalPathSSN in SequenceNumber, Token with
sequence_no
sequenceNo : 2
end

CheckforCartridge_NormalPathTSV in TsnScenarioViewpoint, Token with
tsv_tsn_comm
tsvTsnComm1 :
CheckforCartridgeRequestTextDescription;
tsvTsnComm2 :
ConfirmCartridgePresentTextDescription
end

CheckforCartridge_NormalPathMSV in MscScenarioViewpoint, Token with
cartridge_present
msv_msc_comm
msvMscComm1 :
CheckforCartridgeRequestMSCDescription;
msvMscComm2 :
ConfirmCartridgePresentMSCDescription
end

Check for Cartridge Request Event Instantiation

CheckforCartridgeRequest in CommunicationEvent, Token with
interaction_type
interactionType : "IR"
sequence_no
sequenceNo1 :
CheckforCartridgeRequest_NormalPathSSN;
sequenceNo2 :
CheckforCartridgeRequest_NoCartridgeSSN
included_seq_no
includedSeqNo1 :
CheckforCartridgeRequest_EC_NP_ISN;
includedSeqNo2 :
CheckforCartridgeRequest_EC_NC_ISN;
includedSeqNo3 :
CheckforCartridgeRequest_EC_EC_ISN;
includedSeqNo4 :
CheckforCartridgeRequest_EC_PCMITED_ISN
tsn_communication_event
tsnCommunicationEvent :
CheckforCartridgeRequestTextDescription
msc_communication_event
mscCommunicationEvent :
CheckforCartridgeRequestMSCDescription
end
Appendix C (Part 1)

CheckforCartridgeRequestTextDescription in TsnCommunication, Token with 
communication_description
communicationDescription:
CheckforCartridgeRequestEventText end

CheckforCartridgeRequestEventText in 
ScenarioEventNaturalLanguageStructure, Token with

mnls_composite
mnlsComposite1 : CfCREC1;

mnlsComposite2 : CfCREC2;

mnlsComposite3 : CfCREC3

mnls_plain_text
mnlsPlainText1 : CfCREPT1;

mnlsPlainText2 : CfCREPT2;

mnlsPlainText3 : CfCREPT3;

mnlsPlainText4 : CfCREPT4
tsn_sender_node

tnsSenderNode : MPSInstance
tsn_receiver_node
tsnReceiverNode : CartridgeInstance
tsn_message_node

tnsMessageNode : RequestConfirmCartridge end

CfCREC1 in MatraNLSComposite, Token with
preceeding_fragment
preceedingFragment : CfCREPT1
subject_node

subjectNode : MPSInstance
following_fragment

followingFragment : CfCREC2
end

CfCREC2 in MatraNLSComposite, Token with
preceeding_fragment
preceedingFragment : CfCREPT2
subject_node

subjectNode : CartridgeInstance
following_fragment

followingFragment : CfCREC3
end

CfCREC3 in MatraNLSComposite, Token with
preceeding_fragment
preceedingFragment : CfCREPT3
subject_node

subjectNode : RequestConfirmCartridge
following_fragment

followingFragment : CfCREPT4
end

CfCREPT1 in PlainTextNode, Token with

mlns_text

mlnsText : "The "
end

CfCREPT2 in PlainTextNode, Token with

mlns_text

mlnsText : " asks the "
end

CfCREPT3 in PlainTextNode, Token with

mlns_text

mlnsText : " whether there "
end

CfCREPT4 in PlainTextNode, Token with

mlns_text

mlnsText : " . "
end

RequestConfirmCartridge in Message, Token with

message_name

messageName : "Request Confirm Cartridge"
tsn_msg_parameter

tnsMsgParameter:
RequestConfirmCartridgeTsnParameter
msg_msg_parameter

msgMsgParameter:
RequestConfirmCartridgeMscParameter end

RequestConfirmCartridgeTsnParameter in 
MessageDescription, Token with

msg_parameter

msgParameter : "is a cartridge present in the hardware?"
end

RequestConfirmCartridgeMscParameter in 
MessageDescription, Token with

msg_parameter

msgParameter : "Cartridge in Hardware?"
end

CheckforCartridgeRequestMSCDescription in 
MscCommunication, Token with

link_name

linkName : "unspecified"
synchronisation

Synchronisation : "slim"
frequency

_Frequency : "aperiodic"
delayed

_Delayed : False
msc_sender_instance

mscSenderInstance : MPSInstance
msc_receiver_instance

mscReceiverInstance : CartridgeInstance
msc_message

mscMessage : RequestConfirmCartridge end

Confirm Cartridge Present Event Instantiation

ConfirmCartridgePresent in 
CommunicationEvent, Token with

interaction_type

interactionType : "IP"
sequence_no

sequenceNo1 : 
ConfirmCartridgePresentSSN
included_seq_no

includedSeqNo1 :
ConfirmCartridgePresent_EC_NPC_ISN;
includedSeqNo2 :
ConfirmCartridgePresent_EC_NPC_ISN;
includedSeqNo3 :
ConfirmCartridgePresent_EC_PCNTED_ISN
follows_from

followsFrom : 
CheckforCartridgeRequest 
tsn_communication_event

tsnCommunicationEvent :
ConfirmCartridgePresentTextDescription
msc_communication_event

mscCommunicationEvent :
ConfirmCartridgePresentMscDescription end

ConfirmCartridgePresentTextDescription in 
TsnCommunication, Token with

communication_description
communicationDescription:
ConfirmCartridgePresentEventText end

C-18
ConfirmCartridgePresentEventText in ScenarioEventNaturalLanguageStructure, Token with

mnls_composite
mnlsComposite1 : CCPEC1;

mnlsComposite2 : CCPEC2;

mnlsComposite3 : CCPEC3

mnls_plain_text
mnlsPlainText1 : CCPEPT1;

mnlsPlainText2 : CCPEPT2;

mnlsPlainText3 : CCPEPT3;

mnlsPlainText4 : CCPEPT4

tsn_sender_node
tsnSenderNode : CartridgeInstance

tsn_receiver_node
tsnReceiverNode : MPSInstance

tsn_message_node
tsnMessageNode : ConfirmCartridge
end

CCPEC1 in MatraNLSComposite, Token with
preceding_fragment
precedingFragment : CCPEPT1

subject_node
subjectNode : CartridgeInstance

following_fragment
followingFragment : CCPEC2
end

CCPEC2 in MatraNLSComposite, Token with
preceding_fragment
precedingFragment : CCPEPT2

subject_node
subjectNode : MPSInstance

following_fragment
followingFragment : CCPEC3
end

CCPEC3 in MatraNLSComposite, Token with
preceding_fragment
precedingFragment : CCPEPT3

subject_node
subjectNode : ConfirmCartridge

following_fragment
followingFragment : CCPEC4
end

CCPEPT1 in PlainTextNode, Token with

mnls_text
mnlsText : "The"

end

CCPEPT2 in PlainTextNode, Token with

mnls_text
mnlsText : "responds to the"

end

CCPEPT3 in PlainTextNode, Token with

mnls_text
mnlsText : "that"

end

CCPEPT4 in PlainTextNode, Token with

mnls_text
mnlsText : "."

end

ConfirmCartridge in Message, Token with

message_name
messageName : "Confirm Cartridge"

tsn_msg_parameter
tsnMsgParameter : ConfirmCartridgeTsnParameter

msg_parameter
msgParameter : "there is a cartridge present in the hardware"
end

ConfirmCartridgeMscParameter in MessageDescription, Token with

msg_parameter
msgParameter : "Confirm Cartridge in Hardware"
end

ConfirmCartridgePresentMSCDescription in MscCommunication, Token with

link_name
linkName : "unspecified"
synchronisation
frequency
_frequency : "aperiodic"
delayed
_delayed : False

msc_sender_instance
mscSenderInstance : CartridgeInstance

msc_receiver_instance
mscReceiverInstance : MPSInstance

msc_message
mscMessage : ConfirmCartridge
end

Instantiation of Check for Cartridge: No Cartridge (Loaded in the Hardware)

CheckforCartridge_NoCartridge in Scenario, Token with
scenario_title
scenarioTitle : "Check for Cartridge - No Cartridge Present"
is_exception
isException : True

scenario_event
scenarioEvent1 : CheckforCartridgeRequest;
scenarioEvent2 : NoCartridgePresent

scn_seq_no
scnSeqNo1 :
CheckforCartridgeRequest_NoCartridgeSSN;

scnSeqNo2 : NoCartridgePresentSSN
tsn_viewpoint
tsnViewpoint :
CheckforCartridge_NoCartridgeTSV
msc_viewpoint
mscViewpoint :
CheckforCartridge_NoCartridgeMSV
end

CheckforCartridgeRequest_NoCartridgeSSN in SequenceNumber, Token with
sequence_no
sequenceNo : 1
end

NoCartridgePresentSSN in SequenceNumber, Token with
sequence_no
sequenceNo : 2
end

CheckforCartridge_NoCartridgeTSV in TsnScenarioViewpoint, Token
with
tsv_tsn_comm
tsvnComm1 :
CheckforCartridgeRequestTextDescription

svnComm2 :
NoCartridgePresentTextDescription
end

C-19
Appendix C (Part 1)

CheckforCartridge_NoCartridgeMSV in MscScenarioViewpoint, Token with
msv_msc_comm
  msvMscComm1 : CheckforCartridgeRequestMSCDescription;
  msvMscComm2 : NoCartridgePresentMSCDescription
end

No Cartridge Present Event Instantiation

NoCartridgePresent in CommunicationEvent, Token with
  interaction_type:
    interactionType : "IP"
  sequence_no:
    sequenceNo1 : NoCartridgePresentSN
  included_seq_no:
    includedSeqNo1 : NoCartridgePresent_EC_NC_ISN
follows_from:
  CheckforCartridgeRequest
  tsn_communication_event:
    tsnCommunicationEvent : NoCartridgePresentTextDescription
  msc_communication_event:
    mscCommunicationEvent : NoCartridgePresentMSCDescription
end

NoCartridgePresentTextDescription in TsnCommunication, Token with
  communication_description:
    NoCartridgePresentEventText
end

NoCartridgePresentEventText in ScenarioEventNaturalLanguageStructure, Token with
  mnls_composite:
    mnlsComposite1 : NCPEC1;
    mnlsComposite2 : NCPEC2;
    mnlsComposite3 : NCPEC3
  mnls_plain_text:
    mnlsPlainText1 : NCPEPT1;
    mnlsPlainText2 : NCPEPT2;
    mnlsPlainText3 : NCPEPT3;
    mnlsPlainText4 : NCPEPT4
tsn_sender_node:
  tsnSenderNode : CartridgeInstance
tsn_receiver_node:
  tsnReceiverNode : MPSInstance
tsn_message_node:
  tsnMessageNode : NoCartridge
end

NCPEC1 in MatraNLSComposite, Token with
  preceding_fragment:
    subject_node:
      subjectNode : NoCartridge
    following_fragment:
      followingFragment : NCPEPT1
end

NCPEC2 in MatraNLSComposite, Token with
  preceding_fragment:
    subject_node:
      subjectNode : MPSInstance
    following_fragment:
      followingFragment : NCPEPT2
end

NCPEC3 in MatraNLSComposite, Token with
  preceding_fragment:
    subject_node:
      subjectNode : MPSInstance
    following_fragment:
      followingFragment : NCPEPT3
end

No Cartridge Present Event

Instantiation

NoCartridgePresent in CommunicationEvent, Token with
  interaction_type:
    interactionType : "IP"
  sequence_no:
    sequenceNo1 : NoCartridgePresentSN
follows_from:
  CheckforCartridgeRequest
  tsn_communication_event:
    tsnCommunicationEvent : NoCartridgePresentTextDescription
  msc_communication_event:
    mscCommunicationEvent : NoCartridgePresentMSCDescription
end

NoCartridgePresentTextDescription in TsnCommunication, Token with
  communication_description:
    NoCartridgePresentEventText
end

NoCartridgePresentEventText in ScenarioEventNaturalLanguageStructure, Token with
  mnls_composite:
    mnlsComposite1 : NCPEC1;
    mnlsComposite2 : NCPEC2;
    mnlsComposite3 : NCPEC3
  mnls_plain_text:
    mnlsPlainText1 : NCPEPT1;
    mnlsPlainText2 : NCPEPT2;
    mnlsPlainText3 : NCPEPT3;
    mnlsPlainText4 : NCPEPT4
tsn_sender_node:
  tsnSenderNode : CartridgeInstance
tsn_receiver_node:
  tsnReceiverNode : MPSInstance
tsn_message_node:
  tsnMessageNode : NoCartridge
end

NCPEC1 in MatraNLSComposite, Token with
  preceding_fragment:
    subject_node:
      subjectNode : NoCartridge
    following_fragment:
      followingFragment : NCPEPT1
end

NCPEC2 in MatraNLSComposite, Token with
  preceding_fragment:
    subject_node:
      subjectNode : MPSInstance
    following_fragment:
      followingFragment : NCPEPT2
end

NCPEC3 in MatraNLSComposite, Token with
  preceding_fragment:
    subject_node:
      subjectNode : MPSInstance
    following_fragment:
      followingFragment : NCPEPT3
end

No Cartridge Present Event

Instantiation
Appendix C (Part 1)

inscenario

inmscenario1:
  CheckCartridgeforData_NormalPath;
inscenario2:
  CheckCartridgeforData_NoData
-- other elements may be derived using
an implementation of the rules in in
App. A, Pt. 2 (xi).
end

Instantiation of Check Cartridge for Data: Normal Path

CheckCartridgeforData_NormalPath in Scenario, Token with
scenario_title
  scenarioTitle : "Check Cartridge for Data - Normal Path"
  is_exception : False
scenario_event
  scenarioEvent1 : CheckCartridgeforDataRequest;
  scenarioEvent2 : ConfirmCartridgeDataPresent
scn_seq_no
  scnSeqNo1 :
    CheckCartridgeforDataRequest_NormalPath SSN;
  scnSeqNo2 :
    ConfirmCartridgeDataPresentSSN

CheckCartridgeforData_NormalPath in SequenceNumber, Token with
sequence_no
  sequenceNo : 1
end

ConfirmCartridgeDataPresentSSN in SequenceNumber, Token with
sequence_no
  sequenceNo : 2
end

CheckCartridgeforData_NormalPath in TsnScenarioViewpoint, Token with
tsn_viewpoint
  tsnViewpoint :
    CheckCartridgeforData_NormalPathTSV
msc_viewpoint
  mscViewpoint :
    CheckCartridgeforData_NormalPathMSV
end

CheckCartridgeforDataRequest_NormalPath SSN in SequenceNumber, Token with
sequence_no
  sequenceNo : 1
end

CheckCartridgeforDataRequest_NormalPath in TsnScenarioViewpoint, Token with
tsn_viewpoint
  tsnViewpoint :
    CheckCartridgeforDataRequestTextDescription
msc_viewpoint
  mscViewpoint :
    CheckCartridgeforDataRequestMSCDescription
end

Check Cartridge for Data Request Event Instantiation

CheckCartridgeforDataRequest in CommunicationEvent, Token with
interaction_type
  interactionType : "IR"
sequence_no
  sequenceNo1 :
    CheckCartridgeforDataRequest_NormalPath SSN;
  sequenceNo2 :
    CheckCartridgeforDataRequest_NoDataSSN
included_seq-no
  includedSeqNo1 :
    CheckCartridgeforDataRequest_EC_NP_ISN;
  includedSeqNo2 :
    CheckCartridgeforDataRequest_EC_ND_ISN;
  includedSeqNo3 :
    CheckCartridgeforDataRequest_EC_PINTED_ISN
tsn_communication_event
  tsnCommunicationEvent :
    CheckCartridgeforDataRequestTextDescription
msc_communication_event
  mscCommunicationEvent :
    CheckCartridgeforDataRequestMSCDescription
end

CheckCartridgeforDataRequestTextDescription in TsnCommunication, Token with
communication_description
  communicationDescription :
    CheckCartridgeforDataRequestEventText
end

CheckCartridgeforDataRequestEventText in ScenarioEventNaturalLanguageStructure, Token with
mls_composite
  mlsComposite1 : CCfDREC1;
  mlsComposite2 : CCfDREC2;
  mlsComposite3 : CCfDREC3
mls_plain_text
  mlsPlainText1 : CCfDREPT1;
  mlsPlainText2 : CCfDREPT2;
  mlsPlainText3 : CCfDREPT3;
  mlsPlainText4 : CCfDREPT4
tsn_sender_node
  tsnSenderNode : MPSInstance
tsn_receiver_node
  tsnReceiverNode : CartridgeInstance
tsn_message_node
  tsnMessageNode : RequestConfirmCartridgeData
end

CCfDREC1 in MatraNLSComposite, Token with
preceding_fragment
  precedingFragment : CCfDREPT1
subject_node
  subjectNode : MPSInstance
following_fragment
  followingFragment : CCfDREC2
end

CCfDREC2 in MatraNLSComposite, Token with
preceding_fragment
  precedingFragment : CCfDREPT2
subject_node
  subjectNode : CartridgeInstance
following_fragment
  followingFragment : CCfDREC3
end

CCfDREC3 in MatraNLSComposite, Token with
preceding_fragment
  precedingFragment : CCfDREPT3
subject_node

C-21
subjectNode : RequestConfirmCartridgeData
followingFragment : CCFDREPT4
end

CCFDRP01 in PlainTextNode, Token with
mnls_text
mnlsText : "The "
end

CCFDRP02 in PlainTextNode, Token with
mnls_text
mnlsText : " asks the "
end

CCFDRP03 in PlainTextNode, Token with
mnls_text
mnlsText : " whether there "
end

CCFDRP04 in PlainTextNode, Token with
mnls_text
mnlsText : ".
end

RequestConfirmCartridgeData in Message,
Token with
message_name
messageName : "Request Confirm Cartridge Data."

tns_msg_parameter
RequestConfirmCartridgeDataTsnParameter

msc_msg_parameter
RequestConfirmCartridgeDataMscParameter

RequestConfirmCartridgeDataTsnParameter in MessageDescription,
Token with
msg_parameter
msgParameter : "is data on the cartridge?"
end

RequestConfirmCartridgeDataMscParameter in MessageDescription,
Token with
msg_parameter
msgParameter : "Data on Cartridge?"
end

CheckCartridgeforDataRequestMSCDescription in MscCommunication,
Token with
link_name
linkName : "unspecified"

synchronisation
_Synchronisation_ : "sim"
frequency
_Frequency_ : "aperiodic"
delayed
_Delayed_ : False

msc_sender_instance
mscSenderInstance : MPSInstance

msc_receiver_instance
mscReceiverInstance : CartridgeInstance

msc_message
mscMessage : RequestConfirmCartridgeData
end

Confirm Cartridge Data Present Event Instantiation

ConfirmCartridgeDataPresent in CommunicationEvent, Token with
interaction_type
interactionType : "IP"

sequence_no
sequenceNo1 :
ConfirmCartridgeDataPresentSSN

included_seq_no
includedSeqNo1 :
ConfirmCartridgeDataPresent_EC_NP_ISN;

follows_from
followsFrom : CheckCartridgeforDataRequest

tns_communication_event
tsnCommunicationEvent :

msc_communication_event
mscCommunicationEvent :

ConfirmCartridgeDataPresentMSCDescription
end

ConfirmCartridgeDataPresentMSCDescription in TsnCommunication,
Token with
communication_description
communicationDescription :

ConfirmCartridgeDataPresentEventText
end

ConfirmCartridgeDataPresentEventText in ScenarioEventNaturalLanguageStructure,
Token with

mnls_composite

mnlsComposite1 : CCDPEC1;

mnlsComposite2 : CCDPEC2;

mnlsComposite3 : CCDPEC3

mnls_plain_text

mnlsPlainText1 : CCDPXT1;

mnlsPlainText2 : CCPXT2;

mnlsPlainText3 : CCPXT3;

mnlsPlainText4 : CPPXT4

tsn_sender_node
tsnSenderNode : CartridgeInstance

tsn_receiver_node
tsnReceiverNode : MPSInstance

tsn_message_node
tsnMessageNode :

ConfirmCartridgeData
end

CCDPEC1 in MatraNLSComposite, Token with
preceding_fragment
precedingFragment : CCDPXT1

subject_node
subjectNode : CartridgeInstance

following_fragment
followingFragment : CCDPEC2
end

CCDPEC2 in MatraNLSComposite, Token with
preceding_fragment
precedingFragment : CCDPXT2

subject_node
subjectNode : MPSInstance

following_fragment
followingFragment : CCDPEC3
end

CCDPEC3 in MatraNLSComposite, Token with
preceding_fragment
precedingFragment : CCDPXT3

subject_node
subjectNode : ConfirmCartridgeData

following_fragment
followingFragment : CCDPXT4
end

CCDPXT1 in PlainTextNode, Token with
mnls_text
The end CCDPEPT2 in P1ainTextNode, Token with 
mnlsText " responds to the "
end

The end CCDPEPT3 in P1ainTextNode, Token with 
mnlsText " that "
end

The end CCDPEPT4 in P1ainTextNode, Token with 
mnlsText " .* 
end

ConfirmCartridgeData in Message, Token with 
message_name 
messageName : "Confirm Cartridge Data"
tsn_msg_parameter 
tsnMsgParameter : ConfirmCartridgeDataTsnParameter 
msc_msg_parameter 
mscMsgParameter : ConfirmCartridgeDataMscParameter
end

ConfirmCartridgeDataTsnParameter in MessageDescription, Token with 
msg_parameter 
msgParameter : "there is data on the cartridge"
end

ConfirmCartridgeDataMscParameter in MessageDescription, Token with 
msg_parameter 
msgParameter : "Confirm Data on Cartridge"
end

ConfirmCartridgeDataPresentMscDescription in MscCommunication, Token with 
link_name 
linkName : "unspecified"
synchronisation 
_Synchronisation : = "sim"
frequency 
_Frequency : "aperiodic"
delayed 
_Delayed : False 
msc_sender_instance 
mscSenderInstance : CartridgeInstance 
msc_receiver_instance 
mscReceiverInstance : MPSInstance 
msc_message 
mscMessage : ConfirmCartridgeData
end

Instantiation of Check Cartridge for Data : No Data Present (on Cartridge)

CheckCartridgeforDataService_NoData in Scenario, Token with 
scenario_title : "Check Cartridge for Data - No Data Present"
is_exception 
isException : True 
scenario_event 
scenarioEvent1 : CheckCartridgeforDataServiceRequest; 
scenarioEvent2 : NoCartridgeDataPresent 
scn_seq_no 
scnSegNo1 : CheckCartridgeforDataServiceRequest_NoCartridgeDataSSN; 
scnSeqNo2 : NoCartridgeDataPresentSSN 
tsn_viewpoint 
tsnViewpoint : CheckCartridgeforDataService_NoCartridgeDataTSV 
msc_viewpoint 
mscViewpoint : CheckCartridgeforDataService_NoCartridgeDataMSV
end

CheckCartridgeforDataServiceRequest_NoCartridgeDataSSN in SequenceNumber, Token with 
sequence_no 
sequenceNo : 1 
end

NoCartridgeDataPresentSSN in SequenceNumber, Token with 
sequence_no 
sequenceNo : 2 
end

CheckCartridgeforDataService_NoCartridgeDataTSV in TsnScenarioViewpoint, Token with 
tsv_tsn_comm 
tsvTsnComm1 : CheckCartridgeforDataServiceRequestTextDescription; 
tsvTsnComm2 : NoCartridgeDataPresentTextDescription 
end

CheckCartridgeforDataService_NoCartridgeDataMSV in MscScenarioViewpoint, Token with 
msv_msc_comm 
msvMscComm1 : CheckCartridgeforDataServiceRequestMscDescription; 
msvMscComm2 : NoCartridgeDataPresentMscDescription 
end

No Cartridge Data Present Event Instantiation

NoCartridgeDataPresent in CommunicationEvent, Token with 
interaction_type 
interactionType : "IP" 
sequence_no 
sequenceNo1 : NoCartridgeDataPresentSSN 
included_seq_no 
includedSeqNo1 : NoCartridgeDataPresent_EC_ND_ISN 
follows_from 
followsFrom : CheckCartridgeforDataServiceRequest 
tsn_communication_event 
tsnCommunicationEvent : NoCartridgeDataPresentTextDescription 
msc_communication_event 
mscCommunicationEvent : NoCartridgeDataPresentMscDescription 
end

NoCartridgeDataPresentTextDescription in TsnCommunication, Token with 
communication_description 
communicationDescription : NoCartridgeDataPresentEventText 
end

C-23
Appendix C (Part 1)

NoCartridgeDataPresentEventText in ScenarioEventNaturalLanguageStructure, Token with

mlns_composite

mlnsComposite1 : NCDPEC1;
mlnsComposite2 : NCDPEC2;
mlnsComposite3 : NCDPEC3

mlns_plain_text

mlnsPlainText1 : NCDPEPT1;
mlnsPlainText2 : NCDPEPT2;
mlnsPlainText3 : NCDPEPT3;
mlnsPlainText4 : NCDPEPT4

tsn_sender_node
tsnSenderNode : CartridgeInstance
tsn_receiver_node
tsnReceiverNode : MPSInstance
tsn_message_node
tsnMessageNode : NoCartridgeData

end

NCDPEC1 in MatraNLSComposite, Token with

preceding_fragment

precedingFragment : NCDPEPT1

subject_node

subjectNode : CartridgeInstance

following_fragment

followingFragment : NCDPEC2

end

NCDPEC2 in MatraNLSComposite, Token with

preceding_fragment

precedingFragment : NCDPEPT2

subject_node

subjectNode : MPSInstance

following_fragment

followingFragment : NCDPEC3

end

NCDPEC3 in MatraNLSComposite, Token with

preceding_fragment

precedingFragment : NCDPEPT3

subject_node

subjectNode : NoCartridgeData

following_fragment

followingFragment : NCDPEPT4

end

NCDPEPT1 in PlainTextNode, Token with

mlns_text

mlnsText : "The "

end

NCDPEPT2 in PlainTextNode, Token with

mlns_text

mlnsText : " responds to the "

end

NCDPEPT3 in PlainTextNode, Token with

mlns_text

mlnsText : " that "

end

NCDPEPT4 in PlainTextNode, Token with

mlns_text

mlnsText : "." 

end

NoCartridgeData in Message, Token with

message_name

messageName : "No Cartridge Data"

tsn_msg_parameter

tsnMsgParameter : NoCartridgeDataTsnParameter

msc_msg_parameter

mscMsgParameter : NoCartridgeDataMscParameter

end

NoCartridgeDataTsnParameter in MessageDescription, Token with

msg_parameter

msgParameter : "there is no data on the cartridge"

end

NoCartridgeDataMscParameter in MessageDescription, Token with

msg_parameter

msgParameter : "No Data on Cartridge"

end

NoCartridgeDataPresentMscDescription in MscCommunication, Token with

link_name

linkName : "unspecified"

synchronisation

_Synchronisation : = "sim"

frequency

_Frequency : = "aperiodic"

delayed

_Delayed : False

msc_sender_instance

mscSenderInstance : CartridgeInstance

msc_receiver_instance

mscReceiverInstance : MPSInstance

msc_message

mscMessage : NoCartridgeData

end

Instantiation of Erase Data from Cartridge Interaction Model

EraseDataFromCartridgeModel in InteractionModel, Token with

model_name

modelName : "Erase Data from Cartridge Descriptions"

describes_use_case

describesUseCase : "Erase Data from Cartridge"

inm_scenario

inmScenario1

EraseDataFromCartridge

-- other elements may be derived using an implementation of the rules in in

App. A, Pt. 2 (xi).

end

Instantiation of Erase Data from Cartridge

EraseDataFromCartridge in Scenario, Token with

scenario_title

scenarioTitle : "Erase Data from Cartridge"

is_exception

isException : False

scenario_event

scenarioEvent1

EraseCurrentDataRequest;

scenarioEvent2

CurrentDataErasedNotification

scn_seq_no

scnSeqNo1

EraseCurrentDataRequestSSN;

scnSeqNo2

CurrentDataErasedNotificationSSN

tsn_viewpoint

tsnViewpoint

msc_viewpoint

mscViewpoint

EraseDataFromCartridgeTSV

EraseDataFromCartridgeMSV

end

C-24
Appendix C (Part 1)

EraseCurrentDataRequestSSN in SequenceNumber, Token with sequence_no
  sequenceNo : 1
end

CurrentDataErasedNotificationSSN in SequenceNumber, Token with sequence_no
  sequenceNo : 2
end

EraseDatafromCartridgeTSV in TsnScenarioViewpoint, Token with
tsv_tsn_comm
tsvTsnComm1 : EraseCurrentDataRequestTextDescription;
tsvTsnComm2 : CurrentDataErasedNotificationTextDescription
end

EraseDatafromCartridgeMSV in MscScenarioViewpoint, Token with
msv_msc_comm
  msvMscComm1 : EraseCurrentDataRequestMSCDescription;
  msvMscComm2 : CurrentDataErasedNotificationMSCDescription
end

EraseCurrentDataRequest in CommunicationEvent, Token with interaction_type
  interactionType : "SR"
sequence_no
  sequenceNo1 : EraseCurrentDataRequestSSN
  included_seq_no
  includedSeqNo1 : EraseCurrentDataRequest_EC_NP_ISN
  tsn_communication_event
    tsnCommunicationEvent : EraseCurrentDataRequestTextDescription
    msc_communication_event
      mscCommunicationEvent : EraseCurrentDataRequestMSCDescription
end

EraseCurrentDataRequest in CommunicationEvent, Token with interaction_type
  interactionType : "SR"
sequence_no
  sequenceNo1 : EraseCurrentDataRequestSSN
  included_seq_no
  includedSeqNo1 : EraseCurrentDataRequest_EC_NP_ISN
  tsn_communication_event
    tsnCommunicationEvent : EraseCurrentDataRequestTextDescription
    msc_communication_event
      mscCommunicationEvent : EraseCurrentDataRequestMSCDescription
end

EraseCurrentDataRequestEventText in ScenarioEventNaturalLanguageStructure, Token with
  mnls_composite
    mnlsComposite1 : ECDREC1;
    mnlsComposite2 : ECDREC2;
    mnlsComposite3 : ECDREC3
  mnls_plain_text
    mnlsPlainText1 : ECDREPT1;
    mnlsPlainText2 : ECDREPT2;
    mnlsPlainText3 : ECDREPT3;
    mnlsPlainText4 : ECDREPT4
  tsn_sender_node
    tsnSenderNode : MPSInstance
  tsn_receiver_node
    tsnReceiverNode : CartridgeInstance
  tsn_message_node
    tsnMessageNode : EraseAll
end

EraseCurrentDataRequest in CommunicationEvent, Token with interaction_type
  interactionType : "SR"
sequence_no
  sequenceNo1 : EraseCurrentDataRequestSSN
  included_seq_no
  includedSeqNo1 : EraseCurrentDataRequest_EC_NP_ISN
  tsn_communication_event
    tsnCommunicationEvent : EraseCurrentDataRequestTextDescription
    msc_communication_event
      mscCommunicationEvent : EraseCurrentDataRequestMSCDescription
end

EraseCurrentDataRequestEventText in ScenarioEventNaturalLanguageStructure, Token with
  mnls_composite
    mnlsComposite1 : ECDREC1;
    mnlsComposite2 : ECDREC2;
    mnlsComposite3 : ECDREC3
  mnls_plain_text
    mnlsPlainText1 : ECDREPT1;
    mnlsPlainText2 : ECDREPT2;
    mnlsPlainText3 : ECDREPT3;
    mnlsPlainText4 : ECDREPT4
  tsn_sender_node
    tsnSenderNode : MPSInstance
  tsn_receiver_node
    tsnReceiverNode : CartridgeInstance
  tsn_message_node
    tsnMessageNode : EraseAll
end

EraseCurrentDataRequestEventText in ScenarioEventNaturalLanguageStructure, Token with
  mnls_composite
    mnlsComposite1 : ECDREC1;
    mnlsComposite2 : ECDREC2;
    mnlsComposite3 : ECDREC3
  mnls_plain_text
    mnlsPlainText1 : ECDREPT1;
    mnlsPlainText2 : ECDREPT2;
    mnlsPlainText3 : ECDREPT3;
    mnlsPlainText4 : ECDREPT4
  tsn_sender_node
    tsnSenderNode : MPSInstance
  tsn_receiver_node
    tsnReceiverNode : CartridgeInstance
  tsn_message_node
    tsnMessageNode : EraseAll
end
Appendix C (Part 1)

frequency
  Frequency : "aperiodic"
delayed
    _Delayed : False
msc_sender_instance
    mscSenderInstance : MPSInstance
msc_receiver_instance
    mscReceiverInstance : CartridgeInstance
msc_message
    mscMessage : CartridgeErased
end

Current Data Erased Notification Event Instantiation

CurrentDataErasedNotification in CommunicationEvent, Token with
  interaction_type
    interactionType : "SP"
sequence_no
  sequenceNo1 : CurrentDataErasedNotificationSSN
includedSeqNo
    includedSegNo1 : CurrentDataErasedNotification_EC_NP_ISN
follows_from
  followsFrom : EraseCurrentDataRequest
tsn_communication_event
  tsnCommunicationEvent : CurrentDataErasedNotificationTextDescription
msc_communication_event
  mscCommunicationEvent : CurrentDataErasedNotificationMscDescription
end

CurrentDataErasedNotificationEventText in
  ScenarioEventNaturalLanguageStructure, Token with
msls_composite
  mnlscComposite1 : CDENEC1;
  mnlscComposite2 : CDENEC2;
  mnlscComposite3 : CDENEC3
mnl_class_text
  mnlClassText1 : CDENEPT1;
  mnlClassText2 : CDENEPT2;
  mnlClassText3 : CDENEPT3;
  mnlClassText4 : CDENEPT4
tsn_sender_node
  tsnSenderNode : CartridgeInstance
tsn_receiver_node
  tsnReceiverNode : MPSInstance
tsn_message_node
  tsnMessageNode : CartridgeErased
end

CDENEC1 in MatraNLSComposite, Token with
  preceding_fragment
    precedingFragment : CDENEC2
subject_node
  subjectNode : MPSInstance
following_fragment
  followingFragment : CDENEC3
end

CDENEC2 in MatraNLSComposite, Token with
  preceding_fragment
    precedingFragment : CDENEPT1
subject_node
  subjectNode : CartridgeInstance
following_fragment
  followingFragment : CDENEPT2
end

CDENEC3 in MatraNLSComposite, Token with
  preceding_fragment
    precedingFragment : CDENEPT3
subject_node
  subjectNode : CartridgeErased
following_fragment
  followingFragment : CDENEPT4
end

CDENEPT1 in PlainTextNode, Token with
  mnl_class_text
    mnlClassText : "The ".
end

CDENEPT2 in PlainTextNode, Token with
  mnl_class_text
    mnlClassText : " "
end

CDENEPT3 in PlainTextNode, Token with
  mnl_class_text
    mnlClassText : ":
end

CartridgeErased in Message, Token with
  message_name
    mName : "Cartridge Erased"
tsn_msg_parameter
  tsnMsgParameter : CartridgeErasedTsnParameter
msc_msg_parameter
  mscMsgParameter : CartridgeErasedMscParameter
end

CartridgeErasedTsnParameter in MessageDescription, Token with
  msg_parameter
    msgParameter : "all current data has been deleted"
end

CartridgeErasedMscParameter in MessageDescription, Token with
  msg_parameter
    msgParameter : "Cartridge Data Erased"
end

CurrentDataErasedNotificationMscDescription in MscCommunication, Token with
  link_name
    linkName : "unspecified"
synchronisation
  _Synchronisation : = "sim"
frequency
  _Frequency : "aperiodic"
delayed
    _Delayed : False
msc_sender_instance
  mscSenderInstance : CartridgeInstance
tsn_receiver_instance
  tsnReceiverInstance : MPSInstance
tsn_message
  tsnMessage : CartridgeErased
end

C-26
Instantiation of Erase Cartridge
Interaction Model

EraseCartridgeModel in InteractionModel, Token with model_name
modelName : "Erase Cartridge Descriptions"
describes_use_case
describesUseCase : "Erase Cartridge"
inm_scenario
inmScenario1 : EraseCartridge_NormalPath;
inmScenario2 : EraseCartridge_NoCartridge;
inmScenario3 : EraseCartridge_NoDataOnCartridge;
inmScenario4 : EraseCartridge_PilotChoosesNotToEraseDa
-- other elements may be derived using an implementation of the rules in in App. A, Pt. 2 (xi).
end

Instantiation of Erase Cartridge: Normal Path (Partial)
EraseCartridge_NormalPath in Scenario, Token with scenario_title
scenarioTitle : "Erase Cartridge - Normal Path"
is_exception
isException : False
scenario_event
scenarioEvent1 : EraseCartridgeRequest;
scenarioEvent2 : ConfirmDataPresentonCartridge;
scenarioEvent3 : ProceedCartridgeEraseRequest;
scenarioEvent4 : DataErasedNotification;
scenarioEvent5 : DataErasedAcknowledgement;
scenarioEvent6 : MainScreenDisplay_1
includes_scenario
includesScenario1 :
includesScenario2 :
includesScenario3 :
EraseDatafromCartridge
included_event
includedEvent1 : CheckforCartridgeRequest;
includedEvent2 : ConfirmCartridgePresent;
includedEvent3 : CheckCartridgeforDataRequest;
includedEvent4 : ConfirmCartridgeDataPresent;
includedEvent5 :
EraseCurrentDataRequest;
includedEvent6 :
CurrentDataErasedNotification
-- 'included_event' may be instantiated using a variation of the rule in App. A, Pt. 2 (vi. c);

scn_seq_no
scnSeqNo3 :
EraseCartridgeRequest_NormalPathSSN;
scnSeqNo4 :
DataErasedNotificationSSN;
scnSeqNo5 :
DataErasedAcknowledgementSSN;
scnSeqNo6 :
MainScreenDisplay_1_NormalPathSSN
csn_included_seq_no
scnIncludedSeqNo1 :
CheckforCartridgeRequest_EC_NP_ISN;
scnIncludedSeqNo2 :
ConfirmCartridgePresent_EC_NP_ISN;
scnIncludedSeqNo3 :
CheckCartridgeforDataRequest_EC_NP_ISN;
scnIncludedSeqNo4 :
ConfirmCartridgeDataPresent_EC_NP_ISN;
scnIncludedSeqNo5 :
EraseCurrentDataRequest_EC_NP_ISN;
scnIncludedSeqNo6 :
CurrentDataErasedNotification_EC_NP_ISN

ts_viewpoint
tsMapView
msc_viewpoint
mscMapView
EraseCartridge_NormalPathMSV
end

EraseCartridgeRequest_NormalPathSSN in SequenceNumber, Token with sequence_no
sequenceNo : 1
end
ConfirmDataPresentonCartridge_NormalPathSSN in SequenceNumber, Token with sequence_no
sequenceNo : 6
end
ProceedCartridgeEraseRequestSSN in SequenceNumber, Token with sequence_no
sequenceNo : 7
end
DataErasedNotificationSSN in SequenceNumber, Token with sequence_no
sequenceNo : 10
end
DataErasedAcknowledgementSSN in SequenceNumber, Token with sequence_no
sequenceNo : 11
end
MainScreenDisplay_1_NormalPathSSN in SequenceNumber, Token with sequence_no
sequenceNo : 12
end
CheckforCartridgeRequest_EC_NP_ISN in SequenceNumber, Token with sequence_no
sequenceNo : 2
end
ConfirmCartridgePresent_EC_NP_ISN in SequenceNumber, Token with sequence_no
sequenceNo : 3
end
CheckCartridgeforDataRequest_EC_NP_ISN in SequenceNumber, Token with sequence_no

sequenceNo : 4
end
ConfirmCartridgeDataPresent_EC_NP_ISN in SequenceNumber, Token with
sequence_no
sequenceNo : 5
end
EraseCurrentDataRequest_EC_NP_ISN in SequenceNumber, Token with
sequence_no
sequenceNo : 8
end
CurrentDataEraseNotification_EC_NP_ISN in SequenceNumber, Token with
sequence_no
sequenceNo : 9
end
EraseCartridge_NormalPathTSV in TsnScenarioViewpoint, Token with
tsv_tsn_comm
tsvTsnComm1 ;
EraseCartridgeRequestTextDescription;
tsvTsnComm2 :
ConfirmDataPresentonCartridgeTextDescription;
tsvTsnComm3 :
ProceedCartridgeEraseRequestTextDescription;
tsvTsnComm4 :
DataEraseNotificationTextDescription;
tsvTsnComm5 :
DataEraseAcknowledgementTextDescription;
tsvTsnComm6 :
MainScreenDisplay_1TextDescription
end
EraseCartridge_NormalPathMSV in MscScenarioViewpoint, Token with
mav_msc_comm
mavMscComm1 :
EraseCartridgeRequestMSCDescription;
mavMscComm2 :
ConfirmDataPresentonCartridgeMSCDescription;
mavMscComm3 :
ProceedCartridgeEraseRequestMSCDescription;
mavMscComm4 :
DataEraseNotificationMSCDescription;
mavMscComm5 :
DataEraseAcknowledgementMSCDescription;
mavMscComm6 :
MainScreenDisplay_1MSCDescription
end
Erase Cartridge Request Event Instantiation
EraseCartridgeRequest in CommunicationEvent, Token with
interaction_type
interactionType : "SR"
sequence_no
sequenceNo1 :
EraseCartridgeRequest_NormalPathSSN;
sequenceNo2 :
EraseCartridgeRequest_NoCartridgeSSN;
sequenceNo3 :
EraseCartridgeRequest_NoDataOnCartridge SSN;
sequenceNo4 :
EraseCartridgeRequest_PilotChoosesNotToEraseDataSSN
ts_nCommunication_event
tsnCommunicationEvent :
EraseCartridgeRequestTextDescription
mscCommunication_event
mscCommunicationEvent :
EraseCartridgeRequestMSCDescription
end
EraseCartridgeRequestTextDescription in TsnCommunication, Token with
communication_description
communicationDescription :
EraseCartridgeRequestEventText
end
EraseCartridgeRequestEventText in ScenarioEventNaturalLanguageStructure, Token with
mnlS_composite
mnlSComposite1 : ECREC1;
mnlSComposite2 : ECREC2;
mnlSComposite3 : ECREC3
mnlSPlain_text
mnlSPlainText1 : ECREPT1;
mnlSPlainText2 : ECREPT2;
mnlSPlainText3 : ECREPT3;
mnlSPlainText4 : ECREPT4
ts_nSender_node
tsnSenderNode : Pilotlnstance
ts_nReceiver_node
tsnReceiverNode : MPSlnstance
ts_nMessage_node
tsnMessageNode :
EraseCartridgeRequest
end
ECREC1 in MatraNLSComposite, Token with
preceding_fragment
precedingFragment : ECREPT1
subject_node
subjectNode : Pilotlnstance
followingFragment : ECREC2
end
ECREC2 in MatraNLSComposite, Token with
preceding_fragment
precedingFragment : ECREPT2
subject_node
subjectNode : EraseCartridgeRequest
followingFragment : ECREC3
end
ECREC3 in MatraNLSComposite, Token with
preceding_fragment
precedingFragment : ECREPT3
subject_node
subjectNode : MPSlnstance
followingFragment : ECREPT4
end
ECREPT1 in PlainTextNode, Token with
mnlS_text
mnlSText : "The 
end
ECREPT2 in PlainTextNode, Token with
mnlS_text
mnlSText : " selects the 
end
ECREPT3 in PlainTextNode, Token with
mnlS_text
mnlSText : " on the 
end
ECREPT4 in PlainTextNode, Token with
mnlS_text
mnlSText : " Main Screen."
EraseCartridgeRequest in Message, Token with
message_name
  messageName : "Erase Cartridge Request"
  msgParameter : EraseCartridgeRequestTsnParameter
  mscParameter : EraseCartridgeRequestMscParameter
end

EraseCartridgeRequestTsnParameter in MessageDescription, Token with
  msgParameter : "Erase Cartridge option"
end

EraseCartridgeRequestMscParameter in MessageDescription, Token with
  msgParameter : "Erase Cartridge"
end

EraseCartridgeRequestMSCDescription in MscCommunication, Token with
  linkName : "unspecified"
  synchronisation : "sim"
  frequency : "aperiodic"
  delayed : False
  msc_sender_instance : PilotInstance
  msc_receiver_instance : MPSInstance
  msc_message : EraseCartridgeRequest
end

Confirm Data Present on Cartridge Event Instantiation
ConfirmDataPresentonCartridge in CommunicationEvent, Token with
interaction_type
  interactionType : "IP"
  sequence_no
    sequenceNo1 : ConfirmDataPresentonCartridge_NormalPattern
    sequenceNo2 : ConfirmDataPresentonCartridge_PilotChooseNotToEraseDataSSN
  follows_from
    followsFrom : ConfirmCartridgeDataPresent
  tsn_communication_event : ConfirmDataPresentonCartridgeTextDescription
  msc_communication_event : ConfirmDataPresentonCartridgeMSCDescription
end

ConfirmDataPresentonCartridgeTextDescription in TsnCommunication, Token with
communication_description
  communicationDescription : ConfirmDataPresentonCartridgeEventText
end

ConfirmDataPresentonCartridgeEventText in

ScenarioEventNaturalLanguageStructure, Token with
  mls_composite
    mlsComposite1 : CDPoCEPT1
    mlsComposite2 : CDPoCEPT2
    mlsComposite3 : CDPoCEPT3
    mlsPlain_text
      mlsPlainText1 : CDPoCEPT1
      mlsPlainText2 : CDPoCEPT2
      mlsPlainText3 : CDPoCEPT3
      mlsPlainText4 : CDPoCEPT4
      tsn_sender_node
        tsnSenderNode : MPSInstance
      tsn_receiver_node
        tsnReceiverNode : PilotInstance
      tsn_message_node
        tsnMessageNode : CartridgeDataFoundNotification
end

CDPoCEPT1 in MatraNLSComposite, Token with
  preceding_fragment
    precedingFragment : CDPoCEPT1
  subject_node
    subjectNode : MPSInstance
    following_fragment
      followingFragment : CDPoCEPT2
end

CDPoCEPT2 in MatraNLSComposite, Token with
  preceding_fragment
    precedingFragment : CDPoCEPT2
  subject_node
    subjectNode : PilotInstance
    following_fragment
      followingFragment : CDPoCEPT3
end

CDPoCEPT3 in MatraNLSComposite, Token with
  preceding_fragment
    precedingFragment : CDPoCEPT3
  subject_node
    subjectNode : CartridgeDataFoundNotification
    following_fragment
      followingFragment : CDPoCEPT4
end

CDPoCEPT4 in MatraNLSComposite, Token with
  mls_text
    mlsText : "The "
end

CartridgeDataFoundNotification in Message, Token with
message_name
  messageName : "Cartridge Data Found Notification"
  tsn_msgParameter
    tsnMsgParameter : CartridgeDataFoundNotificationTsnParameter
end
Appendix C (Part 1)

mscMsgParameter : CartridgeDataFoundNotificationMscParameter
end

CartridgeDataFoundNotificationTsnParameter in MessageDescription, Token with
msg_parameter
msgParameter : "there is data on the cartridge"
end

CartridgeDataFoundNotificationMscParameter in MessageDescription, Token with
msg_parameter
msgParameter : "Confirm Data on Cartridge"
end

ConfirmDataPresentOnCartridgeMSCDescription in MscCommunication, Token with
link_name
linkName : "unspecified"
synchronisation
_synchrination : = "sim"
frequency
_frequency : "aperiodic"
delayed
_delayed : False
msc_sender_instance
mscSenderInstance : MPSInstance
msc_receiver_instance
mscReceiverInstance : PilotInstance
msc_message
mscMessage : CartridgeDataFoundNotification
end

Proceed Cartridge Erase Request Event Instantiation

ProceedCartridgeEraseRequest in CommunicationEvent, Token with
interaction_type
interaction_Type : "SR"
sequence_no
sequenceNo1 : ProceedCartridgeEraseRequestSSN
follows_from
followsFrom : ConfirmDataPresentOnCartridge
tsn_communication_event
tsnCommunicationEvent : ProceedCartridgeEraseRequestTextDescription
mcs_communication_event
mcsCommunicationEvent : ProceedCartridgeEraseRequestMscDescription
end

ProceedCartridgeEraseRequestTextDescription in TsnCommunication, Token with
communication_description
communication_Description : ProceedCartridgeEraseRequestEventText
end

ProceedCartridgeEraseRequestEventText in
ScenarioEventNaturalLanguageStructure, Token with
mnlS_composite
mnl8Composite1 : PCEREPT1;
mnl8Composite2 : PCEREPT2;
mnl8Composite3 : PCEREPT3;
mnlS_plain_text
mnl8PlainText1 : PCEREPT4;
mnl8PlainText2 : PCEREPT5;
mnl8PlainText3 : PCEREPT6;

mnlS_plain_text4 : PCEREPT7

tns_sender_node
tsnSenderNode : PilotInstance
tsn_receiver_node
tsnReceiverNode : MPSInstance
tsn_message_node
tsnMessageNode : ProceedEraseCartridge
end

PCEREPT1 in MatraNLSComposite, Token with
preceding_fragment
precedingFragment : PCEREPT1
subject_node
subjectNode : PilotInstance
following_fragment
followingFragment : PCEREPT2
end

PCEREPT2 in MatraNLSComposite, Token with
preceding_fragment
precedingFragment : PCEREPT2
subject_node
subjectNode : MPSInstance
following_fragment
followingFragment : PCEREPT3
end

PCEREPT3 in MatraNLSComposite, Token with
preceding_fragment
precedingFragment : PCEREPT3
subject_node
subjectNode : ProceedEraseCartridge
following_fragment
followingFragment : PCEREPT4
end

PCEREPT4 in MatraNLSComposite, Token with
preceding_fragment
precedingFragment : PCEREPT4
subject_node
subjectNode : ProceedEraseCartridge
following_fragment
followingFragment : PCEREPT5
end

ProceedEraseCartridge in Message, Token with
message_name
messageName : "Proceed Erase Cartridge"
tsn_msg_parameter
tsnMsgParameter :
msc_msg_parameter
mscMsgParameter :
ProceedEraseCartridgeTsnParameter
end

ProceedEraseCartridgeTsnParameter in
MessageDescription, Token with
msg_parameter
msgParameter : "OK to proceed with Cartridge Erase"
end
ProceedEraseCartridgeMscParameter in MessageDescription, Token with msg parameter
  msgParameter : "OK: Proceed Erase"
end
ProceedCartridgeEraseRequestMscParameter in MscCommunication, Token with
link_name
  linkName : "unspecified"
synchronisation
    _Synchronisation : = "sim"
frequency
      _Frequency : = "aperiodic"
delayed
        _Delayed : False
msc_sender_instance
  mscSenderInstance : PilotInstance
msc_receiver_instance
  mscReceiverInstance : MPSInstance
msc_message
  mscMessage : ProceedEraseCartridge
end

Data Erased Notification Event Instantiation

DataErasedNotification in CommunicationEvent, Token with interaction_type
  interactionType : "SP"
sequence_no
  sequenceNo : DataErasedNotificationSSN
follows_from
  followsFrom : CurrentDataErasedNotification
tsn_communication_event
  tsnCommunicationEvent : DataErasedNotificationTextDescription
msc_communication_event
  mscCommunicationEvent : DataErasedNotificationMSCDescription
end
DataErasedNotificationTextDescription in TsnCommunication, Token with communication_description
  communicationDescription : DataErasedNotificationEventText
end
DataErasedNotificationEventText in ScenarioEventNaturalLanguageStructure, Token with
  mnlS_composite
    mnlSComposite1 : DENEC1;
    mnlSComposite2 : DENEC2;
    mnlSComposite3 : DENEC3
mnlS_plain_text
  mnlSPlainText1 : DENEPT1;
  mnlSPlainText2 : DENEPT2;
  mnlSPlainText3 : DENEPT3;
  mnlSPlainText4 : DENEPT4
tsn_sender_node
  tsnSenderNode : MPSInstance
tsn_receiver_node
  tsnReceiverNode : PilotInstance
tsn_message_node
  tsnMessageNode : CartridgeErasedNotification
end
DENEC1 in MatraNLSComposite, Token with preceding_fragment
  precedingFragment : DENEPT2
subject_node
  subjectNode : PilotInstance
following_fragment
  followingFragment : DENEC2
end
DENEC2 in MatraNLSComposite, Token with preceding_fragment
  precedingFragment : DENEPT2
subject_node
  subjectNode : PilotInstance
following_fragment
  followingFragment : DENEC3
end
DENEC3 in MatraNLSComposite, Token with preceding_fragment
  precedingFragment : DENEPT3
subject_node
  subjectNode : CartridgeErasedNotification
following_fragment
  followingFragment : DENEPT4
end
DENEPT1 in PlainTextNode, Token with
  mnlS_text
    mnlSText : "The 
end
DENEPT2 in PlainTextNode, Token with
  mnlS_text
    mnlSText : " indicates to the 
end
DENEPT3 in PlainTextNode, Token with
  mnlS_text
    mnlSText : " that 
end
DENEPT4 in PlainTextNode, Token with
  mnlS_text
    mnlSText : "," 
end

CartridgeErasedNotification in Message, Token with message_name
  messageName : "Cartridge Erased Notification"
tsn_msg_parameter
  tsnMsgParameter : CartridgeErasedNotificationTsnParameter
msc_msg_parameter
  mscMsgParameter : CartridgeErasedNotificationMscParameter
end

CartridgeErasedNotificationTsnParameter in MessageDescription, Token with msg parameter
  msgParameter : "all current data has been deleted"
end
CartridgeErasedNotificationMscParameter in MessageDescription, Token with msg parameter
  msgParameter : "Confirm All Data Deleted"
end

DataErasedNotificationMSCDescription in MscCommunication, Token with link_name
  linkName : "unspecified"
synchronisation
  _Synchronisation : = "sim"
frequency
    _Frequency : = "aperiodic"
delayed
      _Delayed : False
msc_sender_instance
  mscSenderInstance : MPSInstance
msc_receiver_instance
  mscReceiverInstance : PilotInstance
end
Data Erased Acknowledgement Event Instantiation

DataErasedAcknowledgement in CommunicationEvent, Token with interaction_type "IP"
sequence_no sequenceNo1:
DataErasedAcknowledgementSSN follows_from:
DataErasedNotification

tsn_communication_event:
DataErasedAcknowledgementEvent:
DataErasedAcknowledgementTextDescription

msc_communication_event:
DataErasedAcknowledgementMSCDescription

DataErasedAcknowledgementEventText in ScenarioEventNaturalLanguageStructure, Token with

mnls_composite

\[ \text{DEAEC1}, \text{DEAEC2}, \text{DEAEC3} \]

mnls_plain_text

\[ \text{DEAEPT1}, \text{DEAEPT2}, \text{DEAEPT3}, \text{DEAEPT4} \]

tsn_sender_node

\text{Pilotlnstance}

tsn_receiver_node

\text{MPSlnstance}

tsn_message_node

\text{AcknowledgeErase}

data

DEAEC1 in MatraNLSComposite, Token with
preceding_fragment
subject_node

\text{Pilotlnstance}

following_fragment

\text{DEAEC2}

end

DEAEC2 in MatraNLSComposite, Token with
preceding_fragment
subject_node

\text{AcknowledgeErase}

following_fragment

\text{DEAEC3}

end

DEAEC3 in MatraNLSComposite, Token with
preceding_fragment
subject_node

\text{MPSlnstance}

following_fragment

\text{DEAEPT4}

end

DEAEPT1 in PlainTextNode, Token with
mnls_text

\"The \end{quote}
end

DEAEPT2 in PlainTextNode, Token with
mnls_text

\" sends an \end{quote}
end

DEAEPT3 in PlainTextNode, Token with
mnls_text

\" to the \end{quote}
end

DEAEPT4 in PlainTextNode, Token with
mnls_text

\" \end{quote}
end

AcknowledgeErase in Message, Token with
message_name

\text{Acknowledge Erase}:

\[ \text{tsn_msg_parameter} \]

\text{AcknowledgmentTsnParameter}:

\[ \text{msc_msg_parameter} \]

\text{AcknowledgeEraseMscParameter}:

\[ \text{DEAEPT1} \]

\[ \text{DEAEPT2} \]

\[ \text{DEAEPT3} \]

\[ \text{DEAEPT4} \]

AcknowledgeEraseTsnParameter in
MessageDescription, Token with
msg_parameter

\text{acknowledgement}:

end

AcknowledgeEraseMscParameter in
MessageDescription, Token with
msg_parameter

\text{OK: Acknowledge Erase}:

end

DataErasedAcknowledgementMSCDescription

in MscCommunication, Token with

link_name

\text{unspecified}:

synchronisation

\text{Sim}:

frequency

\text{aperiodic}:

delayed

\text{False}:

msc_sender_instance

\text{Pilotlnstance}:

msc_receiver_instance

\text{MPSlnstance}:

msc_message

\text{AcknowledgeErase}:

end

Main Screen Display (from Erase Cartridge) Event Instantiation

MainScreenDisplay_1 in
CommunicationEvent, Token with
interaction_type "IP"
sequence_no sequenceNo1:
MainScreenDisplay_1_NormalPathSSN;
sequenceNo2:
MainScreenDisplay_1_NoCartridgeSSN;
sequenceNo3:
MainScreenDisplay_1_NoDataOnCartridgeSSN;
sequenceNo4:
MainScreenDisplay_1_PilotChoosesNotToEraseDataSSN

follows_from
followsFrom1: DataErasedAcknowledgement;
followsFrom2: NoCartridgeAcknowledgement;
followsFrom3: NoCartridgeDataPresentAcknowledgement;
followsFrom4: CancelEraseRequest
tsn_communication_event
tsnCommunicationEvent:
MainScreenDisplay_1TextDescription
mstCommunicationEvent:
MainScreenDisplay_lMSCDescription
end
MainScreenDisplay_1TextDescription in TsnCommunication, Token with communication_description
communicationDescription:
MainScreenDisplay_1EventText in ScenarioEventNaturalLanguageStructure, Token with mns_composite
mnsComposite1 : MSD_1EC1;
mnsComposite2 : MSD_1EC2;
mnsComposite3 : MSD_1EC3
msnPlain_text
msnPlainText1 : MSD_1EPT1;
msnPlainText2 : MSD_1EPT2;
msnPlainText3 : MSD_1EPT3;
msnPlainText4 : MSD_1EPT4
end
MSD_1EC1 in MatraNLSComposite, Token with preceding_fragment
precedingFragment : MSD_1EPT1
subject_node
subjectNode : MPSInstance
following_fragment
followingFragment : MSD_1EC2
end
MSD_1EC2 in MatraNLSComposite, Token with preceding_fragment
precedingFragment : MSD_1EPT2
subject_node
subjectNode : PilotInstance
following_fragment
followingFragment : MSD_1EC3
end
MSD_1EC3 in MatraNLSComposite, Token with preceding_fragment
precedingFragment : MSD_1EPT3
subject_node
subjectNode : MainScreen
following_fragment
followingFragment : MSD_1EPT4
end
MSD_1EPT1 in PlainTextNode, Token with mns_text
mnsText : "The "
end
MSD_1EPT2 in PlainTextNode, Token with mns_text
mnsText : " displays to the "
end
MSD_1EPT3 in PlainTextNode, Token with mns_text
mnsText : " the "
end
MSD_1EPT4 in PlainTextNode, Token with mns_text
mnsText : " ." end
MainScreen_1 in Message, Token with message_name
messageName : "Main Screen"
end
MainScreenDisplay_lMSCDescription in MscCommunication, Token with link_name
linkName : "unspecified" synchronization
_synchronisation : = "sim" frequency
_frequency : = "aperiodic" delayed
_delayed : = False msc_sender_instance
mscSenderInstance : MPSInstance msc_receiver_instance
mscReceiverInstance : PilotInstance msc_message
mscMessage : MainScreen_1
end
Instantiation of Erase Cartridge: No Cartridge Present
EraseCartridge_NoCartridge in Scenario, Token with
scenario_title
scenarioTitle : "Erase Cartridge - No Cartridge Present" is_exception
isException : True scenario_event
scenarioEvent1
scenarioEvent2
scenarioEvent3
scenarioEvent4
includes_scenario
includesScenario1
includesScenario2
includesScenario3
includesScenario4
scn_seq_no
scnSeqNo1
EraseCartridgeRequest_NoCartridgeSSN;
scnSeqNo2:
NoCartridgePresentNotificationSSN;
scnSeqNo3:
NoCartridgeAcknowledgementSSN;
scnSeqNo4:
MainScreenDisplay_1
scn_included_seq_no
scnIncludedSeqNo1
CheckforCartridge_NoCartridgeIncluded_event
includedEvent1:
CheckforCartridgeRequest;
IncludedEvent2:
NoCartridgePresent
-- again, 'included_event' may be instantiated using a variation of the
rule in App. A, Pt. 2 (vii. c)
scn_seq_no
scnSeqNo1:
EraseCartridgeRequest_NoCartridgeSSN;
scnSeqNo2:
NoCartridgePresentNotificationSSN;
scnSeqNo3:
NoCartridgeAcknowledgementSSN;
scnSeqNo4:
MainScreenDisplay_1_NoCartridgeSSN
scn_included_seq_no
scnIncludedSeqNo1:
CheckforCartridgeRequest_EC_NC_ISN;
scnIncludedSeqNo2:
NoCartridgePresent_EC_NC_ISN
end
Appendix C (Part 1)

EraseCartridge_NoCartridgeTSV
msc_viewpoint:
EraseCartridge_NoCartridgeMSV
end

EraseCartridgeRequest_NoCartridgeSSN in SequenceNumber, Token with sequence_no sequenceNo : 1
end

NoCartridgePresentNotificationSSN in SequenceNumber, Token with sequence_no sequenceNo : 4
end

NoCartridgeAcknowledgementSSN in SequenceNumber, Token with sequence_no sequenceNo : 5
end

MainScreenDisplay_1_NoCartridgeSSN in SequenceNumber, Token with sequence_no sequenceNo : 6
end

CheckForCartridgeRequest_EC_NC_ISN in SequenceNumber, Token with sequence_no sequenceNo : 2
end

NoCartridgePresent_EC_NC_ISN in SequenceNumber, Token with sequence_no sequenceNo : 3
end

EraseCartridge_NoCartridgeTSV in TanScenarioViewpoint, Token with tsv_tsn_comm tsvTsnComm1 : EraseCartridgeRequestTextDescription; tsvTsnComm2 : NoCartridgePresentNotificationTextDescription; tsvTsnComm3 : NoCartridgeAcknowledgementTextDescription; tsvTsnComm4 : MainScreenDisplay_1TextDescription
end

EraseCartridge_NoCartridgeMSV in MscScenarioViewpoint, Token with msv_msc_comm msvMscComm1 : EraseCartridgeRequestMSCDescription; msvMscComm2 : NoCartridgePresentNotificationMSCDescription; msvMscComm3 : NoCartridgeAcknowledgementMSCDescription; msvMscComm4 : MainScreenDisplay_1MSCDescription
end

NoCartridgePresentNotificationEvent Instantiation
NoCartridgePresentNotification in CommunicationEvent, Token with interaction_type

interactionType : "IP"
sequence_no sequenceNo1 :
NoCartridgePresentNotificationSSN follows_from
followsFrom : NoCartridgePresent tsn_communication_event tsnCommunicationEvent :
NoCartridgePresentNotificationTextDescription
msc_communication_event mscCommunicationEvent :
NoCartridgePresentNotificationMSCDescription
end

NoCartridgePresentNotificationTextDescription in TsnCommunication, Token with communication_description communicationDescription :
NoCartridgePresentNotificationEventText
end

end

NCPNEC1 in MatraNLSComposite, Token with preceding_fragment precedingFragment : NCPNEPT1 subject_node subjectNode : MPSInstance following_fragment followingFragment : NCPNEC2
end

NCPNEC2 in MatraNLSComposite, Token with preceding_fragment precedingFragment : NCPNEPT2 subject_node subjectNode : PilotInstance following_fragment followingFragment : NCPNEC3
end

NCPNEC3 in MatraNLSComposite, Token with preceding_fragment precedingFragment : NCPNEPT3 subject_node subjectNode : NoCartridgeNotification following_fragment followingFragment : NCPNEPT4
end

NCPNEPT1 in PlainTextNode, Token with mnls_text mnlsText : "The "
end
NCPNEPT2 in PlainTextNode, Token with

    mnlsText : " informs the 
end

NCPNEPT3 in PlainTextNode, Token with

    mnlsText : " that there is 
end

NCPNEPT4 in PlainTextNode, Token with

    mnlsText : "."
end

NoCartridgeNotification in Message,
Token with
message_name

    messageName : "No Cartridge Notification"
end
tsn_MsgParameter

    tsnMsgParameter : NoCartridgeNotificationTsnParameter
msc_MsgParameter

    mscMsgParameter : NoCartridgeNotificationMscParameter
end

NoCartridgeNotificationTsnParameter in
MessageDescription, Token with

    msgParameter : "no cartridge present in the hardware"
end

NoCartridgeNotificationMscParameter in
MessageDescription, Token with

    msgParameter : "Report No Cartridge"
end

NoCartridgePresentNotificationMSCDescription in
MscCommunication, Token with

    link_name

    linkName : "unspecified"
synchronisation

    _Synchronisation : = "sim"
frequency

    _Frequency : "aperiodic"
delayed

    _Delayed : False
msc_sender_instance

    mscSenderInstance : MPSInstance
msc_receiver_instance

    mscReceiverInstance : PilotInstance
msc_message

    mscMessage : NoCartridgeNotification
end

No Cartridge Acknowledgement Event
Instantiation

NoCartridgeAcknowledgement in
CommunicationEvent, Token with
interaction_type

    interactionType : "IP"
sequence_no

    sequenceNo1 : 1
NoCartridgeAcknowledgementSSN follows_from

    followsFrom : NoCartridgePresentNotification
    tsn_communication_event

    tsnCommunicationEvent : NoCartridgeAcknowledgementTextDescription
end

NoCartridgeAcknowledgementTextDescription in
mcsCommunication, Token with

    communication_description

    communicationDescription : NoCartridgeAcknowledgementEventText
end

NoCartridgeAcknowledgementEventText in
ScenarioEventNaturalLanguageStructure, Token with

    mls_composite

    mlsComposite1 : NCAEC1;
    mlsComposite2 : NCAEC2;
    mlsComposite3 : NCAEC3
mls_plain_text

    mlsPlainText1 : NCAEPT1;
    mlsPlainText2 : NCAEPT2;
    mlsPlainText3 : NCAEPT3;
    mlsPlainText4 : NCAEPT4
tsn_sender_node

    tsnSenderNode : PilotInstance
tsn_receiver_node

    tsnReceiverNode : MPSInstance
tsn_message_node

    tsnMessageNode : AcknowledgeNoCartridge
end

NCAEC1 in MatraNLSComposite, Token with

preceding_fragment : NCAEPT1
subject_node

    subjectNode : PilotInstance
following_fragment : NCAEPT2
end

NCAEC2 in MatraNLSComposite, Token with

preceding_fragment : NCAEPT2
subject_node

    subjectNode : PilotInstance
following_fragment : NCAEPT3
end

NCAEC3 in MatraNLSComposite, Token with

preceding_fragment : NCAEPT3
subject_node

    subjectNode : MPSInstance
following_fragment : NCAEPT4
end

NCAEPT1 in PlainTextNode, Token with

    mnlsText : The 
end

NCAEPT2 in PlainTextNode, Token with

    mnlsText : " sends an 
end

NCAEPT3 in PlainTextNode, Token with

    mnlsText : " to the 
end

NCAEPT4 in PlainTextNode, Token with

    mnlsText : "."
Appendix C (Part 1)

AcknowledgeNoCartridge in Message, Token with
message_name : "Acknowledge No Cartridge"
msg_parameter
  tsn_msg_parameter : AcknowledgeNoCartridgeTsnParameter
  msc_msg_parameter : AcknowledgeNoCartridgeMscParameter
end

AcknowledgeNoCartridgeTsnParameter in MessageDescription, Token with
msg_parameter
  magParameter : "acknowledgement"
end

AcknowledgeNoCartridgeMscParameter in MessageDescription, Token with
msg_parameter
  magParameter : "OK: Acknowledge No Cartridge"
end

NoCartridgeAcknowledgementMscDescription in MscCommunication, Token with
link_name
  linkName : "unspecified"
synchronization
    _Synchronization : = "sim"
frequency
    _Frequency : = "aperiodic"
delayed
    _Delayed : False
mac_sender_instance
  macSenderInstance : PilotInstance
mac_receiver_instance
  macReceiverInstance : MPSInstance
mac_message
  macMessage : AcknowledgeNoCartridge
end

Instantiation of Erase Cartridge: No Data on Cartridge

EraseCartridge_NoDataOnCartridge in Scenario, Token with
scenario_title
  scenarioTitle : "Erase Cartridge - No Data on Cartridge"
is_exception : True
scenario_event
  scenarioEvent1 : EraseCartridgeRequest;
  scenarioEvent2 : NoCartridgeDataPresentNotification;
  scenarioEvent3 : NoCartridgeDataPresentAcknowledgement;
  scenarioEvent4 : MainScreenDisplay_1_NoDataOnCartridge;
end

-- again 'included_event' may be instantiated using a variation of the .
rule in App.A, Pt.2 (vii.c)
scn_seq_no
  scnSeqNo1 : EraseCartridgeRequest_NoDataOnCartridgeSSN;
  scnSeqNo2 : NoCartridgeDataPresentNotificationSSN;
  scnSeqNo3 : NoCartridgeDataPresentAcknowledgementSSN;
  scnSeqNo4 : MainScreenDisplay_1_NoDataOnCartridgeSSN
end

scn_included_seq_no
  scnIncludedSeqNo1 : CheckforCartridgeRequest_EC_ND_ISN;
  scnIncludedSeqNo2 : ConfirmCartridgePresent_EC_ND_ISN;
  scnIncludedSeqNo4 : NoCartridgeDataPresent_EC_ND_ISN;
  scnIncludedSeqNo5 : MainScreenDisplay_1_NoDataOnCartridge;
end

EraseCartridgeRequest_NoDataOnCartridgeSSN in SequenceNumber, Token with
sequence_no
  sequenceNo : 1
end

NoCartridgeDataPresentNotificationSSN in SequenceNumber, Token with
sequence_no
  sequenceNo : 6
end

NoCartridgeDataPresentAcknowledgementSSN in SequenceNumber, Token with
sequence_no
  sequenceNo : 7
end

MainScreenDisplay_1_NoDataOnCartridgeSSN in SequenceNumber, Token with
sequence_no
  sequenceNo : 8
end

CheckforCartridgeRequest_EC_ND_ISN in SequenceNumber, Token with
sequence_no
  sequenceNo : 2
end

ConfirmCartridgePresent_EC_ND_ISN in SequenceNumber, Token with
sequence_no
  sequenceNo : 3
end

CheckCartridgeforDataRequest_EC_ND_ISN in SequenceNumber, Token with
sequence_no
  sequenceNo : 4
end

NoCartridgeDataPresent_EC_ND_ISN in SequenceNumber, Token with
sequence_no
  sequenceNo : 5
end
EraseCartridge_NoDataOnCartridgeTSV in TsnScenarioViewpoint, Token with
tsv_tsn_commm
  tsvTsnComm1 : EraseCartridgeRequestTextDescription;
  tsvTsnComm2 : NoCartridgeDataPresentNotificationTextDescription;
  tsvTsnComm3 : NoCartridgeDataPresentAcknowledgementTextDescription;
  tsvTsnComm4 : MainScreenDisplay_TTextDescription
end
EraseCartridge_NoDataOnCartridgeMSV in MscScenarioViewpoint, Token with
  msv_msc_commm
  msvMscComm1 : EraseCartridgeRequestMSCDescription;
  msvMscComm2 : NoCartridgeDataPresentNotificationMSCDescription;
  msvMscComm3 : NoCartridgeDataPresentAcknowledgementMSCDescription;
  msvMscComm4 : MainScreenDisplay_SMSDescription
end <
No Cartridge Data Present Notification Event Instantiation
NoCartridgeDataPresentNotification in CommunicationEvent, Token with interaction_type
  interactionType : "IP"
sequence_no
  sequenceNo1 : NoCartridgeDataPresentNotificationSSN follows_from :
  NoCartridgeDataPresent tsn_commm
    tsnCommunicationEvent :
      NoCartridgeDataPresentNotificationTextDescription
    tsnCommunicationEvent :
      NoCartridgeDataPresentNotificationMSCDescription
end
NoCartridgeDataPresentNotification in Message, Token with message_name
  messageName : "No Cartridge Data Notification"
  tsn_msg_parameter
    tsnMsgParameter :
      NoCartridgeDataNotificationTsnParameter
    msc_msg_parameter
      mscMsgParameter :
        NoCartridgeDataNotificationMscParameter
end
NoCartridgeDataNotificationTsnParameter in MessageDescription, Token with msg_parameter
  msgParameter :
    Message_Name : "No Cartridge Data Notification"
  msgParameter :
    msgParameter :
      "no data on the cartridge"
end
NoCartridgeDataNotificationMscParameter in MessageDescription, Token with msg_parameter
  msgParameter :
    msgParameter :
      "Report No Data on Cartridge"
NoCartridgeDataPresentNotificationMSCDescription in MscCommunication, Token with

link_name

linkName : "unspecified"
synchronisation

_Synchronisation : "sim"
frequency

_Frequency : "aperiodic"
delayed

_Delayed : False
msc_sender_instance

mscSenderInstance : MPSInstance
msc_receiver_instance

mscReceiverInstance : PilotInstance
msc_message

NoCartridgeDataNotification
end

end

NoCartridgeDataPresentNotificationMSCDescription in MscCommunication, Token with

link_name

linkName : "unspecified"
synchronisation

_Synchronisation : "sim"
frequency

_Frequency : "aperiodic"
delayed

_Delayed : False
msc_sender_instance

mscSenderInstance : MPSInstance
msc_receiver_instance

mscReceiverInstance : PilotInstance
msc_message

NoCartridgeDataNotification
end

No Cartridge Data Acknowledgment Event Instantiation

NoCartridgeDataPresentAcknowledgement in CommunicationEvent, Token with interaction_type

interactionType : "IP"
sequence_no

sequenceNo1 : NoCartridgeDataPresentAcknowledgementSS
follows_from

followsFrom : NoCartridgeDataPresentNotification
tsn_communication_event

tsnCommunicationEvent :
NoCartridgeDataPresentAcknowledgementTextDescription
msc_communication_event

mscCommunicationEvent :
NoCartridgeDataPresentAcknowledgementMSCDescription
end

NoCartridgeDataPresentAcknowledgementTextDescription in TsnCommunication, Token with communication_description

communicationDescription :
NoCartridgeDataPresentAcknowledgementEventText
end

NoCartridgeDataPresentAcknowledgementEventText in ScenarioEventNaturalLanguageStructure, Token with

mlns_composite

mlnsComposite1 : NCDPAEC1;
mlnsComposite2 : NCDPAEC2;
mlnsComposite3 : NCDPAEC3
mlns_plain_text

mlnsPlainText1 : NCDPAEPT1;
mlnsPlainText2 : NCDPAEPT2;
mlnsPlainText3 : NCDPAEPT3;
mlnsPlainText4 : NCDPAEPT4
tsn_sender_node

tsnSenderNode : PilotInstance
tsn_receiver_node

tsnReceiverNode : MPSInstance
tsn_message_node

tsnMessageNode : AcknowledgeNoData
end

NCDPAEC1 in MatraNLSComposite, Token with

preceding_fragment

precedingFragment : NCDPAEPT1
subject_node

subjectNode : PilotInstance

following_fragment

followingFragment : NCDPAEC2
end

NCDPAEC2 in MatraNLSComposite, Token with

preceding_fragment

precedingFragment : NCDPAEPT2
subject_node

subjectNode : AcknowledgeNoData
following_fragment

followingFragment : NCDPAEC3
end

NCDPAEC3 in MatraNLSComposite, Token with

preceding_fragment

precedingFragment : NCDPAEPT3
subject_node

subjectNode : MPSInstance
following_fragment

followingFragment : NCDPAEPT4
end

NCDPAEPT1 in PlainTextNode, Token with

mlns_text

mlnsText : "The 
end

NCDPAEPT2 in PlainTextNode, Token with

mlns_text

mlnsText : " sends an 
end

NCDPAEPT3 in PlainTextNode, Token with

mlns_text

mlnsText : " to the 
end

NCDPAEPT4 in PlainTextNode, Token with

mlns_text

mlnsText : "," end

AcknowledgeNoData in Message, Token with

message_name

messageName : "Acknowledge No Data"
tsn_msg_parameter

tsnMsgParameter

AcknowledgeNoDataTsnParameter
msc_msg_parameter

mscMsgParameter

AcknowledgeNoDataMscParameter
end

AcknowledgeNoDataTsnParameter in MessageDescription, Token with

msg-parameter

msgParameter : "acknowledgement"
end

AcknowledgeNoDataMscParameter in MessageDescription, Token with

msg_parameter

msgParameter : "OK: Acknowledge No Data"
end

NoCartridgeDataPresentAcknowledgementMSCDescription in MscCommunication, Token with

link_name

linkName : "unspecified"
synchronisation

_Synchronisation : "sim"
frequency

_Frequency : "aperiodic"
delayed

_Delayed : False
Instantiation of Erase Cartridge: Pilot Chooses Not to Erase Data

EraseCartridge_PilotChoosesNotToEraseData in Scenario, Token with
scenario_title
scenarioTitle : "Erase Cartridge - Pilot Chooses Not to Erase Data"

is_exception
isException : True

scenario_event
scenarioEvent1 : EraseCartridgeRequest;
scenarioEvent2 : ConfirmDataPresentonCartridge;
scenarioEvent3 : CancelEraseRequest;
scenarioEvent4 : MainScreenDisplay_1

includes_scenario
includesScenario1 :
  CheckforCartridge_NormalPath;
includesScenario2 :
  CheckCartridgeforData_NormalPath
included_event
includedEvent1 :
  CheckforCartridgeRequest;
includedEvent2 :
  ConfirmCartridgePresent;
includedEvent3 :
  CheckCartridgeforDataRequest;
includedEvent4 :
  ConfirmCartridgeDataPresent

-- again 'included_event' may be instantiated using a variation of the rule in App. A, Pt. 2 (vii.c)

scn_seq_no
scnSeqNo1 : EraseCartridgeRequest_PilotChoosesNotToEraseDataSSN;
scnSeqNo2 : ConfirmDataPresentonCartridge_PilotChoosesNotToEraseDataSSN;
scnSeqNo3 : CancelEraseRequestSSN;
scnSeqNo4 : MainScreenDisplay_1_PilotChoosesNotToEraseDataSSN

scn_included_seq_no
scnIncludedSeqNo1 :
  CheckforCartridgeRequest_EC_PCNTED_ISN;
scnIncludedSeqNo2 :
  ConfirmCartridgePresent_EC_PCNTED_ISN;
scnIncludedSeqNo3 :
  CheckCartridgeforDataRequest_EC_PCNTED_ISN;
scnIncludedSeqNo4 :
  ConfirmCartridgeDataPresent_EC_PCNTED_ISN

end

EraseCartridge_PilotChoosesNotToEraseDataTSV in TsnScenarioViewpoint, Token with
tsv_tsn_viewpoint
tsvTsnViewpoint :=
  EraseCartridgeRequestTextDescription;
tsvTsnComm1 :
  ConfirmDataPresentonCartridgeTextDescription;
tsvTsnComm2 :
  ConfirmCartridgePresentEC_PCNTED_ISN;
tsvTsnComm3 :
  CancelEraseRequestTextDescription;
tsvTsnComm4 :
  MainScreenDisplay_1TextDescription

end

EraseCartridge_PilotChoosesNotToEraseDataMSV in MscScenarioViewpoint, Token with
msv_msc_viewpoint
msvMscViewpoint :=
  EraseCartridgeRequestMSCDescription;
msvMscComm1 :
  ConfirmDataPresentonCartridgeMSCDescription;
msvMscComm2 :
  ConfirmCartridgePresentEC_PCNTED_ISN;
msvMscComm3 :
  CancelEraseRequestMSCDescription;
msvMscComm4 :
  MainScreenDisplay_1MSCDescription

end

Cancel Erase Request Event Instantiation
CancelEraseRequest in CommunicationEvent, Token with interaction_type: "IP"
sequence_no: sequenceNo1: CancelEraseRequestSSN
follows_from: ConfirmDataPresentonCartridge
tsn_communication_event: tsnCommunicationEvent:
CancelEraseRequestTextDescription
mac_communication_event: mscCommunicationEvent:
CancelEraseRequestMSCDescription
end
CancelEraseRequestTextDescription in TsnCommunication, Token with communication_description:
CancelEraseRequestTextDescription
end
CancelEraseRequestEventText in ScenarioEventNaturalLanguageStructure, Token with mnls_composite:
mnlsComposite1: CEREC1;
mnlsComposite2: CEREC2;
mnlsComposite3: CEREC3
mnls_plain_text:
mnlsPlainText1: CEREPT1;
mnlsPlainText2: CEREPT2;
mnlsPlainText3: CEREPT3;
mnlsPlainText4: CEREPT4
tsn_sender_node: tsnSenderNode: Pilotlnstance
tsn_receiver_node: tsnReceiverNode: MPSlnstance
tsn_message_node: tsnMessageNode:
CancelEraseCartridge
end
CEREC1 in Message, Token with message-name:
messageName: "Cancel Erase Cartridge"
ts_message_parameter: tsnMsgParameter:
CancelEraseCartridgeTsnParameter
msc_message_parameter: mscMsgParameter:
CancelEraseCartridgeMscParameter
end
CancelEraseCartridgeTsnParameter in MessageDescription, Token with msg_parameter:
msgParameter: "cancel the Erase Cartridge request"
end
CancelEraseCartridgeMscParameter in MessageDescription, Token with msg_parameter:
msgParameter: "Cancel Erase Cartridge"
end
CancelEraseRequestMSCDescription in Message, Token with link_name:
linkName: "unspecified"
synchronisation: _synchronisation: = "sim"
frequency: _Frequency: "aperiodic"
delayed: _Delayed: False
msc_sender_instance: mscSenderInstance: Pilotlnstance
msc_receiver_instance: mscReceiverInstance: MPSlnstance
msc_message: mscMessage: CancelEraseCartridge
end
ii. Instantiation of Retrieve From Cartridge (Timing Fragment)

**Set Data Timer (Event #11) Instantiation**

SetDataTimer1 in TimingEvent, Token with

```
......
  tsn_timing_event
tsnTimingEvent :
  SetDataTimer1TextDescription
  msc_timing_event
  mscTimingEvent :
  SetDataTimer1MSCDescription
end
```

SetDataTimer1TextDescription in TsnTiming, Token with timing_description

```
timingDescription : 
  SetDataTimer1EventText
end
```

SetDataTimer1EventText in ScenarioEventNaturalLanguageStructure, Token with

```
mnls_composite
  mnlsComposite1 : SDTIECI;
mnlsComposite2 : SDTIEC2;
mnlsComposite3 : SDTIEC3
mnls_plain_text
  mnlsPlainText1 : SDTIEPT1;
mnlsPlainText2 : SDTIEPT2;
mnlsPlainText3 : SDTIEPT3;
mnlsPlainText4 : SDTIEPT4
tsn_timer_set_node
tsnTimerSetNode : MPSInstance
tsn_timer_instance_node
tsnTimerInstanceNode : DataTimer
tsn_timer_duration
tsnTimerDuration :
  SetDataTimer1Duration
end
```

SDTIECI in MatraNLSComposite, Token with

```
preceding_fragment : SDTIEPT1
subject_node
  subjectNode : MPSInstance
following_fragment
  followingFragment : SDTIEC2
end
```

SDTIEC2 in MatraNLSComposite, Token with

```
preceding_fragment : SDTIEPT2
subject_node
  subjectNode : DataTimer
following_fragment
  followingFragment : SDTIEC3
end
```

SDTIEC3 in MatraNLSComposite, Token with

```
preceding_fragment : SDTIEPT3
subject_node
  subjectNode : SetDataTimer1Duration
following_fragment
  followingFragment : SDTIEPT4
end
```

SDTIEPT1 in PlainTextNode, Token with

```
  mnl5_text
  mnl5Text : "The 
end
```

SDTIEPT2 in PlainTextNode, Token with

```
  mnl5_text
  mnl5Text : " sets the "
end
```

SDTIEPT3 in PlainTextNode, Token with

```
  mnl5_text
  mnl5Text : " to "
end
```

SDTIEPT4 in PlainTextNode, Token with

```
  mnl5_text
  mnl5Text : "."
end
```

DataTimer in Timer, Token with timer_name

```
timerName : "DataTimer"
timer_duration
  timerDuration1 :
    SetDataTimer1Duration
end
```

SetDataTimer1Duration in TimerDuration, Token with duration

```
  _Duration : "10 Seconds"
end
```

SetDataTimer1MSCDescription in MscTiming, Token with

```
msc_timer_set_instance
  mscTimerSetInstance : MPSInstance
msc_timer_instance
    mscTimerinstance : DataTimer
msc_timer_duration
  mscTimerDuration :
    SetDataTimer1Duration
end
```

**Type_1 (Data) Retrieved Notification (Event #12) Instantiation**

RetrievedType1Notification in CommunicationEvent, Token with interaction_type

```
  interactionType : "SP"
......
  tsn_communication_event
tsnCommunicationEvent :
  RetrievedType1NotificationTextDescription
  msc_communication_event
    mscCommunicationEvent :
      RetrievedType1NotificationMSCDescription
end
```

RetrievedType1NotificationTextDescription in TsnCommunication, Token with communication_description

```
  communicationDescription :
    RetrievedType1NotificationEventText
end
```

RetrievedType1NotificationEventText in ScenarioEventNaturalLanguageStructure, Token with

```
mnls_composite
  mnlsComposite1 : RTINECI;
mnlsComposite2 : RTINEC2;
mnlsComposite3 : RTINEC3
mnls_plain_text
  mnlsPlainText1 : RTINEPT1;
mnlsPlainText2 : RTINEPT2;
mnlsPlainText3 : RTINEPT3;
mnlsPlainText4 : RTINEPT4
tsn_sender_node
```
Appendix C (Part 1)

\[
\text{tanSenderNode} : \text{MPSInstance} \\
\text{tsnReceiverNode} : \text{PilotInstance} \\
\text{tsn_messageNode} \\
\text{tanMessageNode} : \text{Type_1RetrievedNotification}
\]

\[
\text{RTINEC1 in MatraNLSCComposite, Token} \\
\text{preceding_fragment} \text{precedingFragment : RTINEPT1} \\
\text{subject_node} \text{subjectNode : MPSInstance} \\
\text{following_fragment} \text{followingFragment : RTINEC2}
\]

\[
\text{RTINEC2 in MatraNLSCComposite, Token} \\
\text{preceding_fragment} \text{precedingFragment : RTINEPT2} \\
\text{subject_node} \text{subjectNode : PilotInstance} \\
\text{following_fragment} \text{followingFragment : RTINEC3}
\]

\[
\text{RTINEC3 in MatraNLSCComposite, Token} \\
\text{preceding_fragment} \text{precedingFragment : RTINEPT3} \\
\text{subject_node} \text{subjectNode : Type_1RetrievedNotification} \\
\text{following_fragment} \text{followingFragment : RTINEPT4}
\]

\[
\text{RTINEPT1 in PlainTextNode, Token with} \\
\text{mlns_text} \text{mlnsText : "The "}
\]

\[
\text{RTINEPT2 in PlainTextNode, Token with} \\
\text{mlns_text} \text{mlnsText : " informs the "}
\]

\[
\text{RTINEPT3 in PlainTextNode, Token with} \\
\text{mlns_text} \text{mlnsText : " that "}
\]

\[
\text{RTINEPT4 in PlainTextNode, Token with} \\
\text{mlns_text} \text{mlnsText : "."}
\]

\[
\text{Type_1RetrievedNotification in Message, Token with} \\
\text{message_name} \text{messageName : "Type_1 Retrieved Notification"} \\
\text{msg_parameter} \text{msgParameter : "Type_1 Retrieved"}
\]

\[
\text{RetrievedType_1NotificationMSCDescription} \\
\text{in MscCommunication, Token with} \\
\text{link_name} \text{linkName : "unspecified"} \\
\text{synchronisation} \text{Synchronisation : "sim"} \\
\text{frequency} \text{Frequency : "aperiodic"} \\
\text{delayed} \text{Delayed : False}
\]

\[
\text{msc_sender_instance} \text{mscSenderInstance : MPSInstance} \\
\text{msc_receiver_instance} \text{mscReceiverInstance : PilotInstance} \\
\text{msc_message} \text{mscMessage : Type_1RetrievedNotification}
\]

\[
\text{Time-out Data Timer (Event # 13)} \\
\text{Instantiation} \\
\text{TimeoutDataTimer1 in TimingEvent, Token with} \\
\text{tsn_timing_event} \text{tsnTimingEvent : TimeoutDataTimer1TextDescription} \\
\text{msc_timing_event} \text{mscTimingEvent : TimeoutDataTimer1MSCDescription}
\]

\[
\text{TimeoutDataTimer1TextDescription in} \\
\text{TsnTiming, Token with} \\
\text{tsn_text} \text{tsnText : "Type_1 Retrieved"}
\]

\[
\text{TimeoutDataTimer1MSCDescription in} \\
\text{MscCommunication, Token with} \\
\text{link_name} \text{linkName : "unspecified"} \\
\text{synchronisation} \text{Synchronisation : "sim"} \\
\text{frequency} \text{Frequency : "aperiodic"} \\
\text{delayed} \text{Delayed : False}
\]

\[
\text{msc_sender_instance} \text{mscSenderInstance : MPSInstance} \\
\text{msc_receiver_instance} \text{mscReceiverInstance : PilotInstance} \\
\text{msc_message} \text{mscMessage : Type_1RetrievedNotification}
\]
Appendix C (Part 1)

TODTIEPT1 in PlainTextNode, Token with

  mnlstext : "The "
end

TODTIEPT2 in PlainTextNode, Token with

  mnlstext : " times-out."
end

TimeoutDataTimer1MSCDescription in

  MscTiming, Token with
  msc_host_on_timeout_instance
  mscHostOnTimeoutInstance : MPSInstance
  msc_timer_instance
  mscTimerInstance : DataTimer
end

Storage Type_1 Data (Event # 14)

StorageType1 in InternalActionEvent,

Token with

    .......
  tsn_action_event
  tsnActionEvent :
  StorageType1TextDescription
  msc_action_event
  mscActionEvent :
  StorageType1MSCDescription
end

StorageType1TextDescription in TsnAction,

Token with

  action_description
  StorageType1EventText
end

StorageType1EventText in

  ScenarioEventNaturalLanguageStructure,
  Token with
  mnlst_composite
    mnlstComposite1 : ST1EC1;
    mnlstComposite2 : ST1EC2
  mnlstPlain_text
    mnlstPlainText1 : ST1EPT1;
    mnlstPlainText2 : ST1EPT2
  mnlst_null
  mnlstNull : ST1EPT2
  tsn_sdr_rcr_node
  tsnSdrRcrNode : MPSInstance
  tsn_action_node
  tsnActionNode : StoreType_1Data
end

ST1EC1 in MatraNLSComposite, Token with

  preceding_fragment
  precedingFragment : ST1EPT1
  subject_node
  subjectNode : MPSInstance
  following_fragment
  followingFragment : ST1EC2
end

ST1EC2 in MatraNLSComposite, Token with

  preceding_fragment
  precedingFragment : ST1EPT1
  subject_node
  subjectNode : StoreType_1Data
  following_fragment
  followingFragment : ST1EPT2
end

ST1EPT1 in PlainTextNode, Token with

  mnlstext : "The "
end

ST1EPT2 in PlainTextNode, Token with

  mnlstext : " ."
end

StoreType_1Data in Action, Token with

  action_name
  actionName : "Store Type_1 Data"
  tsn_act_parameter
  tsnActParameter :
  StoreType1TsnParameter
end

StoreType1TsnParameter in

  MessageDescription, Token with
  action_parameter
  actionParameter : "stores type_1 data"
end

StorageType1MSCDescription in MscAction,

Token with

  msc_sdr_rcr_instance
  mscSdrRcrInstance : MPSInstance
  msc_system_action
  mscSystemAction : StoreType_1Data
end
Appendix C (Part 1)

III. Instantiation of Choose Mission and Aircraft (Event Group Fragment)

Data Item Request (Event #11) Instantiation

DataItemRequest in CommunicationEvent, Token with interaction_type
interactionType : "IR"

DataItemRequestTextDescription in CommunicationEvent, Token with communicationDescription
communicationDescription : DataItemRequestTextDescription

DataItemRequestMSCDescription in Message, Token with msgParameter
msgParameter : "supply a data item"

RequestDataItem in Message, Token with messageName
messageName : "Request Data Item"

Mission And Aircraft Data Item Provision (Event #12) Instantiation

MACDataItemProvision in CommunicationEvent, Token with interaction_type
interactionType : "IP"
Appendix C (Part 1)

MACDataItemProvisionMSCDescription in MscCommunication, Token with
  link_name
  linkName : "unspecified"
synchronisation
  _Synchronisation : = "sim"
frequency
  _Frequency : = "aperiodic"
delayed
  _Delayed : False
msc_sender_instance
  mscSenderInstance : MissionPlanInstance
msc_receiver_instance
  mscReceiverInstance : MPSInstance
msc_message
  mscMessage : Mission&ACDataItem
end

Storage Mission and Aircraft Data (Event #13) Instantiation

StorageMAC in InternalActionEvent, Token with
...
  tsn_action_event
  tsnActionEvent:
    StorageMACTextDescription
  msc_action_event
    mscActionEvent : StorageMACMSCDescription
end

StorageMACTextDescription in TsnAction, Token with
  action_description
    actionDescription : StorageMACEventText
end

StorageMACEventText in ScenarioEventNaturalLanguageStructure, Token with
  mls_composite
    mlsComposite1 : SMACEC1;
    mlsComposite2 : SMACEC2
  mls_plaintext
    mlsPlainText1 : SMACEPT1;
    mlsPlainText2 : SMACEPT2
  mls_null
    mlsNull : SMACENull
  tsn_sdr_rcc_node
    tsnSdrRccNode : MPSInstance
tsn_action_node
    tsnActionNode : StoreSelectedMission&ACData
end

SMACEC1 in MatraNLSComposite, Token with
  preceding_fragment
    precedingFragment : SMACEPT1
  subject_node
    subjectNode : MissionPlanInstance
  msc_msg_parameter
    mscMsgParameter : Mission&ACDataItemMscParameter
end

SMACEC2 in MatraNLSComposite, Token with
  preceding_fragment
    precedingFragment : SMACEPT2
  subject_node
    subjectNode : MissionPlanInstance
  msc_msg_parameter
    mscMsgParameter : Mission&ACDataItemMscParameter
end

C-45
SMACEPT1 in PlainTextNode, Token with
mls_text : "The "
end

SMACEPT2 in PlainTextNode, Token with
mls_text : "."
end

StoreSelectedMission&ACData in Action, Token with
action_name : "Store Selected Mission&AC Data"
actionName : "Store Selected Mission and Aircraft Data"
tsn_act_parameter :
  tsnActParameter : StoreSelectedMission&ACDataTsnParameter
msc_act_parameter :
  mscActParameter : StoreSelectedMission&ACDataMscParameter
end

StoreSelectedMission&ACDataTsnParameter in MessageDescription, Token with
action_parameter :
  actionParameter : "stores the data item for the selected Mission and Aircraft"
end

StoreSelectedMission&ACDataMscParameter in MessageDescription, Token with
action_parameter :
  actionParameter : "Store Selected Mission and Aircraft Data"
end

StorageMACMSECDescription in MscAction, Token with
msc_sdr_rcr_instance
  mscSdrRcrInstance : MPSInstance
msc_system_action
  mscSystemAction : StoreSelectedMission&ACData
end

Grouping for Events 11-13
Events11To13Group in EventGroup, Token with
group_event
  groupEvent1 : DataItemRequest;
  groupEvent2 : MACDataItemProvision;
  groupEvent3 : StorageMAC
grp_lb
grpLb : Events11To13GroupLB
grp_ub
grpUb : Events11To13GroupUB
end

Events11To13GroupLB in LowerBound, Token with
lower_bound
  lowerBound : "1"
end

Events11To13GroupUB in UpperBound, Token with
upper_bound
  upperBound : "card data types Mission Plan()"
end
Appendix C (Part 2) - Additional Scenarios For Hawk Mission Planning System

Use Case: Retrieve from Cartridge

Scenario Name: Retrieve from Cartridge - Normal Path

2. C[IR] The [MPS] asks the [Cartridge] whether there is a cartridge present in the hardware?
3. C[IP] The [Cartridge] responds to the [MPS] that there is a cartridge present in the hardware.
4. C[IR] The [MPS] asks the [Cartridge] whether there is data on the cartridge?
5. C[IP] The [Cartridge] responds to the [MPS] that there is data on the Cartridge.
6. A The [MPS] deletes any previously retrieved data.
7. C[IP] The [MPS] displays a list of retrievable data available to the [Pilot].
9. C[IR] The [MPS] requests that the [Cartridge] supply type_1 data.
10. C[IP] The [Cartridge] supplies type_1 data to the [MPS].
11. T The [MPS] sets the DataTimer to 10 seconds.
12. C[SP] The [MPS] informs the [Pilot] that type_1 data has been retrieved.
15. C[IR] The [MPS] requests that the [Cartridge] supply type_2 data.
16. C[IP] The [Cartridge] supplies type_2 data to the [MPS].
17. T The [MPS] sets the DataTimer to 10 seconds.
18. C[SP] The [MPS] informs the [Pilot] that type_2 data has been retrieved.
20. A The [MPS] stores type_2 data.
25. A The [MPS] creates a print object from the displayed data.
26. C[SP] The [MPS] sends the print object to the [Printer].
27. C[IP] The [MPS] informs the [Pilot] indicating the data is being printed.
29. C[IP] The [Pilot] informs the [MPS] that he wishes to exit this function and return to the Main Screen.
Appendix C (Part 2)

'MSC: Retrieve from Cartridge - Normal Path'
Appendix C (Part 2)

Pik MPS

22: Data to Display
   Request Data Type

23: Selected Data
   Request Data Type

24: Print Data
   Print Request

25: Create Print Object

26: Print Object

27: Print Message
   Printing Message

28: OK Print Acknowledge
   Print Acknowledge

29: Exit and Return
   Return to Main

30: Main Screen

'MSC: Retrieve from Cartridge - Normal Path (continued)'

C-49
Scenario Name: Retrieve from Cartridge - No Cartridge Present


2. **C[IR]** The [C[IR] MPS] asks the [C[IR] Cartridge] whether there [C[IR] is a cartridge present in the hardware? : Request Confirm Cartridge].

3. **C[IP]** The [C[IP] Cartridge] responds to the [C[IP] MPS] that [C[IP] there is no cartridge present in the hardware : No Data].

4. **C[IP]** The [C[IP] MPS] informs the [C[IP] Pilot] that there is [C[IP] no cartridge is present in the hardware : No Cartridge Notification].


![Diagram of the scenario]
Scenario Name: Retrieve from Cartridge - No Data on Cartridge

2. C[IR] The [5d MPS] asks the [5d Cartridge] whether there [5d is a cartridge present in the hardware?: Request Confirm Cartridge?]
3. C[IP] The [5d Cartridge] responds to the [5d MPS] that [5d there is a cartridge present in the hardware: Confirm Cartridge].
4. C[IR] The [5d MPS] asks the [5d Cartridge] whether there [5d is data on the cartridge?: Request Confirm Cartridge Data].
5. C[IP] The [5d Cartridge] responds to the [5d MPS] that [5d there is no data on the Cartridge: No Cartridge Data].
6. C[IP] The [5d MPS] informs the [5d Pilot] that there is [5d no data on the Cartridge: No Data Notification].

'MSC: Retrieve from Cartridge - No Data on Cartridge'
Scenario Name: Retrieve from Cartridge - Pilot Retrieves Subset of Data

2. C[IR] The [Pilot] asks the Cartridge whether there is a cartridge present in the hardware.
3. C[IP] The Cartridge responds to the [Pilot] that there is no cartridge present in the hardware.
4. C[IR] The [Pilot] asks the Cartridge whether there is data on the cartridge.
5. C[IP] The Cartridge responds to the [Pilot] that there is data on the Cartridge.
9. C[IR] The [Pilot] asks the Cartridge whether there is data on the cartridge.
12. C[IR] The [Pilot] requests that the Cartridge supply type 1 data.
14. T The [Pilot] sets the DataTimer to 10 seconds.
15. C[SP] The [Pilot] informs the Pilot that type 1 data has been retrieved.
17. A The [Pilot] stores type 1 data.
18. C[IR] The [Pilot] requests that the Cartridge supply type 2 data.
19. C[IP] The Cartridge supplies type 2 data to the [Pilot].
20. T The [Pilot] sets the DataTimer to 10 seconds.
21. C[SP] The [Pilot] informs the Pilot that type 2 data has been retrieved.
23. A The [Pilot] stores type 2 data.
26. C[IP] The [Pilot] informs the [Pilot] that he wishes to exit this function and return to the Main Screen.
Appendix C (Part 2)

MSC: Retrieve from Cartridge - Pilot Retrieves Subset of Data

(continued over page)
MSC: Retrieve from Cartridge - Pilot Retrieves Subset of Data (continued)
Use Case: Choose Mission and Aircraft

Scenario Name: New Mission From Open Missions

1. **C[SR]** The [MS Pilot] selects the [MS Request Mission Plan] option from the [MPS] Main Screen.

2. **A** The [MPS] [MS deletes data for existing Missions: Delete Current Mission Data].

3. **C[IR]** The [MPS] [MS requests open Missions and associated Aircraft: Request Open Mission & Aircraft] from the [Mission Plan].

4. **C[IP]** The [Mission Plan] supplies the [open Mission and Aircraft data: Open Mission & Aircraft] to the [MPS].

5. **C[IR]** The [MPS] displays the [open Missions] for the [Pilot] to select from.


9. **C[IP]** The [Pilot] [confirms the selected combination of Mission and Aircraft: Mission & Aircraft Combination] to the [MPS].


**ITERATION:** Lower Bound = 1 ; Upper Bound = card data types MissionPlan() 

11. **C[IR]** The [MPS] asks the [Mission Plan] to [supply a data item: Request Data Item].

12. **C[IP]** The [Mission Plan] supplies the [Data item for the selected Mission and Aircraft: Mission & Aircraft Data Item] to the [MPS].

13. **A** The [MPS] [stores the data item for the selected Mission and Aircraft: Store Selected Mission & Aircraft Data].

14. **C[SP]** The [MPS] indicates to the [Pilot] that [all data has been retrieved from the Mission Plan: Mission & Aircraft Retrieval Complete].


16. **C[IP]** The [MPS] displays to the [Pilot] the [Main Screen].

C-55
Appendix C (Part 2)

'MSC: Choose Mission & Aircraft - New Mission From Open Missions'
Scenario Name: New Mission From All Missions

2. A The [AR MPS] [req deletes data for existing Missions : Delete Current Mission Data].
3. C[IR] The [IR MPS] [req requests open Missions and associated Aircraft : Request Open Mission & Aircraft] from the [IR Mission Plan].

ITERATION: Lower Bound = 1 : Upper Bound = card data types MissionPlan()

16. A The [AR MPS] [req stores the data item for the selected Mission and Aircraft : Store Selected Mission & Aircraft Data].
17. C[IP] The [IP MPS] indicates to the [IP Pilot] that [req all data has been retrieved from the Mission Plan : Mission & Aircraft Retrieval Complete].
'MSC: Choose Mission & Aircraft - New Mission From All Missions'
Scenario Name: Pilot Does Not Initially Select an Aircraft or Mission

2. A The [esr MPS] [esr deletes data for existing Missions : Delete Current Mission Data].
6. C[S] The [esr Pilot] [msg confirms the selected combination of Mission and Aircraft - without selecting Mission or aircraft : Error No Mission] to the [esr MPS].
11. C[S] The [esr Pilot] [msg confirms the selected combination of Mission and Aircraft - without selecting aircraft : Error No Aircraft] to the [esr MPS].
15. C[S] The [esr Pilot] [msg confirms the selected combination of Mission and Aircraft : Mission&AC Combination] to the [esr MPS].

<table>
<thead>
<tr>
<th>Iteration:</th>
<th>Lower Bound = 1 ; Upper Bound = card data types MissionPlan()</th>
</tr>
</thead>
</table>

19. A The [esr MPS] [esr stores the data item for the selected Mission and Aircraft : Store Selected Mission&AC Data].

20. C[S] The [esr MPS] indicates to the [esr Pilot] that [msg all data has been retrieved from the Mission Plan : Mission&AC Retrieval Complete].
22. C[S] The [esr MPS] displays to the [esr Pilot] the [msg Main Screen].
Pilot

1. Create New Mission
   New Mission Request

2. Delete Current Mission Data

Mission Plan

2. Request Open Mission/Aircraft
   Request Open Mission/AAC

3. Request Open Mission/Aircraft
   Request Open Mission/AAC

4. Open Missions and Aircraft
   Open Missions/AAC

5. Open Missions
   Open Missions/AAC

   Error - No Mission

7. Error - Request Mission
   Select Mission Error Message

8. OK: Acknowledge Mission Error
   Acknowledge Missing Mission

9. Choice of Mission
   Mission Choice

10. Aircraft List
    Aircraft List

11. Mission & Aircraft Choice - No Aircraft
    Error - No Aircraft

12. Error - Request Aircraft
    Select Aircraft Error Message

13. OK: Acknowledge Aircraft Error
    Acknowledge Missing Aircraft

14. Choice of Aircraft
    Aircraft Choice

15. OK: Confirm Mission & Aircraft
    Mission/AAC Combination

16. Request Data on Selected Mission/Aircraft
    Request Selected Mission/AAC

17. Request Data Item
    Request Data Item

18. Data Item for Selection
    Mission/AAC Data Item

19. Store Selected Mission and Aircraft Data

20. Retrieval Complete
    Retrieval Complete Notification

21. OK: Retrieval Acknowledge
    Acknowledge Mission/AAC Retrieval

22. Main Screen

'CSC: Choose Mission & Aircraft - Pilot Does Not Initially Select Aircraft or Mission'
Scenario Name: No Data for Selected Mission in Mission Plan


2. A The [sr MPS] [sr deletes data for existing Missions: Delete Current Mission Data].


9. C[IP] The [sr Pilot] [msg confirms the selected combination of Mission and Aircraft: Mission&AAC Combination] to the [sr MPS].


Appendix C (Part 2)

'MSC: Choose Mission & Aircraft - No Data for Selected Mission in Mission Plan'
Appendix D - ARP 4761 Case Study: Updated Fault Trees

'BSCU Commands Braking in Absence of Brake Input Causing Inadvertent Braking - Updated Fault Tree (page 1)'

'BSCU Commands Braking in Absence of Brake Input Causing Inadvertent Braking - Updated Fault Tree (page 2)'

D-1
‘BSCU Commands Braking in Absence of Brake Input Causing Inadvertent Braking - Updated Fault Tree (page 3)’

Instantiation of Fault Analysis Tree Meta-model - Updated Fault Tree

Updated Event Definition ‘BSCUINADD’

```
EventA in Event, Token with
  updated_profile : EventAUpdatedProfile
end
EventAUpdatedProfile in
  UpdatedEventProfile, Token with
  _Type : “top”
  event_connective : OrGate1U
  actual_probability : EventAActualProbability
  preliminary_budget : EventAPreliminaryBudget
  actual_label : EventAActualLabel
end
EventAActualProbability in
  ActualProbability, Token with
  _probability : 1.23E-09
end
```

```
Gate definitions
```

```
OrGate1U in OrGate, Token with
  _Input1 : EventBUpdatedProfile;
  _Input2 : EventCUpdatedProfile
end
```

Updated Event Definition ‘BSCUUNDF’

```
EventB in Event, Token with
  updated_profile : EventBUpdatedProfile
end
EventBUpdatedProfile in
  UpdatedEventProfile, Token with
  _Type : “bas”
  actual_probability : EventBActualProbability
  preliminary_budget : EventB PreliminaryBudget
  actual_label : EventBActualLabel
end
EventBActualProbability in
  ActualProbability, Token with
  _probability : 1.23E-09
end
```

```
Gate definitions
```
probability
  _Probability : 0.00E+00
end

EventBActualSimpleEventDescription in SimpleEventDescription, Token with
  simple_description
    simpleDescription :
  EventBActualSimpleEventDescription
end

EventBActualSimpleEventDescription in SimpleEventDescription, Token with
  entity
    _Entity : "BSCU"
    _Condition : "Single Undetectable Failure Causing Inadvertent Braking"
end

Updated Event Definition 'BSCUDETFD'

EventC in Event, Token with
event_connective
  actual_probability
    actualProbability :
  preliminary_budget
    preliminaryBudget :
  preliminary_label
    actualLabel :
end

Updated Event Definition 'BSCUDETFD'

EventC in Event, Token with
event_connective
  actual_probability
    actualProbability :
  preliminary_budget
    preliminaryBudget :
  preliminary_label
    actualLabel :
end

Gate definitions

Updated Event Definition 'BSCUDETFD'

EventD in Event, Token with
event_connective
  actual_probability
    actualProbability :
  preliminary_budget
    preliminaryBudget :
  preliminary_label
    actualLabel :
end

Gate definitions

Updated Event Definition 'BSCUDETFD'

EventE in Event, Token with
event_connective
  actual_probability
    actualProbability :
  preliminary_budget
    preliminaryBudget :
  preliminary_label
    actualLabel :
end

Gate definitions

Updated Event Definition 'BSCUDETFD'

EventF in Event, Token with
event_connective
  actual_probability
    actualProbability :
  preliminary_budget
    preliminaryBudget :
  preliminary_label
    actualLabel :
end

Gate definitions

Updated Event Definition 'BSCUDETFD'

EventG in Event, Token with
event_connective
  actual_probability
    actualProbability :
  preliminary_budget
    preliminaryBudget :
  preliminary_label
    actualLabel :
end

Gate definitions
Appendix D

EventUpdatedProfile in UpdatedEventProfile, Token with type
   _Type : "int"
event_connective
   EventConnective : AndGate2U
   actual_probability
   actualProbability : 
   EventActualProbability
   preliminary_budget
   preliminaryBudget : 
   EventGPreliminaryBudget
   actual_label
   actualLabel : EventGPreliminaryLabel
end

EventActualProbability in ActualProbability, Token with probability
   _Probability : 6.00E-10
end

Gate definitions
AndGate1U in AndGate, Token with input _Input : HIEventSetU
end
AndGate2U in AndGate, Token with input _Input : JKEventSetU
end

HIEventSetU in EventSet, Token with event_profile
   eventProfile1 : EventHUpdatedProfile;
   eventProfile2 : EventIUpdatedProfile
end

JKEventSetU in EventSet, Token with event_profile
   eventProfile1 : EventJUpdatedProfile;
   eventProfile2 : EventKUpdatedProfile
end

Updated Event Definition 'BSCU1PSF'

EventH in Event, Token with updated_profile
   updatedProfile : EventHUpdatedProfile
end

EventHUpdatedProfile in UpdatedEventProfile, Token with type
   _Type : "bas"
   event_connective
   EventConnective : OrGate4U
   actual_probability
   actualProbability : 
   EventActualProbability
   preliminary_budget
   preliminaryBudget : 
   EventJPreliminaryBudget
   actual_rate
   actualRate : EventJActualRate
   actual_exposure
   actualExposure : 
   EventJPreliminaryExposure
   actual_label
   actualLabel : EventJPreliminaryLabel
end

EventJActualRate in Rate, Token with failure_rate
   failureRate : 1.06E-05
end

Updated Event Definition 'BSCU1MORV'

EventJ in Event, Token with updated_profile
   updatedProfile : EventJUpdatedProfile
end

EventJUpdatedProfile in UpdatedEventProfile, Token with type
   _Type : "int"
   event_connective
   EventConnective : OrGate5U
   actual_probability
   actualProbability : 
   EventActualProbability
   preliminary_budget
   preliminaryBudget : 
   EventJPreliminaryBudget
   actual_rate
   actualRate : EventJActualRate
   actual_exposure
   actualExposure : 
   EventJPreliminaryExposure
   actual_label
   actualLabel : EventJPreliminaryLabel
end

EventJActualProbability in ActualProbability, Token with probability
   _Probability : 1.63E-02
end

Updated Event Definition 'BSCU1CDF'

EventK in Event, Token with updated_profile
   updatedProfile : EventKUpdatedProfile
end

EventKUpdatedProfile in UpdatedEventProfile, Token with type
   _Type : "int"
   event_connective
   EventConnective : OrGate5U
   actual_probability
   actualProbability : 
   EventActualProbability
   preliminary_budget
   preliminaryBudget : 
   EventJPreliminaryBudget
   actual_rate
   actualRate : EventJActualRate
   actual_exposure
   actualExposure : 
   EventJPreliminaryExposure
   actual_label
   actualLabel : EventJPreliminaryLabel
end

EventJActualRate in Rate, Token with failure_rate
   failureRate : 1.43E-07

end

Updated Event Definition 'BSCU1PSF'

EventI in Event, Token with updated_profile
   updatedProfile : EventIUpdatedProfile
end

EventIUpdatedProfile in UpdatedEventProfile, Token with type
   _Type : "bas"
   actual_probability
   actualProbability : 
   EventActualProbability
   preliminary_budget
   preliminaryBudget : 
   EventIPreliminaryBudget
   actual_rate
   actualRate : EventIActualRate
   actual_exposure
   actualExposure : 
   EventIPreliminaryExposure
   actual_label
   actualLabel : EventIPreliminaryLabel
end

EventIActualProbability in ActualProbability, Token with probability
   _Probability : 4.42E-08
end

Updated Event Definition 'BSCU1MORV'

EventJ in Event, Token with updated_profile
   updatedProfile : EventJUpdatedProfile
end

EventJUpdatedProfile in UpdatedEventProfile, Token with type
   _Type : "int"
   event_connective
   EventConnective : OrGate4U
   actual_probability
   actualProbability : 
   EventActualProbability
   preliminary_budget
   preliminaryBudget : 
   EventJPreliminaryBudget
   actual_rate
   actualRate : EventJActualRate
   actual_exposure
   actualExposure : 
   EventJPreliminaryExposure
   actual_label
   actualLabel : EventJPreliminaryLabel
end

EventJActualProbability in ActualProbability, Token with probability
   _Probability : 1.06E-05
end

Updated Event Definition 'BSCU1CDF'

EventK in Event, Token with updated_profile
   updatedProfile : EventKUpdatedProfile
end

EventKUpdatedProfile in UpdatedEventProfile, Token with type
   _Type : "int"
   event_connective
   EventConnective : OrGate5U
   actual_probability
   actualProbability : 
   EventActualProbability
   preliminary_budget
   preliminaryBudget : 
   EventJPreliminaryBudget
   actual_rate
   actualRate : EventJActualRate
   actual_exposure
   actualExposure : 
   EventJPreliminaryExposure
   actual_label
   actualLabel : EventJPreliminaryLabel
end

EventJActualRate in Rate, Token with failure_rate
   failureRate : 1.06E-05
end
Appendix D

actual_probability
  actualProbability : EventKActualProbability
preliminary_budget
  preliminaryBudget : EventKPreliminaryBudget
actual_label
  actualLabel : EventKPreliminaryLabel
end

EventKActualProbability in ActualProbability, Token with probability
  _Probability : 3.68E-08
end

Gate definitions

OrGate4U in OrGate, Token with input
  _Input1 : EventLUpdatedProfile;
  _Input2 : EventMUpdatedProfile
end

OrGate5U in OrGate, Token with input
  _Input1 : EventMUpdatedProfile;
  _Input2 : EventOUpdatedProfile
end

Updated Event Definition 'BSCU1MOPV'

EventL in Event, Token with updated_profile
  updatedProfile : EventLUpdatedProfile
end

EventLUpdatedProfile in UpdatedEventProfile, Token with type
  _Type : "bas"
actual_probability
  actualProbability : EventLActualProbability
preliminary_budget
  preliminaryBudget : EventLActualBudget
actual_rate
  actualRate : EventLActualRate
actual_exposure
  actualExposure : EventLActualExposure
actual_label
  actualLabel : EventLPreliminaryLabel
end

EventLActualProbability in ActualProbability, Token with probability
  _Probability : 1.63E-02
end

EventLActualRate in Rate, Token with failure_rate
  failureRate : 1.63E-07
end

Updated Event Definition 'BSCU1N/GF'

EventO in Event, Token with updated_profile
  updatedProfile : EventOUpdatedProfile
end

EventOUpdatedProfile in UpdatedEventProfile, Token with type
  _Type : "bas"
actual_probability
  actualProbability : EventOActualProbability
preliminary_budget
  preliminaryBudget : EventOActualBudget
actual_rate
  actualRate : EventOActualRate
actual_exposure
  actualExposure : EventOActualExposure
actual_label
  actualLabel : EventOPreliminaryLabel
end

EventOActualProbability in ActualProbability, Token with probability
  _Probability : 5.17E-09
end

EventOActualRate in Rate, Token with failure_rate
  failureRate : 1.24E-06
end

Gate definitions
Appendix D

orGate6U in OrGate, Token with input
  _Input1 : EventPUpdatedProfile;
  _Input2 : EventQUpdatedProfile
end

Updated Event Definition 'BSCU1CPU6'

EventP in Event, Token with updated_profile
  updatedProfile : EventPUpdatedProfile
end

EventPUpdatedProfile in UpdatedEventProfile, Token with type
  _Type = "bas"
  actual_probability
    actualProbability : EventPActualProbability
  preliminary_budget
    preliminaryBudget : EventPPreliminaryBudget
  actual_rate
    actualRate : EventPActualRate
  actual_exposure
    actualExposure : EventPPreliminaryExposure
  actual_label
    actualLabel : EventPPreliminaryLabel
end

EventPActualProbability in ActualProbability, Token with probability
  _Probability : 3.17E-08
end

EventPActualRate in Rate, Token with failure_rate
  failureRate : 7.60E-06
end

Updated Event Definition 'BSCU1CPUDE'

EventQ in Event, Token with updated_profile
  updatedProfile : EventQUpdatedProfile
end

EventQUpdatedProfile in UpdatedEventProfile, Token with type
  _Type = "und"
  actual_probability
    actualProbability : EventQActualProbability
  preliminary_budget
    preliminaryBudget : EventQPreliminaryBudget
  actual_label
    actualLabel : EventQPreliminaryLabel
end

EventQActualProbability in ActualProbability, Token with probability
  _Probability : 0.00E+00
end

Gate definitions

AndGate3U in AndGate, Token with input
  _Input : RSEventSetU
end

RSEventSetU in EventSet, Token with event_profile
  eventProfile1 : EventRUpdatedProfile;
  eventProfile2 : EventSUpdatedProfile
end

Updated Event Definition 'BSCU1CPFU6'

EventR in Event, Token with updated_profile
  updatedProfile : EventRUpdatedProfile
end

EventRUpdatedProfile in UpdatedEventProfile, Token with type
  _Type = "int"
  event_connective
    EventConnective : OrGate7U
  actual_probability
    actualProbability : EventRActualProbability
  preliminary_budget
    preliminaryBudget : EventRPreliminaryBudget
  actual_label
    actualLabel : EventRPreliminaryLabel
end

EventRActualProbability in ActualProbability, Token with probability
  _Probability : 2.31E-4
end

Updated Event Definition 'BSCU2INADCM'

Events in Event, Token with updated_profile
  updatedProfile : EventSUpdatedProfile
end

EventSUpdatedProfile in UpdatedEventProfile, Token with type
  _Type = "int"
  event_connective
    EventConnective : OrGate8U
  actual_probability
    actualProbability : EventSActualProbability
  preliminary_budget
    preliminaryBudget : EventSPreliminaryBudget
  actual_label
    actualLabel : EventSPreliminaryLabel
end

EventSActualProbability in ActualProbability, Token with probability
  _Probability : 1.23E-9
end

Gate definitions

OrGate7U in OrGate, Token with input
  _Input1 : EventTUpdatedProfile;
  _Input2 : EventUUpdatedProfile
end

OrGate8U in OrGate, Token with input
  _Input1 : EventVUpdatedProfile;
  _Input2 : EventWUpdatedProfile
end

Updated Event Definition 'BSCU1TF'

EventT in Event, Token with updated_profile
  updatedProfile : EventTUpdatedProfile
end
Appendix D

Updated Event Definition `SWIFAIL2'

EventU in Event, Token with
updated_profile : EventUUpdatedProfile end

EventUUpdatedProfile in
UpdatedEventProfile, Token with type
  _Type : "bas"
  actual_probability
  actualProbability :
  EventTActualProbability
  preliminary_budget
  preliminaryBudget :
  EventTPreliminaryBudget
  actual_rate
  actualRate : EventTActualRate
  actual_exposure
  actualExposure :
  EventTPreliminaryExposure
  actual_label
  actualLabel : EventTPreliminaryLabel end

EventTActualProbability in
ActualProbability, Token with probability
  _Probability : 2.18E-04 end

EventTActualRate in Rate, Token with failure_rate
  failureRate : 4.36E-05 end

Updated Event Definition `BSCU2PSIND'

EventV in Event, Token with
updated_profile : EventVUpdatedProfile end

EventVUpdatedProfile in
UpdatedEventProfile, Token with
type
  _Type : "int"
  event_connective
    EventConnective : AndGate4U
    actual_probability
      EventTActualProbability
      preliminary_budget
      preliminaryBudget :
      EventTPreliminaryBudget
    actual_label
      EventTPreliminaryLabel end

EventVActualProbability in
ActualProbability, Token with probability
  _Probability : 6.32E-10 end

Gate definitions
AndGate4U in AndGate, Token with
input
  _Input : XYEventSetU end

XYEventSetU in EventSet, Token with event_profile
  eventProfile1 : EventXUpdatedProfile;
  eventProfile2 : EventYUpdatedProfile end

AndGate5U in AndGate, Token with
input
  _Input : ZAAEventSetU end

ZAAEventSetU in EventSet, Token with event_profile
  eventProfile1 : EventZUpdatedProfile;
  eventProfile2 : EventAAUpdatedProfile end

Updated Event Definition `BSZPSMOFV'

EventX in Event, Token with
updated_profile : EventXUpdatedProfile end

EventXUpdatedProfile in
UpdatedEventProfile, Token with

Updated Event Definition `BSCU2CDIND'

EventW in Event, Token with
updated_profile : EventWUpdatedProfile end

EventWUpdatedProfile in
UpdatedEventProfile, Token with

Updated Event Definition `BSZPSMOFV'

EventX in Event, Token with
updated_profile : EventXUpdatedProfile end

EventXUpdatedProfile in
UpdatedEventProfile in
Appendix D

Updated Event Profile, Token with type
   _Type : "bas"
   actual_probability
   EventXActualProbability
preliminary_budget
   EventXPreliminaryBudget
actual_rate
   EventXActualRate
actual_exposure
   EventXPreliminaryExposure
actual_label
   eventConnective : OrGate9U

   EventXActualProbability in ActualProbability, Token with probability
   _Probability : 1.43E-02

   EventXActualRate in Rate, Token with failure_rate
   failureRate : 1.43E-07

Updated Event Definition 'BSCU2PSF'

EventY in Event, Token with updated_profile
   updatedProfile : EventYUpdatedProfile
end

EventYUpdatedProfile in UpdatedEventProfile, Token with type
   _Type : "bas"
   actual_probability
   EventYActualProbability
preliminary_budget
   EventYPreliminaryBudget
actual_rate
   EventYActualRate
actual_exposure
   EventYPreliminaryExposure
actual_label
   eventConnective : OrGate9U

   EventYActualProbability in ActualProbability, Token with probability
   _Probability : 4.42E-08

   EventYActualRate in Rate, Token with failure_rate
   failureRate : 1.06E-05

Updated Event Definition 'BSCU2MORV'

EventZ in Event, Token with updated_profile
   updatedProfile : EventZUpdatedProfile
end

EventZUpdatedProfile in UpdatedEventProfile, Token with type
   _Type : "bas"
   actual_probability
   EventZActualProbability
preliminary_budget
   EventZPreliminaryBudget
actual_label
   EventZPreliminaryLabel

   EventZActualProbability in ActualProbability, Token with probability
   _Probability : 1.63E-02

Updated Event Definition 'BSCU2CDF'

EventAA in Event, Token with updated_profile
   updatedProfile : EventAAUpdatedProfile
end

EventAAUpdatedProfile in UpdatedEventProfile, Token with type
   _Type : "int"
   event_connective
   EventConnective : OrGate10U
   actual_probability
   EventAAActualProbability
preliminary_budget
   EventAAPreliminaryBudget
actual_label
   EventAAPreliminaryLabel

   EventAAActualProbability in ActualProbability, Token with probability
   _Probability : 3.68E-08

Gate definitions

OrGate9U in OrGate, Token with input
   _Input1 : EventBBUpdatedProfile;
   _Input2 : EventCCUpdatedProfile
end

OrGate10U in OrGate, Token with input
   _Input1 : EventDDUpdatedProfile;
   _Input2 : EventEEUpdatedProfile
end

Updated Event Definition 'BSCU2MOFV'

EventBB in Event, Token with updated_profile
   updatedProfile : EventBBUpdatedProfile
end

EventBBUpdatedProfile in UpdatedEventProfile, Token with type
   _Type : "bas"
   actual_probability
   EventBBAactualProbability
preliminary_budget
   EventBBPreliminaryBudget
actual_rate
   EventBBActualRate
actual_exposure
   EventBBPreliminaryExposure

D-8
Appendix D

actual_label
actualLabel : EventBPREliminaryLabel
end

EventBBActualProbability in ActualProbability, Token with probability _Probability : 1.63E-02
end

EventBBActualRate in Rate, Token with failure_rate failureRate : 1.63E-07
end

Updated Event Definition 'BSCU2CMDE'

EventCC in Event, Token with updated_profile updatedProfile :
EventCCUpdatedProfile
end

EventCCUpdatedProfile in UpdatedEventProfile, Token with type _Type : 'und'
actual_probability
actualProbability : EventCCActualProbability
preliminary_budget preliminaryBudget :
EventCCPreliminaryBudget
actual_label
actualLabel : EventCCPreliminaryLabel
end

EventCCActualProbability in ActualProbability, Token with probability _Probability : 0.00E+00
end

Updated Event Definition 'BSCU2CPUD8'

EventDD in Event, Token with updated_profile updatedProfile :
EventDDUpdatedProfile
end

EventDDUpdatedProfile in UpdatedEventProfile, Token with type _Type : 'int'
event_connective
EventConnective : OrGate11U
actual_probability
actualProbability : EventDDActualProbability
preliminary_budget preliminaryBudget :
EventDDPreliminaryBudget
actual_label
actualLabel : EventDDPreliminaryLabel
end

EventDDActualProbability in ActualProbability, Token with probability _Probability : 3.17E-08
end

Updated Event Definition 'BSCU2CPUD8'

EventEE in Event, Token with updated_profile updatedProfile :
EventEEUpdatedProfile
end

EventEEUpdatedProfile in UpdatedEventProfile, Token with type _Type : 'bas'
actual_probability
actualProbability : EventEEActualProbability
preliminary_budget preliminaryBudget :
EventEEPreliminaryBudget
actual_label
actualLabel : EventEEPreliminaryLabel
end

EventEEActualProbability in ActualProbability, Token with probability _Probability : 5.17E-09
end

Gate definitions

OrGate11U in OrGate, Token with input _Input1 : EventFFUpdatedProfile;
_INPUT2 : EventGGUpdatedProfile
end

Updated Event Definition 'BSCU2CPUEF'

EventFF in Event, Token with updated_profile updatedProfile :
EventFFUpdatedProfile
end

EventFFUpdatedProfile in UpdatedEventProfile, Token with type _Type : 'bas'
actual_probability
actualProbability : EventFFActualProbability
preliminary_budget preliminaryBudget :
EventFFPreliminaryBudget
actual_label
actualLabel : EventFFPreliminaryLabel
end

EventFFActualProbability in ActualProbability, Token with probability _Probability : 3.17E-08
end

EventFFActualRate in Rate, Token with failure_rate failureRate : 7.60E-06
end

Updated Event Definition 'BSCU2CPUD8'

EventGG in Event, Token with updated_profile updatedProfile :

Appendix D

Even tGGUpdatedProfile
end

EventGGUpdatedProfile in UpdatedEventProfile, Token with
type _Type : "und"
actual_probability : EventGGActualProbability
preliminary_budget : EventGGPreliminaryBudget
actual_label : EventGGActualLabel
end

EventGGActualProbability in ActualProbability, Token with
probability : EventGGActualProbability

Instantiation of Updated Fault Tree

BacusUpdatedFT in UpdatedFaultTree, Token with
ft_andGate1 : AndGate1U;
ft_andGate2 : AndGate2U;
ft_andGate3 : AndGate3U;
ft_andGate4 : AndGate4U;
ft_or_gate : OrGate1U;
ft_or_gate2 : OrGate2U;
ft_or_gate3 : OrGate3U;
ft_or_gate4 : OrGate4U;
ft_or_gate5 : OrGate5U;
ft_or_gate6 : OrGate6U;
ft_or_gate7 : OrGate7U;
ft_or_gate8 : OrGate8U;
ft_or_gate9 : OrGate9U;
ft_or_gate10 : OrGate10U;
ft_or_gate11 : OrGate11U;

ft_event_set
ft_event1 : HIEventSetU;
ft_event2 : JXEventSetU;
ft_event3 : RSEventSetU;
ft_event4 : XYEventSetU;
ft_event5 : ZAAEventSetU;

ft_event_profile
ft_eventProfile1;
ft_eventProfile2;
ft_eventProfile3;
ft_eventProfile4;
ft_eventProfile5;
ft_eventProfile6;
ft_eventProfile7;
ft_eventProfile8;
ft_eventProfile9;
ft_eventProfile10;
ft_eventProfile11;
ft_eventProfile12;
ft_eventProfile13;
ft_eventProfile14;
ft_eventProfile15;
ft_eventProfile16;

ft_budget
ft_budget1 ; EventAPreliminaryBudget;
ft_budget2 ; EventBPreliminaryBudget;
ft_budget3 ; EventCPreliminaryBudget;
ft_budget4 ; EventDPreliminaryBudget;
ft_budget5 ; EventEPreliminaryBudget;
ft_budget6 ; EventFPreliminaryBudget;
ft_budget7 ; EventGPreliminaryBudget;
ft_budget8 ; EventHPreliminaryBudget;
ft_budget9 ; EventIPreliminaryBudget;
ft_budget10 ; EventJPreliminaryBudget;
ft_budget11 ; EventKPreliminaryBudget;
ft_budget12 ; EventLPreliminaryBudget;
ft_budget13 ; EventMPreliminaryBudget;
ft_budget14 ; EventNPreliminaryBudget;
ft_budget15 ; EventOPreliminaryBudget;
ft_budget16 ; EventPPreliminaryBudget;
ft_budget17 ; EventQPreliminaryBudget;
ft_budget18 ; EventRPreliminaryBudget;
ft_budget19 ; EventSPreliminaryBudget;
ft_budget20 ; EventTPreliminaryBudget;
ft_budget21 ; EventUPreliminaryBudget;
ft_budget22 ; EventVPreliminaryBudget;
ft_budget23 ; EventWPreliminaryBudget;
ft_budget24 ; EventXPreliminaryBudget;
ft_budget25 ; EventYPreliminaryBudget;
ft_budget26 ; EventZPreliminaryBudget;
ft_budget27 ; EventAAPreliminaryBudget;
ft_budget28 ; EventBABreliminaryBudget;
ft_budget29 ; EventCABreliminaryBudget;
ft_budget30 ; EventDABreliminaryBudget;
ft_budget31 ; EventEABreliminaryBudget;
ft_budget32 ; EventFABreliminaryBudget;
ft_budget33 ; EventGABreliminaryBudget;

D-10
Appendix D

ftActual11 : EventKAActualProbability;
ftActual12 : EventLAActualProbability;
ftActual13 : EventMAActualProbability;
ftActual14 : EventNAActualProbability;
ftActual15 : EventOAActualProbability;
ftActual16 : EventPAActualProbability;
ftActual17 : EventQAActualProbability;
ftActual18 : EventRAActualProbability;
ftActual19 : EventSAActualProbability;
ftActual20 : EventTAActualProbability;
ftActual21 : EventUAActualProbability;
ftActual22 : EventVAActualProbability;
ftActual23 : EventWAActualProbability;
ftActual24 : EventXAActualProbability;
ftActual25 : EventYXActualProbability;
ftActual26 : EventZActualProbability;
ftActual27 : EventAAActualProbability;
ftActual28 : EventBBActualProbability;
ftActual29 : EventCCActualProbability;
ftActual30 : EventDDActualProbability;
ftActual31 : EventEEActualProbability;
ftActual32 : EventFFActualProbability;
ftActual33 : EventGGActualProbability

end

ft_rate
ftRate1 : EventJAActualRate;
ftRate2 : EventLAActualRate;
ftRate3 : EventMAActualRate;
ftRate4 : EventNAActualRate;
ftRate5 : EventOAActualRate;
ftRate6 : EventPAActualRate;
ftRate7 : EventQAActualRate;
ftRate8 : EventRAActualRate;
ftRate9 : EventSAActualRate;
ftRate10 : EventTAActualRate;
ftRate11 : EventUAActualRate;
ftRate12 : EventVAActualRate;
ftRate13 : EventWAActualRate

ft_exposure
ftExposure1 : EventJFActualExposure;
ftExposure2 : EventLFActualExposure;
ftExposure3 : EventMFActualExposure;
ftExposure4 : EventNFActualExposure;
ftExposure5 : EventOFActualExposure;
ftExposure6 : EventPFActualExposure;
ftExposure7 : EventQFActualExposure;
ftExposure8 : EventRFActualExposure;
ftExposure9 : EventSFActualExposure;
ftExposure10 : EventTFActualExposure;
ftExposure11 : EventUFActualExposure;
ftExposure12 : EventVFActualExposure

end
Appendix D

(Partial) Instantiation of Fault Tree Analysis

BscuFTA in FaultTreeAnalysis, Token with

\[
\text{fta\_updated\_tree}
\]
\[
\text{ftaUpdatedTree : BscuSsaUpdatedFT}
\]
end
Appendix - E
This page deliberately left blank
Appendix E - Spatial Build Associations

Contains

Supports

Anchors

Holds

Surrounds

Guides

BuildAssociation

source

target

Module

IsParallelTo

IsPerpendicularTo

IsCollinearWith

IsConcentricWith

IsCoplanarWith

IsAlignedWith

IsSkewedFrom

IsAdjoinedTo

IsSeparatedFrom

IsSegregatedFrom

‘Catalogue of Spatial Relations’