Towards
Model-Based Systems Engineering of Autonomous Cyber-Physical Systems

A thesis by

Hansen Salim

Submitted in partial fulfilment of the requirements

for the degree of

Doctor of Philosophy

University of Newcastle upon Tyne

Newcastle upon Tyne, UK

2022

(Submitted October 19, 2022)

I dedicate this thesis to my loving family

Acknowledgements

Many people have supported me during my Ph.D. studies. First, I would like to thank my supervisor, Prof. John Fitzgerald, who has given me the opportunity to undertake a PhD and provided me with invaluable support and guidance throughout my time at Newcastle University.

I would like to thank my family and especially my wife, Sabina. Without your continued love and support, I know I would never have reach the completion of this task.

I am thankful for colleagues that have influenced me throughout my research. In particular, thank you to Mark, Ken, and Carl that have provided invaluable comments and suggestions to my work. I also thank Richie, Troy, and Milad for moments that go beyond just research.

I am immensely blessed by great people surrounding me. Thank you all!

Abstract

Cyber-Physical Systems (CPSs) such as smart grids, autonomous automobiles, and medical monitoring systems are becoming increasingly common. Autonomy (i.e., the ability of a system to achieve high-level objective(s) with limited or no external assistance) is integral to future CPS applications, but it comes with non-trivial design challenges. It is especially challenging to validate Autonomous CPSs (ACPSs) that adapt their behaviour to dynamically changing environments. Model-Based Systems Engineering (MBSE) has been recognised as a way to manage the complexity of CPSs as it supports system design at a high level of abstraction, allowing the analysis of designs and identification of key features, including defects, before they reach implementation. However, existing MBSE techniques and frameworks are targeted for generic CPSs that do not involve the challenges found in ACPSs. The aim of our research is to define an MBSE framework that supports the architecting of ACPSs from requirements to validation.

As the first step to realising this aim, we propose a MBSE ontology for ACPS. This provides a holistic view on the architecture of ACPS by combining several concepts such as MBSE, CPS and Autonomous Systems into one coherent ontology. Our aim is realised as an Architectural Framework for ACPS (ACPSAF), built on top of the ACPS Ontology. ACPSAF is an encapsulation of a minimum set of practices and requirements for artefacts that describe the ACPS architecture. ACPSAF proposes five Perspectives to capture the architecture of ACPSs across the whole MBSE life cycle. In order to validate the framework, two case studies are carried out where ACPSs are designed using ACPSAF. The results suggest that ACPSAF provides information in the requirements, architecture and analysis of ACPS that can be used with existing ACPS specific techniques such as runtime monitoring approach and automated test case generation to gain confidence in the dependability of ACPSs.

Contents

List of Figures

Introduction 1

This thesis is about the engineering of Autonomous Cyber-Physical Systems (ACPSs). Such systems are attracting considerable interest because of their potential to reduce involvement of human operators to achieve complex functionalities. They also attract major concerns because of the level of reliance that may be placed on them in spite of their complexity and scale. This chapter introduces the significance of dependable ACPSs in Section 1.1, motivating the challenges for research objectives described in Section 1.2. In Section 1.3, we outline the contributions and structure of the thesis, and identify the author's published work on which it is built.

1.1 Motivation

Cyber-Physical Systems (CPSs) such as smart grids, autonomous automobiles, and medical monitoring systems are becoming increasingly common [3]. A CPS consists of computation, communication, and control (i.e., cyber) components that are tightly combined with physical processes (e.g., mechanical, electrical) where physical processes affect computations and vice versa. Understanding CPS structures and associated functionalities requires a multidisciplinary body of knowledge and engineering capabilities. Knowledge of methodologies and technologies from a wide range of domains such as computer science, embedded systems, control systems, co-modelling, and system life-cycles have been used to gauge the competency of CPS engineers [4]. As a result, techniques from the cross-disciplinary field of Systems Engineering (SE) have been utilised in the engineering of CPSs [5, 6, 7].

Model-Based Systems Engineering (MBSE) is an SE methodology that focuses on the practice of developing domain models to manage complex systems over their life cycles [8]. MBSE is emerging as a viable way of evaluating CPSs [9, 10, 11] and is gradually becoming the mainstream method for the engineering of CPSs [12, 13]. However, there are growing concerns that the current SE-based approach is unable to cope with future CPSs that are growing in both scale and complexity [14, 15, 10]. In particular, autonomy (i.e., the ability of a system to operate with limited or no human intervention) is an integral part of future CPS applications that come with non-trivial design challenges [16, 17, 18, 19].

In classical systems design, the developers assume that they have complete (or at least sufficient) knowledge of the real environment in which the system operates. This assumption is not new, but it is particularly critical for ACPSs. Accidents involving Google[1], Tesla[2], and Uber[3] vehicles show that unexpected features of the environment during ACPS operations can lead to unexpected and fatal system behaviour. As ACPSs are being increasingly deployed around human activity, there is greater reliance on ACPSs to be dependable. Some research suggests approaches that attempt to capture all possible environment situations during the design time [20, 19], while other works suggest that reliance should no longer be placed on ACPS validation during design time, but rather, methods should be developed to monitor ACPSs during runtime [20, 17]. Regardless of the different methods proposed, it is commonly agreed that current methods for the verification and validation of systems are not sufficient for ACPSs [21].

Some applications of ACPSs incorporate Machine Learning (ML) techniques such as Reinforcement Learning (RL), which allows the ACPS to form its own adaptive behaviour by learning from the environment [22, 23]. The ability to learn and adopt new behaviour during runtime can be especially useful when the ACPSs are required for deployment into unknown environments (i.e., where a mathematical model of the environment may not be available) [24, 25]. For example, RL techniques could be used to enable Unmanned Aerial Vehicles (UAVs) for search and rescue missions to locate a missing human after a natural disaster in unknown environments [26].

Previous experiences in ML projects have shown that runtime learning could result in unexpected, undesirable behaviour. For example, Microsoft launched a Twitter bot: A machine learning project that engages and communicates with people aged 18-24[4]. The bot was designed to interact with Twitter users and learn from its conversations. Within a day of its release, the bot had learned to produce inflammatory comments such that its behaviour had to be edited by Microsoft. Unlike the chatbot, unexpected behaviour arising from ML behaviour in ACPS applications such as autonomous vehicles [27] and smart grid [28, 29] could be very costly and even possibly fatal.

The need for dependable ACPSs calls for a different approach towards autonomy. Aniculaesei et al. [17] argue that since ACPSs have to consider infinite environment scenarios and could learn new behaviour at runtime, an autonomous system is never stable in its implementation even after delivery. Aniculaesei [17] also continues with saying that autonomous systems should be regarded as having 'incomplete specifications'

[1] 'Google's Self-Driving Car Caused Its First Crash', https://www.wired.com/2016/02/googles-self-driving-car-may-caused-first-crash/, (Feb 29, 2016)

[2] 'Tesla deaths', https://www.tesladeaths.com/, Last accessed: Nov 4, 2019

[3] 'Self-driving Uber killed a pedestrian as human safety driver watched', https://www.vice.com/en_us/article/kzxq3y/self-driving-uber-killed-a-pedestrian-as-human-safety-driver-watched, (Mar 19, 2018)

[4] Jane Wakefield, 'Microsoft chatbot is taught to swear on Twitter', https://www.bbc.co.uk/news/technology-35890188, (Mar 24, 2016)

that cannot be verified and validated with traditional techniques which typically assume that a complete specification can be formed during design time. The need to manage complexity of autonomous systems has called for autonomy-explicit approaches, where characteristics of autonomy (e.g., autonomy requirements [30, 31]) are explicitly defined and considered in the architecture of autonomous systems [32, 33, 34].

We conjecture that by adopting an autonomy-explicit approach, the state of the art of MBSE could be advanced to meet with the increasing demand for complex, dependable ACPSs and improve the engineering process of ACPS. This thesis proposes a characterisation of ACPSs in the form of an MBSE ontology, an MBSE framework that utilises this ontology to support the architecting of ACPSs and demonstrate methods to integrate MBSE models with verification and validation techniques for ACPSs.

1.2 Research Objectives

In this thesis, we aim to address the following research question:

> Can we provide a framework to support the architecting of ACPSs such that autonomy can be evaluated in the MBSE of ACPSs?

Evaluating autonomy means being able to describe, identify, and reason about it systematically during design (e.g., identifying and verifying autonomy requirements). Therefore, autonomy needs to be explicitly represented in the design. Architecting means the process of "conceiving, defining, expressing, documenting, communicating, certifying proper implementation of, maintaining, and improving an architecture throughout a system's life cycle", which is a definition adopted from ISO/IEC 42010[35].

We derive three main objectives to address the research question: To provide an MBSE ontology for ACPS, use the ontology to provide a framework that supports the architecting of ACPS, and demonstrate methods that allow techniques for dependable ACPS to be used in the MBSE design process for ACPS.

Objective 1. To provide an ontology for ACPS as a basis for autonomy-explicit MBSE activities. The ontology would provide a characterisation of autonomy in the context of ACPS for MBSE purposes. This means that on top of characterisation for autonomy, the ontology would include and draw the relationship with other concepts required in the MBSE life-cycle, such as Requirements Engineering (RE), Systems Engineering (SE), Architectural Framework (AF), and Modelling and Simulation (M&S). This objective is principally addressed in Chapter 4.

Objective 2. To provide a framework to support architecting of autonomy-explicit ACPS. The framework shall be built upon the ACPS ontology proposed in Objective 1. Autonomy-explicit approach for the framework means that the autonomy property of ACPS can be identified, reasoned, and evaluated during the architecting process from requirements, design, verification, and validation. This objective is mainly addressed in

Chapter 5.

Objective 3. Demonstrate methods that allow external ACPS techniques to be used in the MBSE design process for ACPS. By external, we mean any other engineering methods already used in relation to ACPS or their constituent elements. Examples of these techniques are explored in Section 2.3.2. As some of these techniques provide great insight into addressing ACPS challenges, it is worth investigating how they can be integrated into the MBSE processes. In this thesis, we consider external techniques for verification and validation of the ACPS that address the ACPS challenges mentioned in the introduction of this thesis. This objective is mainly addressed in Chapter 6 in the case studies.

1.3 Thesis Outline

The remainder of this thesis is organised as follows. Chapter 2 presents an overview of research relevant to this work and the tools that will be used. Chapter 3 presents the scope and methodology of our research. Chapter 4 presents the MBSE ontology for ACPS following the autonomy-explicit approach. Chapter 5 presents an Architecture Framework for ACPS (ACPSAF) using the ontology defined in Chapter 4. Chapter 6 presents two ACPS case studies designed with the framework defined in Chapter 5. The case studies serve as a validation of the ontology and framework developed in the thesis. They also demonstrate how techniques for dependable ACPS can be used in conjunction with ACPSAF. Finally, concluding remarks are accompanied by a discussion of possible further work to support the contributions of this thesis in Chapter 7. Figure 1.1 illustrates the overview of the thesis structure using a SysML Block Definition Diagram. A primer to several basic SysML notations is provided in Section 2.4.2.1.

The work presented in this thesis builds on two publications that focused on co-modelling as an approach to support the collaborative design of CPSs [36] and Architectural Framework (AF) as a mean to support the design of self-* capabilities in ACPSs [37].

In [36], we investigate the application of a co-modelling approach for CPSs. Rather than developing a hybrid modelling notation, the co-modelling approach allows exploration of both cyber and physical aspects of a system in their respective design spaces (CT for former and DE for latter). The co-modelling of an adaptive smart-grid was presented to illustrate the co-modelling approach. Simulation of the smart grid is performed in the presence of both physical and cyber faults, where each fault can be modelled from its respective design space. The co-modelling approach is adopted in this thesis as a method to support the verification and validation of the ACPS model. The smart grid model is also used as the basis to design self-healing smart grids presented in Chapter 6.

In [37], we explore the use of an Architectural Framework (MetaSelf [38]) as means to support the acquisition, design, and verification of self-* mechanisms in ACPS. The term ‘Self-*’ refers to the collection of autonomy

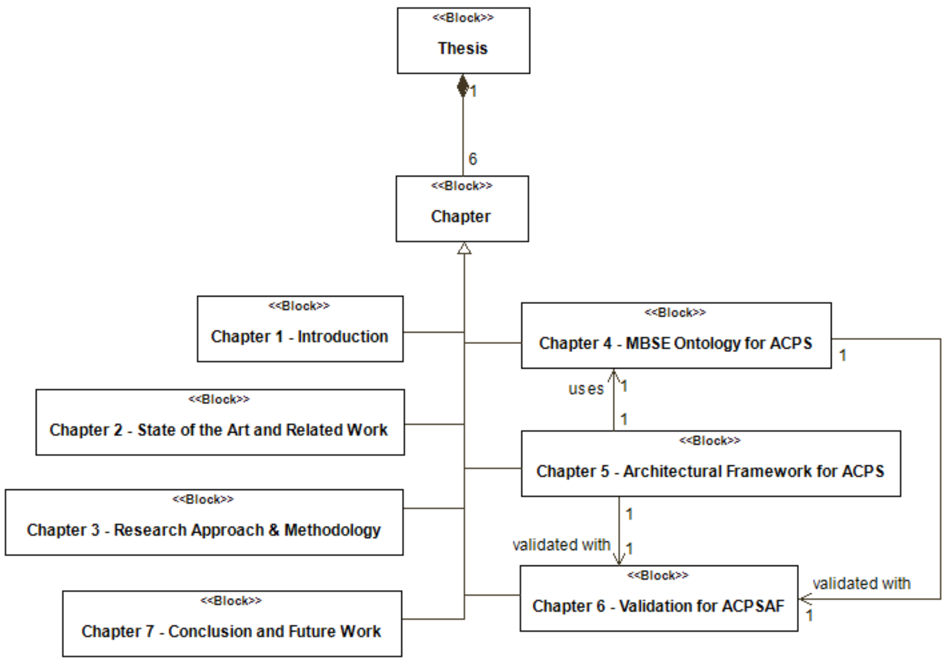

<<Block>>
Thesis
1
6
<<Block>>
Chapter
<<Block>>
Chapter 1 - Introduction
<<Block>>
Chapter 2 - State of the Art and Related Work
<<Block>>
Chapter 3 - Research Approach & Methodology
<<Block>>
Chapter 7 - Conclusion and Future Work
<<Block>>
Chapter 4 - MBSE Ontology for ACPS
1
uses
1
1
<<Block>>
Chapter 5 - Architectural Framework for ACPS
1
validated with
1
<<Block>>
Chapter 6 - Validation for ACPSAF
validated with
1

Figure 1.1: Overview of the structure and content of the thesis

mechanisms such as self-adaptive, self-organising, self-healing, and self-optimising. A feasibility study based on a self-adaptive swarm of UAVs was modelled and simulated with the INTO-CPS tool chain. Functional Mock-up Unit (FMU) based on the MetaSelf architecture is integrated with the ACPS co-model as means to achieve self-* behaviour. An evaluation of the study led to suggestions for further proof-of-concept studies with increasingly challenging ACPS architectures. The AF presented in Chapter 5 includes a perspective to support the integration of co-modelling and simulation technology that is influenced by the work done in this paper. The swarm model is also used as a base for a more complex case study carried out in Chapter 6.

State of the Art and Related Work 2

This chapter introduces and explores the published state of the art in areas relevant to the research challenge. The materials covered here form the basis of our research question, methodology, and method solutions. In Section 2.1, we elaborate the method for conducting our literature review. Section 2.2 provides a brief overview on CPSs and Section 2.3 looks into the state of the art of ACPSs, its applications, challenges, and characterisation of autonomy. Sections 2.4 and 2.5 introduce MBSE and co-simulation as a method to support the engineering of ACPS. It is then closed with a summary in Section 2.6.

2.1 Literature Review Method

Mallett et al. [39] suggest that systematic literature reviews that are grounded on the principles of rigour, transparency, and replicability can improve on the quality of traditional literature reviews. Although first applied in the medical sciences, systematic literature reviews have been increasingly used in other fields as well.

Our approach to systematic literature reviews will be based on works devoted to the application of systematic literature reviews in the domain of computer science [40, 41]. We also analyse systematic literature review papers from related domains and adapt them to our methodology. Mallett el al. [39] concludes that rather than following a rigid systematic review methodology, it is more useful to involve a mixture of compliance and flexibility, with the focus on improving the quality of the overall findings. Related systematic literature review papers that we analyse and learn from include domains such as Cyber-Physical Systems [42], Self-Adaptation for Cyber-Physical Systems [43], Systems-of-Systems [44], and requirements engineering [45].

Mallett et al. [39] note that systematic literature research typically follows a formal procedure, with well-defined protocols often peer-reviewed and piloted. While this is important and necessary for some disciplines (e.g., medical research) or types of study (e.g., surveys), this formal process is expensive for a computer science project in which systematic literature reviews are not the main research contribution. While systematic literature reviews can be taken as an exclusive method in a research, we view systematic literature reviews as an addition to our non-formal literature review.

With this intention in mind, the protocol for systematic literature reviews in this thesis will not be as formal (i.e., a lack of peer-review) as one would typically find for systematic literature reviews. However, we believe that this is justifiable for the purpose and context of our research, and we argue that the research would still benefit from this approach as opposed to relying solely on non-formal literature reviews. The protocol that we follow for the systematic literature review is:

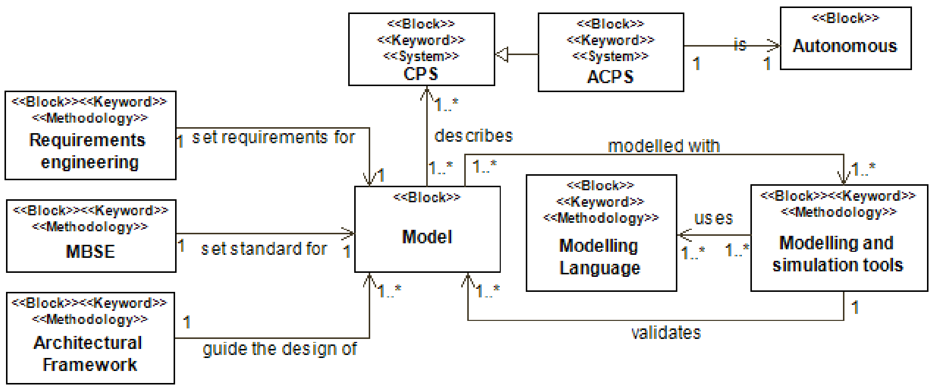
1. Defining keywords and search strings. Keywords are chosen based on the research question. The search strings are then composed by concatenating the keywords using the AND/OR operator following the method proposed by [41].
2. Defining search engine and search refinement. Search engines are selected and elaborated if any changes are made to the search engine settings to enhance the search results.
3. Defining inclusion and exclusion criteria. Define whether all the search results will be included (e.g., is there a limit?) and whether any results will be excluded (e.g., publication is too old).
4. Selection of papers and data extraction. The papers selected from the search results after passing the inclusion and exclusion criteria are listed. This also includes where the papers are cited in the thesis.

When systematic literature reviews are used as a method to answer a research question, the final step of the protocol would include an extraction of answers related to the research questions. This is not included in our protocol as the research question posed in this research cannot be addressed with systematic literature reviews. Rather, the systematic literature reviews are used to support the review of the state of the art. As a result, the final stage of our protocol would include information about where the papers are cited in the thesis and show how each paper contributes to our research.

2.1.1 Defining Keywords and Search Strings

Keywords for the search string are based on the research question. In this section, we extract those keywords so that it may be used to construct the search string. Figure 2.1 highlights the keywords that are relevant to our research question and summarises the relationship between each keyword with a SysML Block Definition Diagram.

Most of the blocks in the diagram have a «Keyword» stereotype associated with them, highlighting those blocks to be a keyword. However, there are two blocks (i.e., Autonomous and Model) that are not keywords as they would make a keyword that is too ambiguous in the sense that the keyword on its own is well recognised in other disciplines distinct from our research domain. We also include two additional stereotypes in the diagram. The «System» stereotype is applied to blocks that represent a technology at the system level, and



<<Block>> <<Keyword>> <<System>> CPS
<<Block>> <<Keyword>> <<System>> ACPS
<<Block>> Autonomous
is
1
1
<<Block>><<Keyword>> <<Methodology>> Requirements engineering
set requirements for
1
1
1..*
describes
1..*
1..*
modelled with
1..*
<<Block>> Model
<<Block>> <<Keyword>> <<Methodology>> Modelling Language
uses
1..*
1..*
<<Block>><<Keyword>> <<Methodology>> Modelling and simulation tools
<<Block>><<Keyword>> <<Methodology>> MBSE
set standard for
1
1
<<Block>><<Keyword>> <<Methodology>> Architectural Framework
1
guide the design of
1..*
1..*
validates
1

Figure 2.1: Related keywords

«Methodology» is applied to blocks that loosely represent methods for certain activities.

These stereotypes were introduced because the search string for system type keywords might be composed differently from methodology type keywords. For system type keywords, we are interested in the general state of the art of the technology so that we can build the search string from system type keywords by combining the keywords with some generic descriptive words such as “design” and “challenges”. There are three types of keywords. K_s is the sequence of keywords of the system type, K_m is the sequence of keywords of the methodology type, and R is the sequence of descriptive words.

K_s = [“Cyber Physical System”,“Autonomous Cyber Physical System”]

K_m = [“Model Based System Engineering”, “Requirement Engineering”, “Modelling”, “Simulation”, “Architectural Framework”, “Modelling language”, “Dependability”]

R= [“Design”,“Model”,“Challenge”]

The first set of search strings based on system keywords can be generated with the following rule:

$\forall x \in inds\ K_s \cdot S_x = K_s(x) \cup R(1) \cap K_s(x) \cup R(2) \cap K_s(x) \cup R(3)$

The method above would give us the first set of search strings shown in Table 2.1.

Key	**Search String**
S_1	“Cyber Physical System Design OR Cyber Physical System Model OR Cyber Physical System Challenge”
S_2	“Autonomous Cyber Physical System Design OR Autonomous Cyber Physical System Model OR Autonomous Cyber Physical System Challenge”

Table 2.1: Search strings for system related literature review

For methodology type keywords, they will be paired with the types of systems that we are interested in (i.e., CPS and ACPS). The second set of search strings based on methodology keywords can be generated with the following rule:

$\forall x \in inds\, K_m \cdot S_{(x+2)} = K_m(x) \cup K_s(1) \cap K_m(x) \cup K_s(2)$

Search strings based on methodology keywords are shown in Table 2.2. 2.1.

Key	Search String
S_3	"Model Based System Engineering Cyber Physical System OR Model Based System Engineering Autonomous Cyber Physical System"
S_4	"Requirement Engineering Cyber Physical System OR Requirement Engineering Autonomous Cyber Physical System"
S_5	"Modelling Cyber Physical System OR Modelling Autonomous Cyber Physical System"
S_6	"Simulation Cyber Physical System OR Simulation Autonomous Cyber Physical System"
S_7	"Architectural Framework Cyber Physical System OR Architectural Framework Autonomous Cyber Physical System"
S_8	"Modelling language Cyber Physical System OR Modelling language Autonomous Cyber Physical System"
S_9	"Dependability Cyber Physical System OR Dependability Autonomous Cyber Physical System"

Table 2.2: Search string for method related literature review

2.1.2 Defining Search Engine and Search Refinement

The search engines used are IEEE Xplore[1], ScienceDirect[2], ACM [3], and Scopus [4]. Search will be refined to only include peer-reviewed work such as proceedings, books, and journals. Although further changes could be made to the search engine tools to refine the search results (e.g., changing the sorting criteria), we use the default search engine settings as the tools are optimised to return the most relevant results based on the search string given. Aside from the differences in the publication databases, each search engine is likely to operate under different algorithms as well. Therefore, future attempts at replicating this literature review might result in a different set of search results.

Our method would include the use of each search string in all four search engines. The top/first search results from each search engine will be considered. If some initial results are excluded (see exclusion criteria in the next section), we would take the next item returned from the search result.

2.1.3 Define Inclusion and Exclusion Criteria

With all the results returned by the search engine, selecting and extracting data from papers can be an overwhelming task. We only intend to carry out a small-scale systematic literature review, and our inclusion criteria will include at least 3 distinct search results from each keyword.

[1]IEEE Xplore Digital Library, https://ieeexplore.ieee.org/Xplore/home.jsp
[2]Science Direct, https://www.sciencedirect.com/
[3]ACM Digital Library, https://dl.acm.org/
[4]Scopus, https://www.scopus.com/

As an exclusion criterion, we exclude papers that are outside the domain of our research interest. This type of result might be possible if the search string contains keywords that are shared in other domains. We also exclude abstract papers that do not contain the full research contribution.

2.1.4 Selection of Papers and Data Extraction

Based on the search engines selected and the included criteria, the exact steps that are used to gather the papers are as follows:

1. Run the search string in the IEEE Xplore, ScienceDirect, ACM, and Scopus search engines.
2. Get the top result from IEEE Xplore, and check against the exclusion criteria. If the result meets the exclusion criteria (e.g., not a peer-reviewed paper), ignore this paper and repeat this step using the next paper from the result. If the result does not meet the exclusion criteria, the paper is selected for the search string.
3. Get the top result from ScienceDirect and check against the exclusion criteria and the paper selected from the search string. If the result meets the exclusion criteria or is found in the same paper selected from the previous step, ignore this paper and use the next paper from the result instead. Repeat this step until a paper is selected.
4. Repeat the steps described in step 3. However, apply it in the context of the ACM search engine.
5. Repeat the steps described in step 3. However, apply it in the context of the Scopus search engine.
6. Repeat the process from step 1 using the next search string until all search strings are included.

The results of this process are captured in Table 2.3. It shows all the papers selected for each search string and where it is cited throughout this thesis. We also categorise the papers in terms of relevance towards our work:

- **Research question.** These papers are relevant to our research question and form our research question and objectives to a certain degree.
- **Tool choice.** These papers concern the choice of tools and methods in building our research contribution.
- **Future work.** These papers provide insight into research challenges that are relevant to our research question but are unable to be covered within the scope of our research.
- **Supplementary.** These papers are within the domain but have different focus and possibly different levels of abstraction. These papers are referenced for context or interest.
- **Background.** These papers are used to provide background context of the literature, such as context definition, history, and the state of the art.

Table 2.3: Paper selection from systematic literature review

Search String	Selection	In text Cite	Relevance
S1	EDA for cyber-physical systems [46]	Section 6.2.2	Future works
S1	Design optimization of WirelessHART networks in Cyber-Physical Systems [47]	Section 4.6	Supplementary
S1	Teaching cyber-physical systems: a programming approach [48]	Section 2.5	Research question, Tool choice
S1	From embedded to cyber-physical systems: Challenges and future directions [49]	Section 2.5	Background, Tool choice
S2	A holonic-based method for design process of cyber-physical reconfigurable systems [50]	Section 2.4	Research question
S2	High Level Design of a Home Autonomous System Based on Cyber Physical System Modeling [51]	Section 2.3	Future works
S2	Experience report: constraint-based modeling of autonomous vehicle trajectories [52]	Section 2.3	Research question, Tool choice
S2	Modeling cps swarms: An automotive use case [53]	Section 2.4	Tool choice
S3	State of the art of cyber-physical systems security: An automatic control perspective [54]	Section 7.3	Supplementary
S3	An effective framework to simulate the cyber-physical systems with application to the building and energy saving [55]	Section 2.5	Tool choice
S3	Enabling Model Testing of Cyber-Physical Systems [56]	Section 2.5	Research question, Tool choice
S3	The role of models in engineering of cyber-physical systems-challenges and possibilities [57]	Section 2.4.1	Research question
S4	Cyber-physical systems and their security issues [58]	Section 7.3	Supplementary
S4	Verifying Cyber-Physical System Behavior in the Context of Cyber-Physical System-Networks [59]	Section 4.6	Future works

Continued on next page

Table 2.3 – continued from previous page

Search String	Selection	In text Cite	Relevance
S4	TORUS: Tracing Complex Requirements for Large Cyber-Physical [60]	Section 7.3	Future works
S4	Goal-oriented co-engineering of security and safety requirements in cyber-physical systems [61]	Section 2.6	Supplement
S5	Physical modeling of material flows in cyber-physical production systems [62]	Section 2.5	Tool choice
S5	A state-prediction-based control strategy for UAVs in Cyber-Physical Systems [63]	Section 6.3.1.2	Supplementary
S5	Hybrid simulation for cyber physical systems: a panel on where are we going regarding complexity, intelligence, and adaptability of CPS using simulation [64]	Section 2.5	Tool choice
S5	Complexity challenges in cyber physical systems: Using modeling and simulation (M&S) to support intelligence, adaptation and autonomy [65]	Section 2.3.1	Supplementary
S6	Integrated Cyber Physical Simulation Modelling Environment for Manufacturing 4.0 [66]	Section 7.3	Research question
S6	Robustness assessment of cyber-physical systems with weak interdependency [67]	Section 4.6	Future works
S6	A Virtual Playground for Testing Smart Cyber-Physical Systems [68]	Section 2.3.2	Tool choice
S6	A scene-driven modeling reconfigurable hardware-in-loop simulation environment for the verification of an autonomous CPS [21]	Section 1.1	Research question
S7	Classification of cyber-physical production systems applications: Proposition of an analysis framework [69]	Section 2.3.1	Supplementary
S7	Viewpoints and views for the architecture description of cyber-physical manufacturing systems [70]	Section 5.11	Research question

Continued on next page

Table 2.3 – continued from previous page

Search String	Selection	In text Cite	Relevance
S7	An architecture framework for experimentations with self-adaptive cyber-physical systems [71]	Section 5.11	Research question
S7	Towards integration of modeling methods for cyber-physical systems [72]	Section 2.5	Research question
S8	Model-based documentation of dynamicity constraints for collaborative cyber-physical system architectures: Findings from an industrial case study [73]	Section 2.4	Future works
S8	Integration-Oriented Modeling of Cyber-physical Interactive Process [74]	Section 2.4	Research question, Tool choice
S8	Toward a unified object model for cyber-physical systems [75]	Section 2.5	Research question
S8	A safe autonomous vehicle trajectory domain specific modeling language for non-expert development [76]	Section 2.4.2	Supplementary
S9	Some Considerations on Dependability Issues and Cyber-Security of Cyber-Physical Systems [77]	Section 2.2.3 -	Supplementary
S9	An approach to model dependability of cyber-physical systems [78]	Section 2.2.3 -	Background
S9	Holistic Cyber-Physical Management for Dependable Wireless Control Systems [79]	Section 2.2.3 -	Background
S9	Enhancing dependability and security of cyber-physical production systems [80]	Section 2.2.3 -	Supplementary

2.1.5 Discussion

One advantage of systematic literature reviews is that it ensures a wide range of literature is being considered in the research, preventing the researcher from only using cherry-picked literature. We can observe this benefit here as the literature selected from the review is being referenced throughout different places in the thesis, from providing the basis of the research question to the suggestion for future works.

We also observe the potential downfall of systematic literature reviews, especially when conducted on a small set of literature. For example, one of the search keywords is “Model Based Systems Engineering”, which forms part of the search string in S_3. For that search string, we were expecting highly reputable sources of MBSE literature, such as publications by INCOSE, to be within the top results on the Scopus search engine. However, this was not the case at all for S_3. Instead, many of the top results for S_3 on Scopus were based on CPS modelling in general, rather than MBSE.

One potential issue would be the keywords used as the formation of the search string. For example, the search string “Model Based Systems Engineering Cyber Physical System” could score highly on publications with keywords of “Model” as it is more common than “Model Based Systems Engineering” as a compound keyword. However, this might not have been the problem in our case. We have attempted to redo the S_3 search string by replacing “Model Based Systems Engineering” with “MBSE”. Although we received a slightly better and more relevant result in respect to MBSE, there was still no result from INCOSE.

There are many possibilities to how this could have happened. It may be possible that INCOSE does not have any publication that is relevant to the “Model Based Systems Engineering Cyber Physical System”, but that is not true for this case[81]. It could be possible that publications with cyber physical systems as a keyword are more relevant to MBSE due to factors like number of publications and/or citation. There are also other possibilities such as publications not being present in the search engine or not being properly tagged with the keyword.

Although we are not going to investigate deeper into the weakness of systematic literature reviews, we tend to agree with the findings of Mallett et al. [39] that systematic literature reviews have its challenges and are probably best used not as a rigid methodology, but with some flexibility and focus on the overall quality. To a certain degree, this result validates our reasoning to consider systematic literature reviews as a supplement to our literature review rather than as an exclusive method as mentioned in the earlier part of section 2.1. Many publications, such as those from INCOSE, will be referenced in the thesis as they are deemed relevant to the research despite not appearing from the systematic literature review (.e.,g [19, 82, 83, 84, 85, 8])

2.2 Cyber-Physical Systems

In this section, we explore the contextual definition of Cyber-Physical Systems (CPSs) and the fields in which CPSs have been used. The definitions explored here will provide necessary context as we set out to define the ontology for the proposed ACPS framework later in Section 4.5.

The increase in integration of Information Technology (IT) into physical systems has led to the introduction of the term Cyber-Physical Systems (CPS) that refers to systems in which the computational and physical elements are tightly conjoined [2]. The coining of the term ‘cyber-physical systems’ [86] is often attributed to Helen Gill of the US National Science Foundation (NSF) where consideration of both cyber and physical elements are essential to describing the overall performance of the system [87]. Early definitions described CPSs as “physical, biological, and engineered systems whose operations are integrated, monitored, and/or controlled by a computational core... The computational core is an embedded system, usually demands real-time response, and is most often distributed” [88].

Lee et al. [89], distinguish CPSs from other disciplines by the requirement to understand the joint dynamics of computers, software, networks, and physical processes in order to design them. They state that “as an intellectual challenge, CPS is about the intersection, not the union, of physical and cyber”, and comment that “it is not sufficient to separately understand the physical components and computational components”.

An Integrated Research Agenda by the German National Academy of Sciences and Engineering provided a detailed definition of CPS, first published in 2011 [90]. Their definition describes a CPS as a system with embedded software, which:

- uses sensors and actuators to identify and influence changes in physical processes.
- records and evaluates data, and actively or reactively interacts with the physical and digital world.
- utilises networks to communicate with other CPSs.
- utilises globally available services and data.
- offers a number of human-machine interfaces.

Within a CPS, the embedded computing element monitors, coordinates, controls, and integrates physical processes [91, 92], where such processes and computation interact via feedback loops [2]. CPSs are more than simply networked embedded systems; they are computationally complex, intelligent systems with the capacity to collaborate with other systems, adapt to their environment, and evolve over time. The emergence of CPSs concedes a paradigm shift from traditional embedded systems, where computational ability is not just added to some physical system, but the functionality of the system as a whole relies on the tight integration of cyber and physical.

Advances in the application of CPSs can be found across a wide range of domains. In health care, CPSs have been used in automated vital sign reading [93], health care monitoring and decision support systems [94], assertive systems for intensive care [95], and more [96]. The application of CPSs in industrial settings has been referred to as ‘Industrial CPS’ (ICPS) [97, 98] with a body of interest in realising the idea of a smart factory [99, 100, 101]. The application of CPS can also be seen in the domain of infrastructure, such as the water system [102, 103, 104], roadways and traffic system [105, 106, 107], environment monitoring [108, 109, 110], emergency management systems [111, 112, 113] and power grids [114, 115, 116]. Many recent advances have also been made in automotive systems [117, 118], aviation [119], aerospace [120, 121], railway systems [122, 123], and marine systems [124, 125], with focus on its consideration as a CPS.

Given the wide spectrum of its application, the engineering of CPSs requires a multidisciplinary body of knowledge and engineering capabilities [126, 127]. A survey on the competency and qualifications required of CPS engineers in both industrial and academic settings showed that disciplines from computer science, embedded systems, discrete and continuous mathematics, control system, modelling of heterogeneous and dynamic systems, and the CPS life-cycle are being identified as 'CPS Foundations' [4], where competency in the mentioned disciplines is treated as the most important aspect for CPS engineers. On top of those functional factors, experience in non-functional characteristics such as security, interoperability, and safety and human factors also contribute to the competency assessment of CPS engineers. As a result, techniques from a discipline such as Systems Engineering (SE) that is based on an interdisciplinary field of engineering to manage complex systems has been utilised in the engineering of CPSs [5, 6, 7]. The role of SE approaches in the engineering of CPSs is discussed further in Section 2.4.

2.2.1 Related Technology

Although there is a general understanding of the CPS concept, there is no standardised formal definition used globally. Organisations from different domains, industries, and countries may refer to a slightly different notion of system when they use the term "CPS". This can lead to confusion over the terms being used, especially when there are other related technologies being used interchangeably. To better understand the context of CPS engineering, this section explores several CPS-related technologies.

Embedded system As previously mentioned, there is a fundamental link between CPSs and embedded systems. Although Lee and Seshia [89] emphasise that CPSs are about the intersection between cyber and physical instead of the union, it is unclear how this differs from embedded systems at the practical level. Embedded systems are commonly regarded as information processing systems that are embedded into an enclosing physical product [128]. On the practical level, embedded systems revolve around embedded computers acting as the cyber part to interact with physical processes. Typical examples of embedded systems include washing machines, dishwashers, printers, and digital cameras.

Following this train of thought, Marwedel [129] states that instead of its internal properties, CPSs differ from embedded systems through the challenges they are trying to address. Embedded systems focus on software integration, where the technical issue lies in the management of time and concurrency of the computer system [130]. Meanwhile, CPSs emphasise the integration challenges of physical qualities such as space, time, and energy. Marwedel [129] concludes that CPSs encompass most embedded systems and that the two terms could be used interchangeably, only referring to CPS whenever there is an emphasis on the physical aspects.

Another perspective on the relationship between embedded systems and CPSs is the involvement of networked communication. Traditional embedded systems are often stand-alone units; However, CPSs are networks of ICT architectures [131]. A CPS such as a smart grid [132] needs to distribute its system components across a wide geographical area. Smart meters can serve as sensors to monitor real-time data, but this needs to be communicated to the central system to enable the CPS. Geographical constraint is not the only reason contributing to communication challenges. Some CPSs are a collection of independent systems, usually referred to as the Internet-of-Things (IoT) or System-of-Systems (SoS). A Smart Home CPS might consist of several embedded devices (i.e. mobile phones, house lighting, heater) forming a networked infrastructure. These devices can be dynamic and heterogeneous, posing communication and control challenges not found in the traditional embedded system.

However, we consider the biggest difference between CPS and embedded systems the fact that CPS uses the architecture of closed-loop control feedback to achieve its tight cyber and physical integration loop, while an embedded system can be either an open or closed-loop controlled system [133]. For example, a drying machine does not have a sensor to detect the state (e.g., the dryness) of the clothes in the system. It simply takes a time duration or operation mode as an input, and executes it without any influence from the state of the environment. So although CPS is a type of embedded system (since CPSs must have an embedded computer as the controller), our work on autonomous systems will not be applicable for all embedded systems, especially those with open-loop control as the system would not be able to make any autonomous decision meaningfully without knowing the state of the plant and/or environment.

Internet of Things (IoT) Both IoT and CPS have been gaining a lot of attention in the last decade. Although the phrases have distinct origins [134], it appears that they have overlapping definitions, with both involving the integration of computation, networks, and physical systems [135, 136]. The relationship between CPS and IoT has been uncertain as the terms are occasionally used interchangeably [137] but each have their own respective communities. Comparing the relationship between CPS and IoT is not an easy task due to the lack of globally accepted definitions and their continually evolving nature (i.e., what could constitute as “IoT” could change very slightly over time) [138].

Depending on which definition is taken when comparing CPS and IoT, they could be regarded as mutually interchangeable, partially overlapping, CPS as a subset of IoT, or IoT as a subset of CPS [139]. A study comparing and contrasting literature in CPS and IoT argues that a distinction between the two concepts lies in the issues around control, platform, internet, and human interactions [139]. While some consider that the infrastructure of IoT has to include the internet, Mattern and Floerkemeier [140] state that it is possible to consider the word ‘Internet’ in the IoT as a metaphor rather than the more restricted sense of the technical IP protocol stack.

Due to the difficulty in forming a formal definition that makes a clear distinction between the two terms, Greer et al. [139] suggest that there should be a unified CPS/IoT perspective instead as it would provide the opportunity for CPS and IoT research communities to work together in addressing common challenges. However, similarly as addressed in the previous section regarding embedded systems, our work on autonomous architecture will rely on the closed-loop feedback which is already the standard architecture for CPS. However, there are examples of systems that do rely on closed-loop feedback that are still considered IoT. For example, a smart home with multiple internet connected devices such as light bulbs, TV, and door lock that are controllable from a phone via the internet is often considered as IoT. However, the lack of sensors for any form of closed-loop feedback separates IoT from CPS for the purpose of our work. This means that although the term IoT and CPS can be found being used interchangeably in the literature, our work refers to CPS that must be based on closed-loop feedback. Therefore, methods developed here for ACPS may not be applicable to IoT.

2.2.2 Architecture

As discussed in the previous section, one observation in the technical architecture of CPSs is that it uses closed-loop control feedback to enable its tight cyber and physical integration loop. This is the opposite of an open-loop control system, where the input of the system is independent from the output of the system (i.e., without feedback loop). From the perspective of control systems, CPS can be seen as being made of one or more control systems. Where a basic control system is described as having an input, control unit, and output, CPS may be made of multiple inputs, control units and outputs in complex configurations.

The inputs in CPS almost always include at least one physical sensor, but it may also receive some form of cyber (i.e., computer/communication signal) as input. The control unit is typically an embedded computer, but this may just be a computer in larger systems. The output of CPS will include at least one actuator, but similarly to the input, it may also include an output communication signal (i.e., to other CPSs). For example, an Unmanned Aerial Vehicle (i.e., drone) can be considered a CPS with multiple sensor input (e.g., accelerometer, inertial measurement units, and GPS), a control unit (the on-board embedded computing), and multiple actuators output (i.e., rotors).

The UAVs involve a certain level of complexity that is not found in systems such as a room heating unit, which is considered an embedded system. The correlation and interaction between the thermometer input reading and the heating unit output is relatively straightforward. However, in the UAV, it is not sufficient to only know about the number and type of the rotors. The position, orientation, power, weight, and responsiveness between all rotors are absolutely critical in order to design a functioning UAV. It is this type of complexity that differs a CPS from an embedded system. CPSs can also be part of a larger CPS. For example, UAVs can collaborate and operate together as a part of Swarm UAVs [141].

The characteristic of closed-loop control architecture in CPS is important in our research as our work for autonomy in CPS will be based on existing theory for autonomous systems, which is based on closed-loop control architecture [142]. The next section will cover the concept of autonomy in the context of CPSs.

2.2.3 Dependability

The concept of dependability is one that is often found in CPS literature [77, 78, 79, 80] and will also be relevant in our research. This section aims to provide a short overview on the topic of dependability, focusing on its definition and threats to dependability. Our basis for the concept of dependability follows the work done by Avizienis et al. [143], which defines dependability of a system as "the ability to avoid service failures that are more frequent and more severe than is acceptable". The concept of dependability encompasses several attributes:

- **availability**: readiness for correct service.
- **reliability**: continuity of correct service.
- **safety**: absence of catastrophic consequences on the user(s) and the environment.
- **integrity**: absence of improper system alterations.
- **maintainability**: ability to undergo modifications and repairs.

To understand these concepts better, we need to further define the many keywords being used here such as system, environment, service, and failures. Avizienis et al. [143] define system as "an entity that interacts with other entities, i.e., other systems, including hardware, software, humans, and the physical world with its natural phenomena". A *system* in itself can be decomposed into a set of interacting entities, where each entity can be considered a *subsystem* (each subsystem is also considered a system in its own right). The other systems that the system interacts with are the *environment* of the given system. The *system boundary* is the common frontier between the system and its environment, or as defined by Checkland [144], "a distinction made by an observer which marks the difference between an entity he takes to be a system and its environment".

Service is the behaviour of a system as perceived by its user(s) and service failure (or just *failure*) is an event that occurs when the delivered service deviates from the correct service. This deviation is called an *error*, and the hypothesised cause of an error is called a *fault*. It is important that an error is just a part of the total state of the system, and it may or may not lead to service failure. The means to attain dependability can be grouped into four categories:

- **Fault prevention** means to prevent the occurrence or introduction of faults.
- **Fault tolerance** means to avoid service failure in the presence of faults.
- **Fault removal** means to reduce the number and severity of faults.
- **Fault forecasting** means to estimate the present number, the future incidence, and the likely consequences of faults.

One or more of these types of dependability can often be found in ACPS. For example, one of the case studies in Chapter 6 will be based on a "self-healing" grid where grid configurations are manipulated in real-time to prevent or reduce service failure, which is a form of fault tolerance. When faults lead to the failure of some but not all services, the system can be said to suffer partial failure and is operating in a degraded mode.

One method to testing dependability of a system is through fault modelling. A fault model is an engineering model of something that could go wrong in the construction or operation of a piece of equipment, which can be used to predict the consequences of a given fault [145, 146]. In our case study, we would be introducing a fault model into the system to test, for example, the fault tolerance mechanism of the self-healing grid in the presence of fault.

2.3 Autonomous CPSs

The concept of autonomy can be found in many disciplines such as philosophy, biology, and linguistics. In the context of abstract systems, the Oxford English Dictionary defines *autonomous* as a system that is "independent of, or not subject to, external influences or controls"[5]. The word autonomy comes from the Greek word *autos* that means 'self', and *nomos* that means 'law'. For a more explicit definition, an autonomous system contains its own governing rules and is capable of performing tasks without explicit external (e.g., human) control [147].

Autonomy in the social context typically focuses on the topic of 'permission' for governance. For example, an autonomous area in a country has freedom from an external authority. The freedom here could mean freedom from laws and jurisdictions of the country in which the autonomous area resides. The autonomy in this context does not refer to the 'ability' to perform certain actions. Non-autonomous areas could have the ability to behave exactly as the autonomous areas. However, the non-autonomous area will have to deal with the repercussions that come from law-breaking behaviours. In this context, the autonomous area has autonomy not because it has the 'ability' to perform certain actions, but because it is given the 'permission' to behave without consequences.

In autonomous systems, the concern is typically less about permission and more about the ability to perform certain tasks autonomously. For example, the primary concern in the design of an autonomous vehicle is its level of competence in performing driving behaviour traditionally performed by a human operator. As the application of CPSs become more common, the ability to perform complex tasks autonomously without human intervention will become an integral part of future CPSs. In the literature, there are several factors that contribute to the desirability of Autonomous CPS (ACPS).

One of the principal benefits of ACPS is the potential to be more cost-effective than systems reliant on human operators. The role of automation and its impact can be observed from the industrial revolution. The advances in CPSs, along with other technologies such as cloud computing, are contributing to a new

[5]"autonomous, adj.". OED Online. December 2019. Oxford University Press. https://www.oed.com/view/Entry/13498?redirectedFrom=autonomous (accessed December 12, 2019).

era of technology termed “Industry 4.0” [148, 131, 149, 150] for its potential to disrupt the conventional approaches across domains, and autonomy is integral to this movement. Application of ACPSs can be seen from the manufacturing level, such as autonomous production lines [151, 152, 153] and consumer products such as autonomous vacuum cleaners [154, 155].

Some applications of ACPSs also enable functionalities that would otherwise be too dangerous for human operators, with typical examples in military missions [156, 157]. There is also the premise that with Artificial Intelligence (AI), ACPS might perform better and be more reliable than systems with human operators. Accomplishments in AI such as AlphaGo [158] that were able to beat some of the best human players at Go, hint that AI has the potential to outperform humans in complex operations. However, there is currently no proof that this has been achieved for complex ACPS, where the list of possible actions, permitted actions, and end goals are typically not as easily and clearly defined. For example, Robert [159] states that it is difficult to determine whether Autonomous Vehicles (AV) can be safer than manually driven cars as the literature in the measurement of AV safety and definition of AV is still not well established.

In this thesis, we will use the term ACPS separately from the term ‘autonomous system’. ACPS is a subset of autonomous systems and is only concerned with autonomous systems that can be categorised as CPS. Literature in autonomous systems, such as autonomous robots and autonomous vehicles, would be relevant to our research as autonomous robots and autonomous vehicles can be considered as ACPS. However, there are also fields such as Autonomic Computing [160] that are often considered to be a sub-field of autonomous systems. However, they would not be considered as ACPS as they lack the physical aspects of a CPS. Our interest in ACPS is based on the ‘CPS-first’ approach. This means that we look into CPSs that evolve into ACPSs by introducing autonomy features into an existing CPS, as opposed to working from an autonomous system and transforming it into an ACPS with CPS features. For the purpose of our work, we define ACPSs as CPSs that have the capability to control their behaviour and/or structure in reaction to internal or external changes to meet a defined system goal with limited or no external (e.g., human) intervention.

2.3.1 ACPS across domains

As mentioned in Section 2.2, the application of CPS can be found across a wide range of domains. It is not a surprise that ACPS can be found across many types of domains as well. In this section, we provide a brief overview of several such application areas.

Energy

The effort to improve and modernise the traditional power grid has led to what is now known as a "Smart Grid". The Oxford dictionary defines a smart grid as "an electricity supply network that uses digital communications technology to detect and react to local changes in usage.". However, there are several views on the definition of a smart grid in the literature. Bamberger et al. [161] and Song et al. [115] do not provide an explicit definition of a smart grid and simply state that a smart grid is a response to the new challenges and opportunities in the electricity grids of the 21st century. The idea that a smart grid is an abstract and ever-changing concept can be reflected from the fact that standardisation of a smart grid is a big challenge in of itself [116]. However, some authors such as Yu et al. [114] propose a definition that includes a more concrete description of smart grid features such as real time monitoring, automated control, and two-way current flow.

Regardless of the definition, it is agreed that future smart grids will be highly integrated with distributed sensors that monitor energy transmission and consumption, giving energy providers real-time insights to provide a more reliable and efficient service. The introduction of smart meters in houses means that utility companies can better understand and predict energy consumption trends. When combined with an autonomous capability, a smart grid can provide a more robust integration with distributed power generation from renewable energy sources where electricity is not always generated consistently. Smart appliances could enable autonomous demand response in which the smart grid coordinates generators and smart appliances to reduce energy consumption spikes (e.g., schedule washing machines to operate during low energy demand periods). Intelligent energy forecasting methods involving machine learning may be incorporated with a smart grid to further improve demand-side management, load shedding, and optimum dispatch [162].

One special technique that can be used in a smart grid is called the ‘self-healing grid’, where the smart grid utilises sensors and power switches distributed across the grid to detect faults and minimise their impact by isolating the faulty area [163, 164]. The sensors are used to monitor the activity of electricity transmissions across the grid and provide a real-time notification when a fault emerges in the grid. The power switches can be reconfigured autonomously such that the generated electricity is transmitted through a different route away from the faulty area, effectively isolating the fault and minimising the number of affected grid customers.

Smart grid technology also introduces the concept of the ‘Microgrid’ that consists of a group of electricity sources and loads that could disconnect or reconnect to the grid dynamically [165, 166]. The principle behind the microgrid is to enable a distributed energy resource (DER) as it provides greater flexibility compared to the traditional centralised approach. Microgrids typically function autonomously as dictated by the physical or economic interest of the grid.

It is important to note that the specific method/technique (e.g., intelligent forecasting, self-healing, microgrid) used in a smart grid does not define the smart grid. Rather, it is the introduction of the enabling components such as distributed sensors and/or actuators that provide the foundation for the transition from a traditional grid to a smart grid. What method or technique is being utilised in a smart grid to improve the grid operation can differ from one grid to another.

Transportation

The goal of a fully autonomous transportation system would be to reliably and safely transport entities from one location to another without control from a human operator. The concept of the autonomous vehicle is not a new one and many transportation systems have been equipped with features that allow partial autonomy. The first aircraft autopilot was developed in 1912 by Sperry Corporation to allow aircrafts to fly straight and level on a compass course without a pilot's attention. A similar principle can be found applied to ships and boats with the self-steering gear, and commercial cars with cruise control. While these features do provide a certain degree of autonomy, their use is limited to only a portion of the transportation process and human operators are still required.

Current research on autonomous CPSs in transportation is heading towards fully autonomous vehicles for almost all types of transport mediums such as trains [167, 168], cars [27, 52, 65], boats [169] and UAVs [170]. Unlike other domains covered in this section, the end goal of each autonomous vehicle is often very well defined. The requirement for a successful autonomous transportation system often includes more than just reaching the destination. For example, the common expectation for autonomous vehicles would include compliance with traffic rules, safety, and security.

Infrastructure

Areas in civil infrastructure have seen a widespread use of digital devices in the form of embedded systems. It is not uncommon now to see lighting, heating, and security systems in a building being connected digitally. Infrastructure systems in the city, such as traffic and street lights, are also controlled digitally and moving towards autonomous traffic management systems [171]. The attempt to modernise these infrastructures is often referred to as the concept of making the infrastructure "Smarter", and terms such as "smart home", "smart building", and "smart city" have been used to describe the concept at its respective application levels.

The concept of smart infrastructure is not inherently an infrastructure transformation process that is limited to the discipline of CPSs. Smart infrastructure for certain applications could mean installing embedded systems to objects or systems that do not traditionally have them. For example, an initiative towards creating a smart city in Barcelona [172] includes the installation of sensors in garbage bins to allow for content monitoring and optimisation of garbage collection services [173]. Other applications such as traffic lights that traditionally operate with fixed periodic scheduling could be paired with sensors that monitor traffic flow to enable dynamic traffic light control [174] [175], effectively creating a CPS.

In a smart home [51], the focus is to connect various electric and electronic appliances to a central control system. This would in turn provide the smart home user accessibility through the Internet and automation [176]. Even without sensors, this IoT concept could provide a certain level of automation through program scheduling. When coupled with sensors, the IoT could be coupled with the CPS concept for a much more powerful autonomous operation. User behaviour could be observed with the appropriate sensors and analysed with machine learning techniques to predict future user behaviour [177] [178] for a more efficient home management system [179].

It is worth mentioning that while privacy is always a concern, the privacy issue is often amplified in this domain. When compared to ACPSs in other domains such as manufacturing, smart homes and smart cities typically rely on direct observation of the residents' behaviour, which raises significant privacy concerns [180, 181, 182].

Manufacturing
CPSs have been seen as a key part of future industrial growth [183]. Indeed, such has been their significance that Industrial CPSs (ICPSs) are seen as a "fourth industrial revolution" [184, 150]. ICPSs often utilise technologies such as Multi-Agent Systems (MAS) [185, 186] and Service-Oriented Architectures (SOA) [187, 188] to achieve distributed intelligence and self-adaptation capabilities [189]. Research in Cyber-Physical Production Systems (CPPS) [190, 69] includes an expectation that CPPSs will include the ability to be context-adaptive through autonomy features such as self-organisation, self-maintenance, and self-repair.

2.3.2 ACPS Challenges

As ACPSs are increasingly deployed in safety critical situations, there is a greater reliance on their dependability. For many types of ACPS such as autonomous vehicles, engineers need to establish a high level of confidence in the dependability of the ACPS before it can be accepted to the market. Reliance on a high level of dependability is not a new topic in systems engineering. However, what is special for ACPSs is the difficulty to establish this confidence. We take an example of autonomous vehicles to explore why this is the case. One major question for autonomous vehicles in recent years is about how much testing is required to demonstrate vehicle reliability. In this case, reliability would typically refer to the vehicle's safety in terms of fatalities and injuries. Kalra and Paddock [191] attempted to answer this question by using a statistical approach to get the number of miles that a vehicle needs to be driven to demonstrate that the failure rate is below a certain threshold. The result shows that depending on the level of confidence required, autonomous vehicles would need to be driven billions of miles to demonstrate their reliability. Needless to say, it is highly impractical, if not impossible, to achieve this number by physical testing in real life.

Testing for autonomous vehicles now relies on a combination of physical and virtual testing [68] with simulation tools [192]. Even with this approach, it is noted [191] that it is still not possible to establish full confidence in the safety of autonomous vehicles as uncertainty will remain. The causes of this uncertainty are directly related to uncertainties in the environment. The reason why autonomous vehicles need to be tested over many miles is to increase the chances of the test encountering unexpected environmental conditions that result in autonomous behaviours that compromise safety. As the vehicle is tested against more and more unique scenarios, more confidence can be put into the autonomous vehicle as well, which can be measured with specific methods like Safety Integrity Level (SIL) [193] to demonstrate reliability and achievement of failure rate thresholds.

There are alternatives to testing, such as the use of automated theorem proving [194] to provide formal verification. However, the task of developing formal verification for complex system such as CPS is often many times more challenging than testing. For example, Sanwan and Hasan [195] argue that automated theorem proving usually deals with the continuous aspects of CPS by using abstracted discrete models, which causes the designs to be prone to errors, and proposes to use higher-order-logic theorem proving for CPS. The combination with the unique characteristics brought in by autonomy means that there are a lot of relatively new and ongoing work on formal verification for ACPS, such as formal verification of ethical choices in autonomous systems [196].

Uncertainties in the environment that cause failure in ACPSs can be categorised into two types. The first type of uncertainty is one in which the ACPS sensors fail to detect the environment properly. For example, a camera sensor might fail to render an image properly when it is faced against bright skylight. If the autonomous system relies on this image to detect an obstacle, it might fail to recognise the presence of the obstacle, and as a result, crash into it. The second type of uncertainty is one in which an ACPS fails due to a poorly designed decision-making policy. Most of the literature in autonomous systems typically concerns the second type of uncertainty. Given that the behaviour of ACPS is dependent on the environment and that there are infinite possibilities in the environment, how can we gain confidence that the autonomous system will make the right adaptation choice? This question forms the basis of the first ACPS challenge:

> Challenge 1: How can we have assurance in the behaviour of ACPSs even in uncertain environments?

Challenges in ACPS do not stop at uncertainties in the environment. Some applications of ACPSs have started leveraging Machine Learning (ML) to provide capabilities that could not be achieved before. For example, Reinforcement Learning (RL) techniques could be used to enable Unmanned Aerial Vehicles (UAVs) for search and rescue missions to locate missing humans after natural disasters in unknown environments [26]. RL techniques allow UAVs to develop their own navigation plan by learning from their environment instead of an explicitly defined behaviour. UAVs with ML capability are able to learn from their environment in real time and develop a better validation strategy.

However, previous experiences in ML projects have shown that runtime learning could result in unexpected, undesirable behaviour. For example, Microsoft launched a Twitter bot - a machine learning project that engaged and communicated with people aged 18-24[6]. The bot was designed to interact with Twitter users and

[6]Jane Wakefield, ‘Microsoft chatbot is taught to swear on Twitter’, https://www.bbc.co.uk/news/technology-35890188, (Mar 24, 2016)

learn from its conversations. Within a day of its release, the bot had learned to produce inflammatory comments such that its behaviour had to be edited by Microsoft. Unlike the chatbot, unexpected behaviour arising from ML behaviour in ACPSs applications are likely to be costly and even possibly fatal. As requirements for ACPS tend to be multifaceted, ML behaviour that focuses on one requirement might cause unexpected behaviour that violates others.

> Challenge 2: How can we have assurance in the behaviour of an ACPS even when the system's behaviour changes at runtime?

Proposed Solutions
When considering non-domain-specific solutions that could be applied to all types of ACPSs, the solutions can be categorised into two different types. The first type is the more conventional approach that focuses on improving existing testing methods to enhance the verification and validation of ACPSs. The principle behind this approach is that despite the impossibility to deliver a fault-free product, it just has to be better than current solutions. For example, it is not necessary for autonomous vehicles to be perfect, but they just need to be statistically better (i.e., safer) than human drivers. This type of solution still puts its primary concern on testing, where the focus is to improve the quantity and quality of testing done for ACPSs.

Improving the quantity of testing has been briefly mentioned. Utilising virtual testing [197] with simulation tools, engineers are able to run more tests especially with techniques such as automated test case generation [198] and accelerated testing [199]. One example of work focused on the quality of test cases is that of Althoff and Lutz [200], which looked at a constructive algorithm where scenarios generated are optimised towards certain characteristics for testing collision avoidance of autonomous vehicles.

The second type of solution accepts the fact that ACPS can not be validated at design time. Since ACPSs adapt to the changing environment, it is possible that ACPSs produce behaviours that have not been tested during the design phase. The proposed solution typically involves a form of runtime monitoring system, where the behaviour of an ACPS is tested during runtime. One novel example of this principle is the Dependability Cages proposed by Aniculaesei et al. [17]. The concept behind the Dependability Cages is that ACPSs should have a knowledge of the behaviour that has been tested to be 'safe' during design time. During runtime, the ACPS would check whether the behaviour it is trying to enforce has been tested to be 'safe'. If it is trying to perform a behaviour that has not been tested, the Dependability Cage may stop or influence its autonomous decision.

Autonomy Requirement Based on the challenges that have been identified in ACPS, it is clear that some requirements for ACPSs should not be treated as other requirements. To be precise, requirements that describe the need for a system to be autonomous should be addressed differently. This is not a new concept and there are several related works in the literature that have attempted to accomplish this. Vassev and Hinchey [31] proposed a model called the Autonomy Requirement Engineering (ARE). One of the enabling features for ARE is the description of the Generic Autonomy Requirement (GAR). GAR is a framework to capture autonomy requirements by considering system requirements based on its self-* characteristics. GAR defines nine attributes that the requirement has to elaborate in order to be properly processed in the ARE approach:

- **Autonomicity (self-* requirements)**. Autonomous systems are characterised by their self-* objectives that require autonomous behaviour (e.g., self-healing, self-configuring, and self-optimising).
- **Knowledge**. Autonomous systems need to have relevant knowledge to make autonomous decisions.
- **Awareness**. Awareness is a product of knowledge representation, environment monitoring, and reasoning.
- **Monitoring**. The process of obtaining data from the environment with sensors.
- **Adaptability**. The ability to achieve change in the system behaviour or structure.
- **Dynamicity**. The ability to reconfigure at runtime, such as adding or removing of components.
- **Robustness**. The ability to cope with errors during runtime.
- **Resilience**. The ability to bounce back from unanticipated disruptions.
- **Mobility**. Physical part of the system that moves during runtime.

A demonstration of GAR was carried out in a case study based on a space mission [201] system. It is shown in the case study that the GAR framework is targeted at capturing all self-* requirements that the system must achieve, and the ARE process would then refine these self-* requirements into self-* objectives. The rest of the properties on the list are then used to support or constrain the means by which the autonomous system can achieve the self-* objectives. Yahya et al. [30] propose a similar approach to identifying Autonomy Requirements, but with vastly different characteristics. This separates requirements specification and system goals into seven characteristics that have to be addressed when identifying Autonomy Requirements:

- **System Environment**. Analysing the environment will assist the prediction of changes and adaptive responses required of the system.
- **System Capability**. Refer to the type of system interaction with the environment through physical or data interaction.
- **Level of Autonomy**. There are different ways to define the level of autonomy, and we will discuss this further in the next section. In the framework, it uses five levels of autonomy based on work proposed by Kephart and Chess [202]. The five levels of autonomy are: *Basic*, where the system executes simple tasks; *Managed*, where the system collects information in a consolidated view; *Predictive*, where the system recognises patterns and provides advice on action; *Adaptive*, where the system can take the right action; and *Autonomic*, where system operation is governed by business policies.
- **Choice of Technology**. This characteristic leans more towards autonomic computing as it is used to describe the type of technology, such as the development environment and programming language that the system uses.

- **Runtime Requirement Assessment**. Monitor and evaluate the success or failure of achieving the system goals at runtime.
- **Decision-Making**. Deciding the action that the system should perform in order to achieve the system goal.
- **Goal Achievement Alternatives**. Clarifying the alternatives that the system could perform to achieve the system goal.

Although we are not able to find literature describing an implementation of the framework proposed by Yahya, the characteristics that it provides are clearer compared to GAR. It is unclear why dynamicity, adaptability, robustness, resilience, and mobility in GAR are required to identify the autonomy requirement. For example, autonomous vehicles do not necessarily need to have a requirement for dynamicity where components are added or removed during runtime. On the other hand, most of the characteristics in the framework presented by Yahya are more generic and applicable to all types of autonomous systems. When comparing the similarity that both frameworks share, we can summarise several factors that are important when identifying autonomy requirements:

- **Identify environment**. As an autonomous system decides its behaviour as a reaction to the environment, it is important that the environment is well understood.
- **Self-* goal**. It is a commonly accepted approach that autonomy should always be considered with respect to the system goal (and not autonomy for the sake of autonomy) [203]. This goal is not limited to any particular type of action, and since the autonomous system itself is responsible in ensuring this goal is met, it is often referred as a self-* goal, self-* requirement, or self-* objective. As such, an autonomy requirement must be based on a goal.
- **Awareness**. In order for the system to make its own autonomous decisions to fulfil the goal, it needs to be aware of all the relevant contexts. In ACPS, this typically means that the ACPS should be aware about its current state and the state of the environment.
- **Decision-making**. There must be some algorithm in the control system that decides on how the system should behave based on its awareness to achieve the self-* goal.

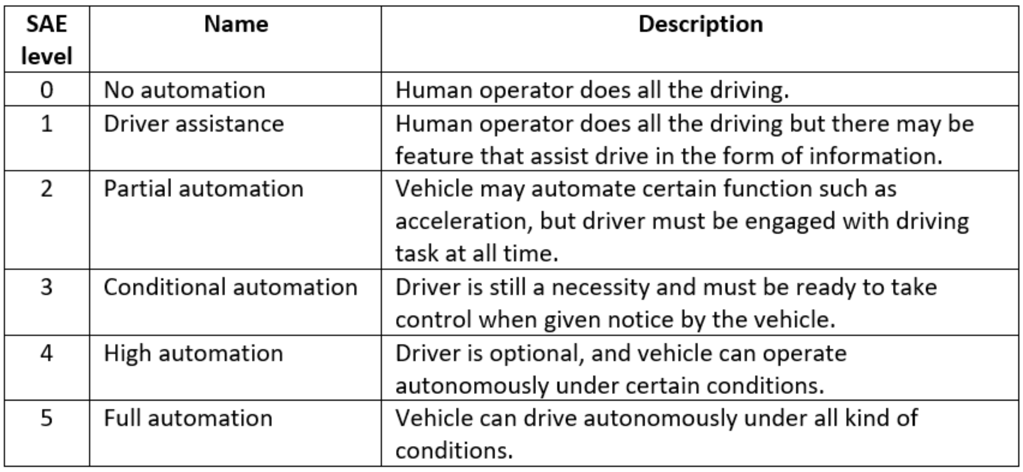
2.3.3 Level of Autonomy

In the efforts to understand the relationship between human and computer in system automation, there have been several works that propose a scale for Levels of Automation (LoA) [204] [205]. An example of such a scale is proposed by Sheridan et al. [206]:

1. The computer offers no assistance: the human must take all decisions and actions.
2. The computer offers a complete set of decision/action alternatives, or
3. narrows the selection down to a few, or
4. suggests one alternative;

5. executes that suggestion if the human approves, or
6. allows the human a restricted time to veto before automatic execution, or
7. executes automatically, then necessarily informs the human, and
8. informs the human only if asked, or
9. informs the human only if it, the computer, decides to.
10. The computer decides everything and acts autonomously, ignoring the human.

This LoA scale is very generic and could theoretically be applied to any computer-controlled systems. However, when compared to a more domain-specific LoA scale such as SAE LoA for autonomous vehicles, the comparison between levels is not as easily discernible. SAE International defines six levels of car automation [207], from level 0, where there is no autonomy involved, up to level 5, where a vehicle is fully autonomous. The detailed explanation of each level is shown in Figure 2.2.



SAE level	Name	Description
0	No automation	Human operator does all the driving.
1	Driver assistance	Human operator does all the driving but there may be feature that assist drive in the form of information.
2	Partial automation	Vehicle may automate certain function such as acceleration, but driver must be engaged with driving task at all time.
3	Conditional automation	Driver is still a necessity and must be ready to take control when given notice by the vehicle.
4	High automation	Driver is optional, and vehicle can operate autonomously under certain conditions.
5	Full automation	Vehicle can drive autonomously under all kind of conditions.

Figure 2.2: SAE level of automation for on-road motor vehicles

Figure 2.3 shows a correspondence between the LoA proposed by Sheridan et al. for computer systems, and the SAE LoA for autonomous on-road vehicles.

The full driving automation on level 5 of the SAE scale states that the vehicle should be able to operate unconditionally regardless of the presence of a human operator. However, it is not explicitly mentioned whether the human operator shall have the authority to take control of the vehicle if the driver decides to do so. Consider a vehicle design that is fully autonomous and does not even have a driver's seat anymore. In this design, the passenger could not take over the driving as there is no interface for it (e.g., no steering wheel). As long as the vehicle could drive autonomously, this design can be considered to be at level 5 on SAE LoA. We could also consider a second design which, just like a traditional vehicle, still has all the features that allow a human to drive the vehicle manually. However, it also has a system that allows it to drive autonomously. In this design, the passenger has the choice to switch between manual and autonomous driving. Similar to the first design, this vehicle would be considered to be at level 5 as well.

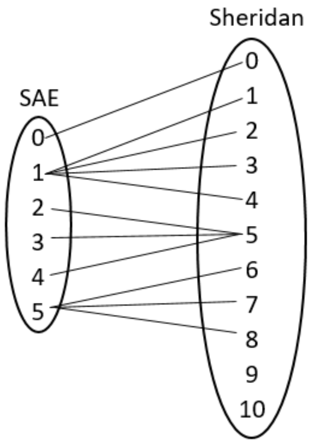
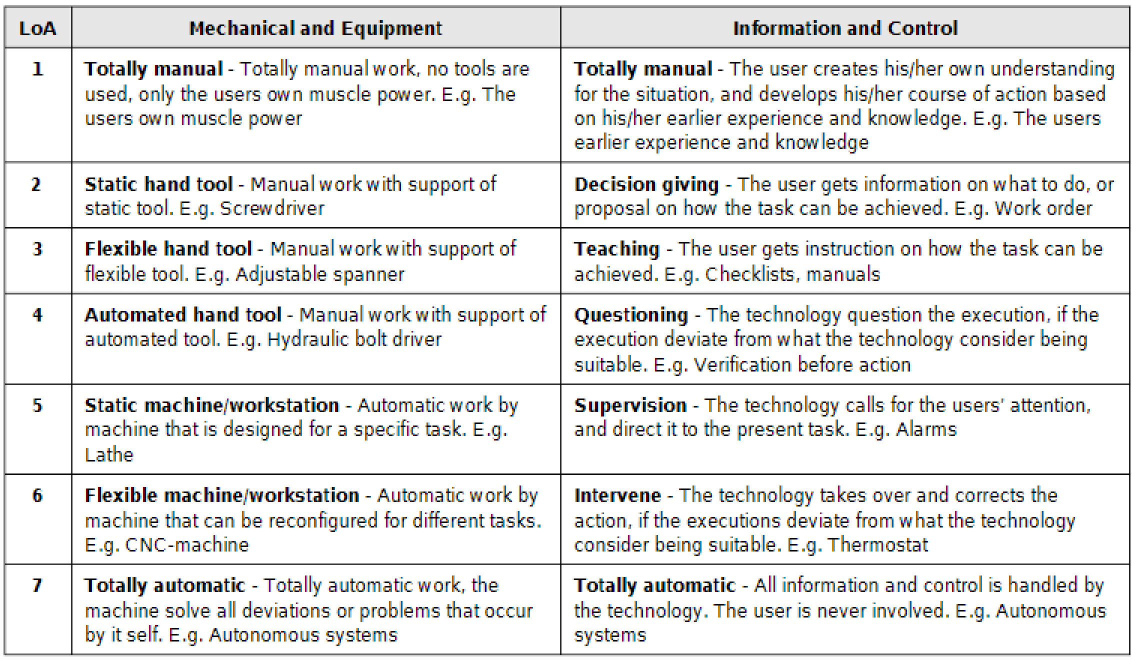


Figure 2.3: A possible correspondence between Sheridan and SAE LoA scales

However, those vehicles will be graded differently in Sheridan's LoA. It is almost impossible to grade the autonomous vehicles using Sheridan's LoA based on the description given, as Sheridan's LoA provides autonomy grades for each autonomous operation. Instead of grading an autonomous vehicle as a whole, it is more suitable that Sheridan's LoA is used to grade a specific function such as collision avoidance. In the first vehicle design, the vehicle would avoid collision autonomously and there is nothing the passenger could do to stop this. In this case, the collision avoidance feature of the autonomous vehicle would be graded at level 10. For the second design, the autonomous vehicle could avoid collision autonomously as well. However, the driver could also switch to manual control and disable the collision avoidance function. In this case, Sheridan's LoA would grade the collision avoidance feature of the autonomous vehicle to be at level 5.

This difference could perhaps be attributed to the fact that the two LoA scales are based on different autonomy characteristics. The SAE LoA scale focuses on the extent of a human driver's involvement while still achieving a system requirement: level 0 signifies the full involvement of a human driver in the activity, while level 5 is where a human driver is not required at all. Meanwhile, Sheridan's LoA focuses on the relationship between a human operator and computer in terms of the decision-making process. Levels 1 to 5 allow the human operator to have the final decision in the activity. Level 6 also allows the operator to have the final decision-making albeit being bound to a time constraint. As for levels 7 to 10, the autonomous system makes the final decision for the system processes and cannot be influenced or changed by the human operator in any way.

Another LoA scale in the domain of manufacturing is proposed by Frohm et al. [208], as described in Figure 2.4. This LoA scale focuses on the capacity of tools involved in the manufacturing process. Levels 1 to 5 of Frohm's automation scale have almost no automation at all, while level 6 shows partial autonomy and full autonomy at level 7. When the equivalence of this LoA is mapped against the LoA we explored previously (see Figure 2.5), the distinction is even clearer as there are more LoA levels that could not be directly mapped.



LoA	Mechanical and Equipment	Information and Control
1	**Totally manual** - Totally manual work, no tools are used, only the users own muscle power. E.g. The users own muscle power	**Totally manual** - The user creates his/her own understanding for the situation, and develops his/her course of action based on his/her earlier experience and knowledge. E.g. The users earlier experience and knowledge
2	**Static hand tool** - Manual work with support of static tool. E.g. Screwdriver	**Decision giving** - The user gets information on what to do, or proposal on how the task can be achieved. E.g. Work order
3	**Flexible hand tool** - Manual work with support of flexible tool. E.g. Adjustable spanner	**Teaching** - The user gets instruction on how the task can be achieved. E.g. Checklists, manuals
4	**Automated hand tool** - Manual work with support of automated tool. E.g. Hydraulic bolt driver	**Questioning** - The technology question the execution, if the execution deviate from what the technology consider being suitable. E.g. Verification before action
5	**Static machine/workstation** - Automatic work by machine that is designed for a specific task. E.g. Lathe	**Supervision** - The technology calls for the users' attention, and direct it to the present task. E.g. Alarms
6	**Flexible machine/workstation** - Automatic work by machine that can be reconfigured for different tasks. E.g. CNC-machine	**Intervene** - The technology takes over and corrects the action, if the executions deviate from what the technology consider being suitable. E.g. Thermostat
7	**Totally automatic** - Totally automatic work, the machine solve all deviations or problems that occur by it self. E.g. Autonomous systems	**Totally automatic** - All information and control is handled by the technology. The user is never involved. E.g. Autonomous systems

Figure 2.4: LoA scales for computerised tasks in manufacturing

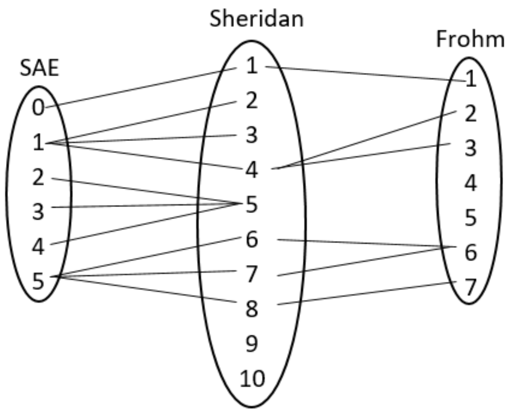


Figure 2.5: Level equivalence between multiple LoA

There are many other LoA scales, but we will only present a few examples here. Oreizy et al. [1] suggest that a spectrum of self-adaptability can be determined based on the complexity of the algorithm used (see Figure 2.6), rather than splitting into discrete levels.

Nikitenko et al. [209] define five levels of autonomy for unmanned systems with an emphasis on military applications such as Radio-control, Tele-operation (e.g., control by wire, where a controller may subtly influence control), Supervised Autonomy (goal-based control and scenario-based control), Adaptive Autonomy (the vehicle is capable of suggesting, changing, or overriding previous operator commands based on new

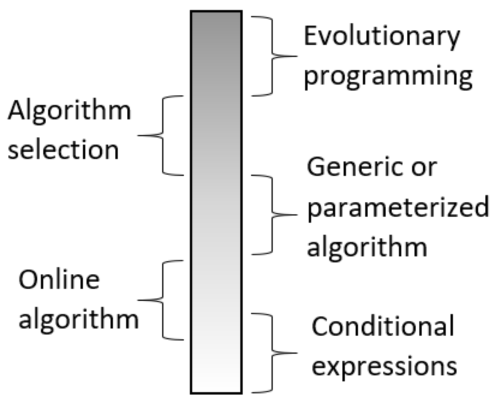


Figure 2.6: Spectrum of self-adaptability by Oreizy et al. [1]

situational awareness), and Higher Intelligence (no need for access to all sensor readings, just data for missions).

Clough (2012) defines ten levels of autonomy for UAVs based on capabilities for situational awareness, decision-making, and cooperation. Similar to other LoA scales, level 0 would indicate a UAV that has to be remotely controlled, and at level 10, the UAV is described to be 'Human-like'. Similar to the SAE LoA scale, this is very domain-specific, and there are specific characteristics such as the ability to communicate with other air vehicles that are taken into consideration when grading the LoA of a UAV.

Observation from existing works in the field of LoA scales shows that LoAs typically classify automation, ranging from full human control at the lowest level to absolutely no human involvement at the highest level. However, there are factors that differentiate one LoA from another, such that direct comparison mapping of level equivalence between scales is often not possible. In the following section, we attempt to identify the reason why one LoA scale could differ so much from others.

2.3.3.1 Automation of Process vs Autonomy of System

The term 'automation' is perhaps most heavily used in research in industrial domains. Early works in the 1950s and 1960s define automation as the extent of human involvement in manufacturing/production processes being replaced by machines [210, 211, 212]. The context is often used to describe a complex mechanical system (e.g., automobile, production line) that is in line with the characteristics of the second industrial revolution. However, as IT started to be integrated in the manufacturing domain, works in the 1990s extended the definition of automation to include computerised decision-making processes [213][214].

Frohm [208] observed this phenomenon of LoA in the manufacturing domain and drew a distinction between "mechanisation" (automation of physical tasks) and "computerisation" (automation of control and information handling) in the automation process. In contrast to works that combine the mechanisation and computerisation in a single LoA [215], Frohm proposes an LoA that separates this factor (Fig 2.4).

Following this definition, we assume that automation is the use of a machine or technology that reduces human involvement in a process and may use some form of mechanical and/or highly computerised system to achieve this. Meanwhile, autonomy has been generally defined as the ability or right to self-govern, which is heavily dependent on the decision-making mechanism. With this, it is safe to say that a system can only achieve autonomy if it is computerised, as decision-making mechanisms are taken in the context of logic gates of computer systems. This means that the scope of automation is much bigger as it includes the process of mechanisation, but autonomy can only apply to computerised systems.

While LoA scales in the manufacturing domain are focused on the process of automation, LoA scales in other domains tend to be more system-specific. One example of this is the SAE LoA for on-road vehicles.

2.3.3.2 Capability vs Right of Autonomy

The right of autonomy is determined by the relationship between a system and its human operator in terms of decision-making in system operation. Sheridan's work on LoA is the extreme application of this factor, where only the right of autonomy is considered in the LoA. Sheridan's LoA shows that the human must take all decisions and actions on one side of the LoA scale, while the computer decides everything and acts autonomously on the other. Although the capability of the computer is a part of the scale, it is not the focus as it does not determine the levels in the LoA.

Meanwhile, the capability of autonomy simply means that the LoA is determined by the capability of the system to operate without human intervention regardless of the relationship between the system and its human operator. An example of this was mentioned earlier when we tried to compare the LoA of two autonomous vehicles with SAE and Sheridan's scales.

2.3.4 Defining Autonomy for CPS

The previous section explores the different definition, classification, and application of autonomy in CPS. Although there is some degree of shared understanding to the concept of autonomy, there are still some differences across different domain applications in terms of definition and especially classification. In this section, we define what we mean by autonomy for the purpose of our work.

It is important to note that unless explicitly stated, the term autonomy will always be used in the context of CPS in this thesis. Antsaklis et al. [216] provide a good basis to the definition of autonomy: "Autonomy is the objective, and 'intelligent' controllers are one way to achieve it". It is a good description because it attempts to define autonomy rather than describe its characteristics (e.g., fault tolerant, adaptability). However, such a high level definition raises questions (e.g., can any objective be an autonomous objective? Is there a characteristic to the method the system used to achieve the autonomous objective for the system to be considered autonomous? How does the autonomous objective apply in ACPS?).

To put it simply, our definition of ACPS is that "Autonomous CPS is a CPS that is capable of achieving one or more autonomy objectives" where "autonomy objective is a high-level objective that must be achieved with limited or no external assistance". And when put together, "ACPS is CPS that is capable of achieving high-level objective(s) with limited or no external assistance". The following part of this section will elaborate on the key words used and the rationale behind this definition.

2.3.4.1 The what of autonomy objective

Defining objective

The word objective here refers to the plain English definition of the word. One interpretation would be the ISO 9001:2015, where objective is the result the system intends to achieve. Objectives can be strategic, tactical, or operational, and can apply to the CPS as a whole or as part of a process.

High-level objective

The term "high-level", "intelligent", and "smart" is often used to describe the sophistication of autonomous systems as covered in section 2.3.3 about levels of autonomy. However, we observe that the definition of "high-level" can be different between different domains of application and also evolves over time. For the purpose of our work, we define objectives in the context of ACPS as being "high-level" when the objective is both "indirect" and "multifaceted".

Indirect objective

An autonomous objective in CPS must be an indirect objective. By indirect, we mean that the objective cannot be achievable as an immediate output of control unit(s) making up the CPS. To understand this better, we can consider CPS as being made up of one or more control loop systems (this is an oversimplification as simple control loop systems are not typically considered CPSs).

In basic control loop systems [133], we have an input, control element, and output. An input in CPS could be a sensor that sends a signal to the control unit with information from the environment or the plant. An output in CPS could be an actuator that performs some physical activity based on the signal from the control unit. A direct objective would then be an objective that could be directly performed by the output (i.e., actuator).

Multifaceted objective

System	Direct objective	Indirect objective
Car	Cruise control. Wheel rotation speed is a direct output of the vehicle.	Self-driving. Wheel rotation is one of the methods to achieve the self-driving objective.
Computer chess program	Player control. Move chess piece based on user input.	Chess AI. Able to decide and move chess pieces as a method to achieving the objective of winning the game.
Air conditioner	Car air conditioner. Output cold/hot air following the user setting	Temperature Control Unit (TCU). Output the appropriate air temperature to reach/maintain the desired temperature in the plant.

Table 2.4: Examples of direct vs indirect objectives (not limited to CPS)

An autonomous objective is typically a multifaceted objective in the sense that there should be multiple ways or methods to achieve the objective. For example, the objective of collision avoidance in a self-driving car could be achieved with several methods, such as stopping the wheel to stop the vehicle, steering the wheel to change direction, or sounding the vehicle horn to alert/prevent the potential hazard. Without a multifaceted objective, the autonomy tends to be referred to as automation instead. This can be seen from the air conditioning example from table 2.1. Although the temperature control unit system is an indirect objective, the single-faceted nature of the objective can be achieved with typical closed-loop control architecture and is not typically considered an autonomous system.

2.3.4.2 The how of autonomy objective

An autonomous system is typically described as being capable of performing its objective under significant uncertainties without external assistance [216]. We generally agree with this characteristic of an autonomous system, but it would be slightly refined for our work.

External assistance

Autonomy objective(s) should be achieved with partial or no external assistance. The literature on levels of autonomy in section 2.3.3 shows that partial assistance is a common progression in the technology of autonomous systems before reaching full autonomy. This is sometimes referred to as partial-autonomy, or low-level autonomy. For example, in self-driving cars, a partial-autonomy is used to describe self-driving cars with low reliability that requires human supervision for the sake of potential control take-over. At this stage, we are more interested in defining whether or not a CPS is autonomous rather than classifying its level of autonomy (e.g., based on its reliability); Therefore, even autonomy with partial assistance is perfectly acceptable.

By partial assistance, we mean that the ACPS is perfectly capable of making decisions in the attempt to achieve its high-level objective. However, the decision made may be unreliable or wrong, so occasional assistance may be required to override the decision made at times. Note that the ACPS must have a logical reasoning to the decisions it made even when it is an unexpected/undesirable decision. This is opposed to having a random, dice-rolling decision-making computer combined with human-assistance, which will not be considered as an autonomous CPS as it lacks an intelligent control system.

Intelligent control system

ACPS requires an intelligent control system that is responsible for decision-making and controlling the system to achieve its high-level objectives. It is the presence of this intelligence control system, rather than the action of the systems itself, that marks a CPS as autonomous. Consider the difference between a self-driving vehicle that drives through an empty field and a computerised vehicle that is pre-programmed to mimic the path that the self-driving car just took. Although both vehicles would perform the same identical action that is indistinguishable from the perspective of an external observer, the pre-programmed vehicle is not considered an autonomous system.

The difference between the two vehicles only becomes apparent when there is an environmental obstacle that causes the self-driving vehicle to adapt its action, which is often referred to as "significant uncertainties" [216] in the literature. This distinction highlights that an autonomous system is not always distinguishable from its non-autonomous counterpart simply from the actions that the systems are performing. However, the presence of an intelligent control system capable of making the decision to achieve its high-level objectives makes the CPS autonomous.

Further elaboration on the classification of the control system, as well as the architecture required to achieve an intelligent control system as required to achieve high-level objectives, are covered in section 2.3.5.1.

2.3.4.3 Summary

In this section, we provide the definition of ACPS for the purpose of our work. Several key points from this definition is that the autonomy of a CPS is about its ability of being "capable of achieving" one or more "high-level objectives". A "High-level" objective is an objective that is both indirect and multifaceted, and the ACPS must be equipped with an intelligent control system to achieve its high-level objectives with either limited or no external assistance.

2.3.5 ACPS Classification

In this section we present two classifications for ACPSs that will be used to evaluate the framework for ACPSs presented in Chapter 5. The intention of this section is not to gather all types of classification available in the literature, but rather, select classifications as a way to help evaluate the research contribution. As LoA has been covered in the previous sections, it will not be included here.

2.3.5.1 ACPS Architecture

For this classification, we assume that the architecture of an ACPS follows the architecture of a closed-loop control system. We describe the minimum elements found in the architecture of ACPS to be composed of a controller, sensor, and actuator. A controller is a system or device that manages the behaviour of another system or device. In this case, the controller is a system that controls the sensor and actuator. An actuator is a system element that translates the controller output into an action by the control device. An actuator in an ACPS is used to influence the physical environment in which the ACPS resides. A sensor is a component that translates the physical process variables into measurements used within the controller. A sensor in an ACPS is used to gather information regarding the environment in which the ACPS resides. Based on this set of elements and its configuration, the architecture of ACPSs can be classed as centralised, decentralised, or hybrid (see Figure 2.7).

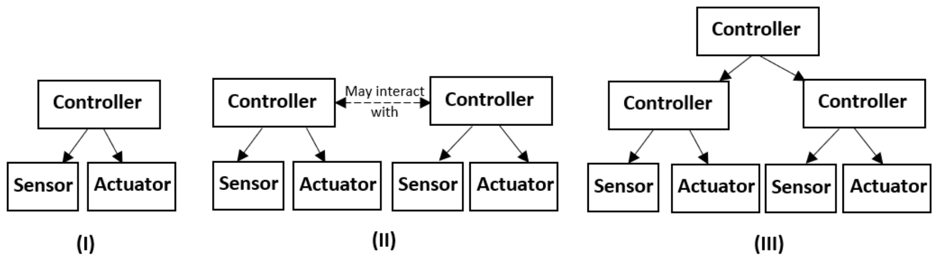


Figure 2.7: ACPS architecture with (i) Centralised control (ii) Decentralised control (iii) Hybrid control

A centralised system relies on a single controller acting as the central controller exercising control over other (lower-level) components in the system hierarchy. An example of a centralised system is shown in Figure 2.7(I) where a single controller would exercise control over the sensor and actuator in an ACPS. A centralised system typically makes better autonomous process decisions compared to other alternatives, as the central controller has access to all available information from the sensors to formulate the best decision in the given circumstances. However, the centralised approach might cause an overhead when dealing with a large system with large amounts of data.

A common limitation in a centralised system is that as the system scales to include more elements, a more powerful controller will be required to process all the information from the sensors distributed across the systems. As the data gets larger, processing time also increases, which is often undesirable. Some ACPSs may also require sensors and actuators to be distributed further apart geographically, causing an increase in communication time between system elements, which is undesirable for ACPSs that operate under strict real-time requirements. These challenges can be mitigated by using the decentralised approach, as decision-making responsibility is split between multiple controllers.

A decentralised approach utilises multiple controllers distributed within the system. This approach allows the responsibility of an ACPS to be distributed between several controllers. The controllers may interact with one another as peers and may cooperate to deliver an emergent behaviour. It is also possible that instead of a single ACPS, there would be multiple ACPSs interacting with each other. This decentralised architecture of multiple interacting ACPSs would enable a new type of autonomy known as self-organisation [217], where emergent behaviour appears as a result of interactions from several localised entities. A typical example of self-organisation found in the natural world is a school of fish that produce a certain characteristic through their individual cooperative behaviour without an explicit order from a ‘leader’.

Self-organising systems are typically considered in the context of multi-agent systems [218], where self-organisation is the result of collective behaviour between multiple agents. In the context of autonomous systems, each agent in the system would be self-adaptive, and self-organisation emerges from the cooperation between multiple self-adaptive agents and their environments. Developing a self-organising system is much harder than developing a self-adaptive system simply because self-organising systems are made of a collection of self-adaptive systems.

One example of self-organising behaviour in ACPS is the self-organising swarm of UAVs [170]. Each individual UAV in the swarm can be considered an ACPS and is capable of being self-adaptive. However, self-organising arises when a swarm of these UAVs are deployed together for a specific mission. Innocente and Grasso (2019) [219] demonstrate how self-organising UAVs can be used to fight wildfire autonomously.

Decentralised systems do not always make the best autonomous decision in a given situation as each controller does not have access to information from all the sensors distributed in the system. The controller may interact with each other to disseminate the data available to each controller, but this will in turn increase the processing requirements.

The hybrid control approach tries to combine both the centralised and decentralised approach to gain the benefits from both approaches. Hybrid control still uses multiple distributed controllers to split the autonomous responsibility. However, this operation is limited to simple and often lower-level autonomous processes. There will be another controller sitting on a higher hierarchy level above those distributed controllers. This controller will be responsible for higher-level autonomous operation and may enforce a certain degree of control over other controllers. When done properly, this ensures the top controller is only responsible for autonomous operations that require full knowledge of the system and leave the lower-level tasks to the distributed controllers to reduce overhead.

Apart from the operational independence that separate controllers may have, ACPSs might be developed following the decentralised or hybrid approach if the project requires parts of the system to have managerial independence. Managerial independence could be desired when there are multiple stakeholders in a single ACPS and the stakeholder requires their portion of the system to retain their own managerial independence.

When designing an ACPS with the decentralised and hybrid approach, it is possible that the ACPS is actually a collection of independently managed ACPSs instead. In this case, the ACPS would be considered as a System-of-Systems (SoS).

ISO/IEC/IEEE 21841 [220] provides a standard to the taxonomy of Systems-of-Systems (SoS). It states that SoS consists of a set of systems or system elements that interact to provide a unique capability that none of the constituent systems (the individual systems making up the SoS) can accomplish on its own. It also states that there are four types of SoS:

1. **Directed.** A directed SoS is created for specific purposes, and constituent systems are subordinate to the SoS. Each constituent system still retains its ability to operate independently. However, its normal operational mode is subject to the SoS' centrally managed purpose.
2. **Acknowledged.** An acknowledged SoS has recognised objectives, a dedicated manager, and resources for the SoS. Each constituent system still retains its own independent ownership, but changes in the system are based on cooperative agreement between SoS and the system.
3. **Collaborative.** Constituent systems in collaborative SoSs interact more or less voluntarily to fulfil centrally agreed purposes. Each constituent system collectively decides how to provide or deny services.
4. **Virtual.** A virtual SoS lacks central management and a centrally agreed purpose for the SoS. Emergent behaviour may still arise, but it relies on a relatively non-managed mechanism to maintain it.

SoS may unintentionally exist in an unrecognised state and be considered as "accidental" [221] SoS. However, when they become significant enough to be recognised and are brought under some management to operate in a defined way, it will then be described as "discovered" [222] and would usually fall into one of the four SoS types.

2.3.5.2 ACPS Decision-Making

This classification is concerned with the extent of the controller's decision-making ability. Figure 2.8 shows that ACPS can be classified into three types depending on the complexity of the controller's decision-making mechanism.

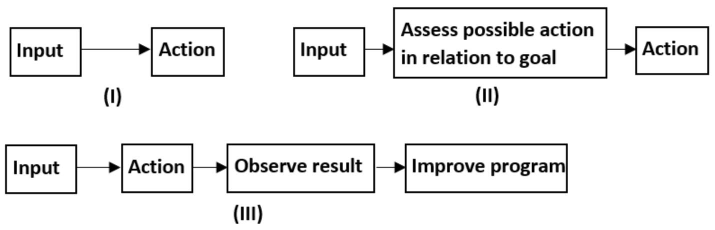


Figure 2.8: (I) Teleo-Reactive Program (II) Artificial Intelligence (III) Machine Learning

Typical operation in an ACPS involves the controllers gathering data from sensors to derive information and make an appropriate response through the actuator. When there is a strong correlation between observable input and the expected output, an ACPS controller may adopt a teleo-reactive (TR) [223] mechanism to handle the autonomous processes. Teleo-Reactive programs are a set of reactive rules that continuously sense the environment and trigger actions whose continuous execution eventually leads the system to satisfy a goal. The main advantage of a TR program is its ability to react robustly to changes in its environment owing to the continuous computation of sensing values.

$$\begin{array}{l} K_1 \rightarrow a_1 \\ K_2 \rightarrow a_2 \\ \ldots \\ K_m \rightarrow a_m \end{array}$$

Figure 2.9: Common denotation of Teleo-Reactive program

Figure 2.9 shows a structure for the TR program, where the *Ki* are conditions on perceptual inputs and on statements in a model of the world that keeps track of current perceptual data. The list of rules is evaluated from the top for the first rule whose condition is true, and its corresponding action is then executed. An action *ai* may consist of a single action or may itself be a Teleo-Reactive program. An example of an autonomous process in ACPS that is suitable for the TR program is the braking safety system in an autonomous vehicle. Another example would be a heating system that takes the reading of a room temperature and adjusts the level of heating accordingly to maintain the room temperature at a desired level. Most of the control techniques for basic embedded systems such as an electric kettle, washing machine, and printers are based on the TR program.

There are situations when the best action cannot be directly or feasibly derived from the known input. A classic example is a computer program to win a game of chess. While there is theoretically a formula that the computer can use to calculate the next best move in any given chess configuration, it is not feasible to calculate all the possible combinations of chess configuration to calculate the next best move. Certain techniques are used to make a feasible calculation that might not be the best possible move but is still designed to make the next move one that maximises the chance of winning. There are many techniques to support this kind of problem, such as heuristic algorithms and artificial neural networks. This type of work can be categorised as ‘Artificial Intelligent’. An AI solution may also be desirable when the ACPS has complex multi-faceted requirements.

For example, suppose that a Smart Grid system detects a failure in the power line and has to make a decision between two possible configurations to prevent a total power failure. The first configuration allows power to be restored to half of the electricity users in the grid with no expected downtime, while the second configuration allows power to be fully restored to all the users with an expected 15 minutes of downtime. If the objective is to both maximise the number of clients receiving electricity as well as minimise downtime, and the available options do not guarantee a result (e.g., 15 minutes downtime is just an estimation), the controller must be equipped with the capability to make a justified decision, accounting the objective with risk assessment. This requires the controller to have a model representation of itself, the environment, and the system goal to calculate the best action for each particular context.

One step further from the AI program is the machine learning (ML) capability. Some applications of ACPSs incorporate ML techniques such as Reinforcement Learning (RL) that forms its own adaptive behaviour by learning from the environment [22, 23]. In reinforcement learning, a reward signal is used to determine the performance of an agent, where each action the agent makes affects the next data it receives. The agent is typically equipped with some form of exploration mechanism, allowing it to select actions with the highest reward potential with reference to an estimated probability distribution.

The ability to learn and adopt new behaviour during runtime can be especially useful when ACPSs are required for deployment into an unknown environment (i.e., where the mathematical model of the environment may not be available) [24, 25]. For example, RL techniques could be used to enable Unmanned Aerial Vehicles (UAV) for search and rescue missions in unknown environments [26].

2.4 Model-Based Systems Engineering

To understand the concept of MBSE, we begin from the definition of system and systems engineering. The International Council on Systems Engineering (INCOSE) defines a system as "a combination of interacting elements organised to achieve one or more stated purposes"[83]. The set of interacting entities as a system may also interact with other entities, (i.e., other systems) including hardware, software, humans, and the physical world [224]. As systems become increasingly complex, there is an emerging need for techniques or methods to manage system complexities over their life cycle. This leads to interdisciplinary systems engineering. INCOSE describes systems engineering as "a transdisciplinary and integrative approach to enable the successful realisation, use, and retirement of engineered systems, using systems principles and concepts, and scientific, technological, and management methods." [84].

Model-based systems engineering (MBSE) is an approach to systems engineering that is centred around a *model*, where a *model* is "an abstract description of the reality of a putative system" [225]. A model is abstract in that it only contains details relevant to the particular purpose for which it was constructed. The MBSE methodology focuses on the creation of models as the primary means to communicate information relating to the system between engineers. INCOSE describes MBSE as "the formalised application of modelling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases" [82]. This is contrasted with a more 'document-centric' approach.

The MBSE approach is proposed as a way to reduce the cost, time, and resources for development of complex systems. The benefits of an effective MBSE approach are outlined by Holt and Perry in [226]. Specifically, they propose that the advantages include:

- Automated generation and maintenance of system documents. Systems are typically modelled digitally with computer tools that also provide automated document generation.
- The ability to measure and control system complexity. Different views can be generated to represent different aspects of a model, such as a view to represent system structure and a view for system behaviour. The different annotations allow engineers to isolate each perspective and manage the system complexity.
- Consistency across the whole system architecture. Systems that are modelled digitally can be syntax checked to ensure that the entirety of the model is consistent.
- Traceability between system artefacts. Good MBSE practices include the modelling of relationships between each artefact created. Evaluation in CPS projects shows that explicit documentation of traceability provides information that might be valuable in the development of CPSs [73].
- Increased understanding of the system. An element to the MBSE approach is to define the concepts and terminologies that will be used in the project such that engineers would have a common language for describing the system.

While MBSE is gaining popularity as a methodology to design CPS [53], the transition from a document-centric to a model-based approach does not automatically guarantee the advantages mentioned above. A poorly designed model could still inherit the same drawbacks that the document-centric approach has. MBSE activities typically involve the use of special-purpose modelling languages, tools, processes, and architecture frameworks. The following sections elaborate on these enablers.

2.4.1 Architecture Framework

A system *architecture* describes both the structure and the behaviour of the system; an architecture should define the major components of the system, and identify relationships and interactions between them [227, 228, 229, 230, 231]. Architectures must take into account development processes and must evolve to reflect changes in a system over time as they evolve to meet any alterations in requirements [230, 231]. ISO/IEC/IEEE 42010 [35] defines a system architecture to be the "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution". Maier [232] suggests that the precise form of the definition is less important than the background of what the architecture should be about and how an architecture would differ between domains.

The activity of developing an architecture is described as *architecting*. Architecting takes place within the context of a project and/or organisation and is performed throughout the entire life cycle of a given system.

It is defined by the ISO as "the process of conceiving, defining, expressing, documenting, communicating, certifying proper implementation of, maintaining, and improving an architecture throughout a system's life cycle" [35, 233].

Architectural Frameworks (AFs) have become a popular means to cope with the complexity of today's enterprises [57]. They support the specifications of architectures by providing a method for designing and describing them. As MBSE focuses on modelling of system architecture, MBSE approach would typically include an AF. An AF typically defines a common terminology, a set of views focusing on particular aspects of the architecture, a set of architecture types with varying levels of detail, and a methodology for the development and maintenance of an architecture and its views [234]. According to the standard ISO/IEC/IEEE 42010 [35], the product used to describe the architecture of a system is called an architectural description. In practice, an architectural description contains a variety of lists, tables, and diagrams. These models are known as Views.

When describing architectures, *viewpoints* are used to define a template for describing some aspect of the system. A *view* is a realisation of a viewpoint, and so represents some artefact that is usually visualised by an appropriate diagram. An *architectural framework* can be used to describe a minimum set of viewpoints, for which views must be described in order to describe a system of interest.

ISO/IEC/IEEE 42010 [35] defines an AF as "conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders". An AF specifies any number of views and viewpoints deemed necessary to produce in order to satisfy the requirements of the framework, but does not necessarily direct how the views should be derived[226].

Whilst the use of an AF is considered a prerequisite for the development of a robust architecture, it is essential that an appropriate AF is used in the process of describing a given system. Before adopting an AF, its intended use should be considered. The most widely used AFs are evaluated in [235], which include:

- The Zachman framework [236] has been proposed for defining IT-based enterprise architectures.
- Defence frameworks, such as the UK Ministry of Defence Architectural Framework (MODAF) [237] and the US Department of Defence Architectural Framework (DoDAF) [238] are intended to be used for system acquisition.
- The Open Group Architectural Framework (TOGAF) [239] is an enterprise architecture methodology, intended to provide an approach for developing IT AFs.

MBSE Architectural Framework (MBSEAF)

Although not as widely recognised as the AFs listed above, MBSEAF [240] is intended as a good starting point for any systems engineers looking to adopt an MBSE approach to designing a system. MBSEAF is a generic framework for modelling systems that is not domain-specific. It consists of nine perspectives that capture different concerns in an MBSE project. Similar to MODAF and DoDAF, MBSEAF contains one perspective dedicated to capturing the structure and behaviour of a system. Our work on producing AF for ACPS will utilise some viewpoints found in MBSEAFs that are based on capturing the structure and behaviour of a system.

Framework for Architectural Frameworks (FAF)
Where an existing AF is not suitable for a particular project, a bespoke AF should instead be created based on the requirements of the AF. Holt and Perry [226] define a meta-Architectural Framework, intended to support the definition of a new AF. AF built according to FAF can be specialised to be applied in a particular domain, or for a particular class of system. Number processes for defining an architectural framework have been elucidated according to the 'seven views' approach to process modelling. The 'seven views' approach to process modelling has been used successfully in both industry and academia for over a decade and uses a context-based approach to create various Views [241]. These processes for an architectural framework development are encapsulated into six viewpoints, which are discussed further in Chapter 5 when used in the development of a bespoke AF to be used in the engineering of ACPSAF.

2.4.2 Modelling Language

A modelling language is a graphical or textual computer language with a consistent set of rules that is used to express models. There are many modelling languages available, each with its own strengths and limitations. The quality of the chosen modelling language would affect the success of the modelling activity. The model quality framework SEQUAL [242] states that a good modelling language should be appropriate to the domain, able to express the stakeholder's knowledge, tool supported, and easily comprehensible by relevant social actors.

A comprehensive comparative assessment of modelling languages is outside the scope of our research. However, this section will explore and compare some of the most commonly accepted modelling languages in our research domain to select the one best suited for our research. As our approach follows the MBSE methodology, we explore commonly accepted MBSE modelling languages and other possible alternatives.

The Systems Modelling Language (SysML) is a general purpose architectural modelling language that supports the specification, design, analysis, verification, and validation of complex systems [243]. SysML was first created in 2003, and a version of SysML was adopted and standardised in 2006 by the Object Management Group (OMG) in collaboration with INCOSE as OMG SysML [7]. One of the changes between SysML 2003 and 2006 were some additions to deal with completeness issues associated with Heidegger's philosophical views.

[7]http://www.omgsysml.org

SysML[8] is considered a dialect of the Unified Modelling Language (UML) 2, formally defined as UML 2 Profile. Before SysML was created, UML was adopted for use outside of the software field in areas such as systems engineering and process modelling due to its appeal as a graphical notation. Despite the growing use of UML for systems engineering, it was perceived that there was a need to tailor UML for systems engineering needs. The result of this is SysML, which has 9 diagrams compared to the 13 diagrams in UML. Figure 2.10 shows the relationship between UML and SysML, along with diagram types in each language. Apart from diagram types, SysML also has a few additional stereotypes specially designed for systems engineering needs. The Venn diagram from Figure 2.10 shows the overlap between UML and SysML without making any assumptions regarding the ratio or percentage of the overlapping features.

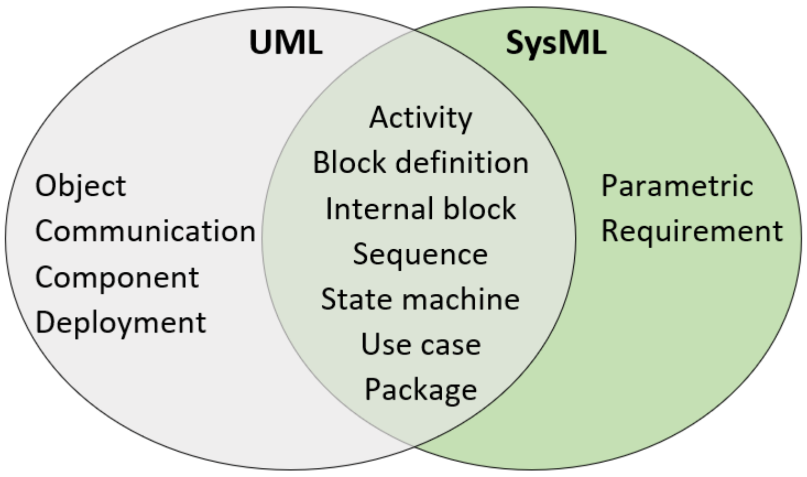


Figure 2.10: Relationship between UML and SysML diagrams

Although SysML is widely recognised as a modelling language of choice, there are other alternatives that are worth considering [244]. Formerly known as the Avionics Analysis and Design Language, the Architecture Analysis and Design Language (AADL) is a SAE published standard that uses a combination of graphical and textual modelling. AADL is composed of a relatively small set of modelling components to abstract the software and hardware entities found in a real-time embedded system. Comparison in the literature suggests that when compared with SysML, AADL generally has better support for real-time software, such as the ability to model multi-threaded systems [245]. It also captures lower-level implementation better than SysML due to its strict semantics, and has been used to model CPS as well [74]. As both AADL and SysML are seen as having their own unique strengths, it has been suggested that combining both notations would result in a more comprehensive modelling approach [246, 247]. Developing custom made domain specific modelling language for very specific applications could greatly reduce language complexity [76] but would require resources to develop a language that is very limited in domain application.

The Lifecycle Modeling Language (LML) [248] is an open-standard modelling language designed for sys-

[8]https://sysml.org/

tems engineering. It supports the full life cycle: conceptual, utilisation, support, and retirement stages. LML was developed as an approach for incorporating a logical construct and ontology within the same framework. Many of the LML models are equivalent to the familiar models that have been developed over time by UML, SysML and Business Process Modelling Notation (BPMN). Although not as widely used as SysML, it is suggested that enhancing SysML with LML would [249] provide an environment with an ontology that allows system concepts to be better represented.

2.4.2.1 Basic SysML Notations

This section introduces several basic SysML elements that will be seen frequently in this thesis. Descriptions for other SysML elements not covered here can be found from online public resources such as the OMG SysML specification documents (https://sysml.org/sysml-specs/).

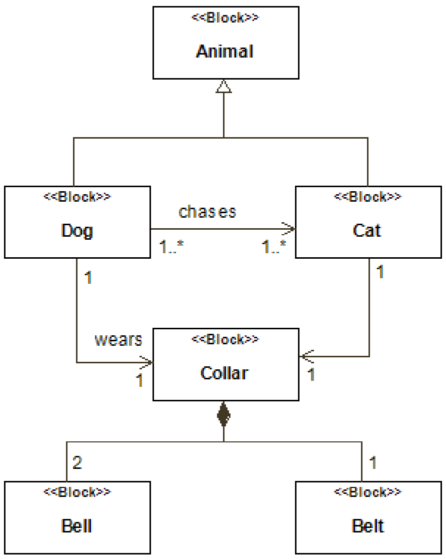


Figure 2.11: Block Definition Diagram example

Figure 2.11 is an example of a *block definition diagram*, one of the most common SysML diagrams that will be found throughout this thesis. A *block definition diagram* is used to describe System properties, classification, and hierarchy.

- *Block* represents a type of "entity" or "thing" in the System. In the diagram, a *Block* is represented by rectangles with a text inside indicating the name of the block. In the example diagram, entities such as "Dog" and "Cat" are represented as Blocks.
- *Stereotype* is represented by a text inside a Block, enclosed by double chevrons and typically located above the Block name. By default, a Block contains the stereotype *«block»*. However, new stereotypes may be introduced as a way to extend the SysML.
- A *relationship* is used to relate one or more Blocks together. Blocks are typically named with nouns, and

when combined with the relationship connecting the Blocks, it could be read together as a sentence. There are several types of relationship, such as *association*, *composition*, and *generalisation*.

- *Association relationship* is a general type of relationship that relates two Blocks together by a solid line in the diagram. The association name should be located on, or around the line, and a direction marker is used to indicate the direction the relationship should be read. In the example diagram, *association* is used to relate "Dog" to "Cat", and should be read as "Dog chases cat".
- *Composition relationship* focuses on the "parts" making up a "whole" element and is represented with a relationship line accompanied with a solid diamond at the "whole" end of the relationship. Figure 2.11 shows that a "Collar" is composed of "Belt" and "Bell".
- *Generalisation relationship* is used when a Block is refined to a more detailed type of itself. This is represented by a line with an unfilled triangle pointing to the Block that is to be generalised. In our example, it shows that "Dog" and "Cat" are a type of "Animal".

2.4.3 Modelling Tools

As MBSE shifts the systems engineering process from the traditionally document-centric approach to increased use of computer-based models, it is natural that modelling tools are required to create the model. As the modelling language defines what can be expressed in the model, the modelling language should be chosen first before the suitable modelling tools are selected. In terms of modelling tools for SysML, there is a community maintained web[9] that is dedicated to SysML modelling tools information. The modelling tools used to produce the SysML diagram throughout this thesis is Modelio[10]. The tools used and the rationale for their selection are described in Section 3.4.2.

2.4.4 MBSE Life cycle

A life cycle for a system generally consists of a series of stages, managed with a set of processes that direct whether a system is ready to transition from one stage to the other. A survey on MBSE methodologies shows that there are three commonly used life cycle models in MBSE projects [8] based on the Waterfall Model [250], Spiral Model [251], and Vee Model [252] (or also V-model).

The Waterfall Model is one of the oldest systems engineering process models. The model consists of a set of sequential phases, covering six distinct stages that start from requirements specification all the way to operations. Beginning from the start of the phase until the end, the Waterfall process consists of requirements specification, analysis, design, construction, verification, and operations. The Waterfall Model provides a very clear segmentation between each stage in the life cycle. However, this model has been criticised for rigidity [253] and an inability to readily accommodate large-scale design changes [254]. While large-scale

[9]SysMLtools, https://sysmltools.com/
[10]Modelio, https://www.modelio.org/

design changes are not ideal, they are often difficult to avoid depending on the context of the project. When forced to integrate a new system capability with the Waterfall process, there is a chance that the entire process has to be restarted.

The Spiral Model provides a unique alternative to the Waterfall Model. The Spiral Model process is represented as a spiral rather than a sequence of activities with some backtracking from one activity to another. The Spiral Model combines change avoidance with change tolerance. It assumes that changes are a result of project risks and includes explicit risk management activities to reduce these risks. Each loop in the spiral is split into four sectors. One sector is concerned with setting objectives for that particular phase, together with its constraints, project risk, and management plan. Another sector deals with the risk assessment, where a detailed analysis of project risk is carried out and a plan to mitigate it. After the risk assessment, development and validation is carried out. The final sector is the planning that deals with reviews and the decision on whether further loops are needed. The Spiral Model can also be described as a 'meta-model' [255], as each of the spirals could be conducted as a Waterfall or as another process type.

In the V-model, execution of processes happen in a sequential manner in a V-shape. The V-model is an extension of the Waterfall Model, where for each development stage, there is an associated validation and verification stage. The phases in the V-model can be categorised into the decomposition stage, implementation stage, and verification stage. The decomposition stage represents the top-down approach for the left-hand side of the V. It starts with understanding user requirements, designing the architecture of the system, sub-system, and eventually the low-level components. At the end of the decomposition stage is the implementation stage that happens at the bottom of the V process. After implementation, a bottom-up approach follows for the right side of the V. It starts by testing the system components, moving on to the sub-systems and eventually, the whole system is tested for validation. Compared to other models, the V-model is more popular for CPS [256, 50]. Figure 2.12 shows an example of the V-model process. Depending on the context of the project, the V-model could differ slightly. For example, it is possible to add several stages such as 'sub-system design' after the system design. Following the V-model principle, an extra stage such as 'sub-system test' would need to be added on the other side for verification.

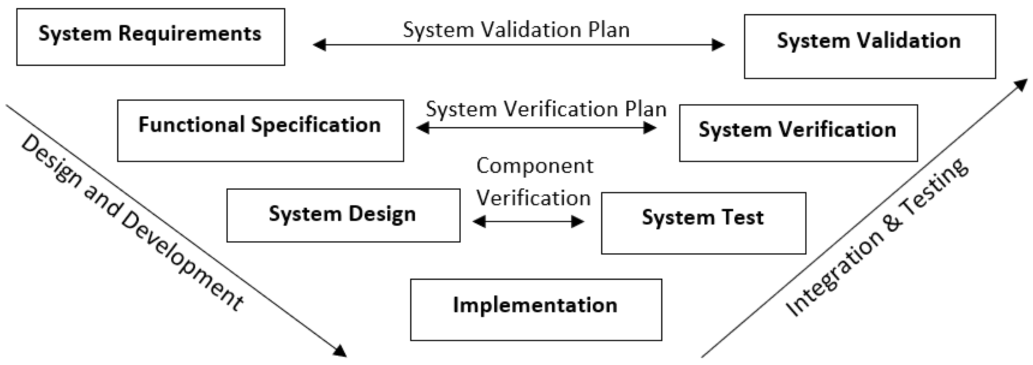


Figure 2.12: Example of V-model process

2.5 Collaborative Modelling and Co-Simulation

Since CPSs interact with the physical world, it is important to establish confidence in their dependability [257, 72]. Modelling and simulation approaches are often applied to establish this confidence. However, CPS simulation is challenged by the heterogeneity inherent in CPS [258, 64]. When modelling CPSs, a system model can be created with a homogeneous or heterogeneous approach [259]. A homogeneous approach means that a single modelling language is used to describe a CPS with a hybrid model. However, the homogeneous approach struggles to find a modelling language that could competently model both the physics and software characteristics of CPS. Virtual prototyping of CPS designs and the simulation of their behaviour in various environments typically involves a number of physical and computation/communication domains interacting with each other.

Software engineers typically employ Discrete-Event (DE) formalism to model systems. In a DE model, "only the points in time at which the state of the system changes are represented" [260]. It is common to employ a DE formalism for modelling digital hardware [261, 262, 75, 263], communication systems [264], and embedded systems [265]. Engineers expressing physical phenomena (e.g., electrical, mechanical) typically employ Continuous-Time (CT) formalism to model systems such as material flow [62]. In a CT model, "the state of the system changes continuously through time" [260]. In CT formalism, differential equations are used to describe dynamic behaviour. In order to use models to aid the development of CPSs, engineers from complementary fields must cooperate to integrate models of both cyber and physical phenomena [48, 266]. As several models based on several formalisms (i.e., CT and DE) are needed to describe the whole system, this approach is often termed as co-modelling [259, 267, 268, 49].

Co-simulation is the simultaneous collaborative execution of models and the allowing of information to be shared between them. Co-simulation consists of the theory and techniques to enable global simulation of coupled systems via the composition of simulation technologies [269]. Although each individual model in the co-model can be simulated individually in their respective simulation tools, validation with such an approach can be difficult as it provides two separate views of the overall system. Co-simulation is especially useful in CPS as it allows consideration of multiple domains' models at the same time. Co-simulation is more than just a simultaneous simulation of several simulation tools. A co-simulation framework typically supports strong interoperability among the participating elements, allowing models with different formalism to share variables during co-simulation.

One of the key enablers to co-simulation technology is a standardised interface to ensure high interoperability between modelling and simulation tools involved in a co-simulation. The Functional Mockup Interface (FMI) is a tool independent standard for the exchange of dynamic models and for co-simulation [270]. The concept behind FMI for co-simulation is to allow several simulation tools to be coupled in a co-simulation environment using a combination of XML-files and compiled C-code. The FMI standard allows users to make their models and simulation tools accessible in the form of so-called Functional Mock-up Units (FMU) that contain an FMI-based formalisation and some form of representation of the tool in question. Data exchange between FMUs and the synchronisation of all simulation solvers is controlled by a Master Algorithm.

The FMI standard is divided into two main parts, FMI for Model Exchange and FMI for Co-Simulation. FMUs produced following the FMI for Model Exchange are not co-simulation ready as it expects the FMUs to be solved by a given Master Algorithm. Meanwhile, FMUs created with the FMI for Co-Simulation standard are co-simulation ready as the FMUs themselves contain a solver such that the Master Algorithm is only responsible for data exchange.

In Chapter 5, we propose an AF for MBSE of ACPS that includes the integration of co-modelling and co-simulation as a method to support verification and validation. Although the AF itself is not tool-specific, modelling tools and co-simulation tools are required to conduct the case study in Chapter 6. In those works, we consider the use of Overture Tool[11] to produce FMUs based on DE formalism written in VDM-RT. VDM-RT is a real-time dialect of the VDM formal modelling language that can be applied to the specification of discrete controllers for Cyber-Physical Systems (CPSs). A primer to the VDM-RT language is provided in Appendix D which is based on the work by Fitzgerald in [271].

In terms of CT formalism, which is typically used to describe and model the physical phenomenon in ACPS, we will be using a tool called 20-sim[12], which supports physical modelling by providing graphical notations, which are closely related to the physical world, such as iconic diagrams and bond graphs. 20-sim supports hierarchical modelling and has powerful integration algorithms and tools for symbolically or numerically solving algebraic loops. Figure 2.13 shows an example of 20-sims graphical icons, but further details on 20-sim components will be described in the case studies chapter when they are used.

[11] Overture Tool, http://overturetool.org/
[12] 20-sim, https://www.20sim.com/

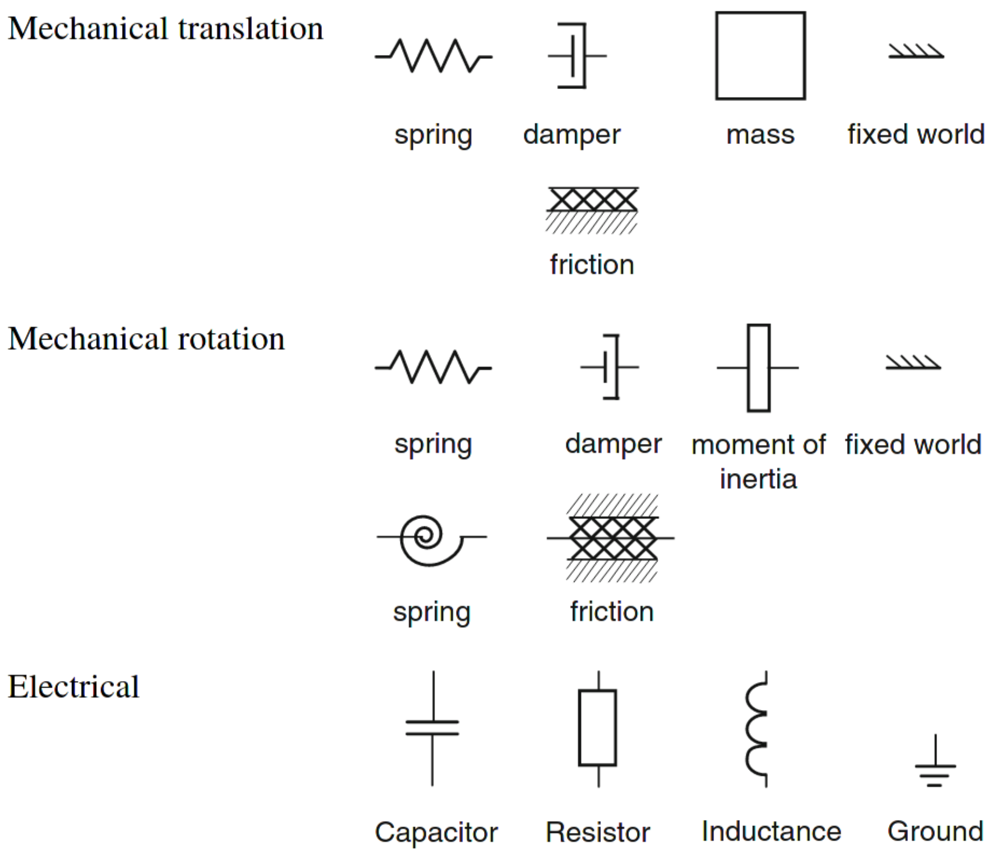


Figure 2.13: Summary of 20-sim basic icons in various domains

Although Overture and 20-sim will be the chosen co-modelling tools in our research, any FMI compliant tools can be used as a replacement or in conjunction with the tools we chose. The official website for FMI standardisation[13] maintains a list of tools that support co-simulation through FMI formalism.

Although there are non-FMI simulation frameworks specifically designed for CPS [55, 56], the model formalism supported by the framework is very limited. It is generally more beneficial to consider co-simulation based on the FMI standard as it has been supported by many modelling tools in the industry. We consider the use of Integrated Tool Chain for Model-based Design of Cyber-Physical Systems (INTO-CPS) [272] to facilitate co-simulation in our research. INTO-CPS[1] is a tool chain that supports co-modelling of CPSs from requirements through design, down to realisation in hardware and software, and enables traceability at all stages of the development. INTO-CPS supports co-simulation that is based on the FMI standard, wrapping models from different tools in FMUs, enabling interFMU communication and importing into hosting tools. The tool chain allows developers to build system models from multiple modelling tools and collaborate on heterogeneous constituent models.

[13]FMI standard, https://fmi-standard.org/tools/
[1]INTO-CPS.http://projects.au.dk/into-cps/. Accessed: 2017-09-05.

INTO-CPS tools use a Co-simulation Orchestration Engine (COE) that serves as the Master Algorithm for the co-simulation. FMUs can be generated from each individual simulation solver (e.g., Overture and 20-sim) and then provided to COE for co-simulation. The INTO-CPS Application is a front-end to the INTO-CPS tool chain. The application allows the specification of the co-simulation configuration and the co-simulation execution itself. In the INTO-CPS Application, a project comprises of: a number of FMUs, optional source models (from which FMUs are exported), a collection of multi-models, and an optional SysML architectural model. Similar to the other modelling tools, the work presented in this thesis is not restricted to INTO-CPS as the AF would support any co-simulation environment that supports FMI co-simulation.

2.6 Summary

In this chapter, we have introduced the use of systematic literature reviews in 2.1 as a means to supplement the rigour and diversity of the literature sources selected for our background research. Section 2.2 introduced the concept of CPS and its nature of closed-loop feedback architecture that will be relevant in our architecture for ACPS. It also shows that while some related technologies such as embedded systems like IoT are often used interchangeably in some literature, they are not equal to CPS in our perspective. Therefore, our work on autonomy for ACPS may not be directly transferable to them.

Section 2.3 introduced the concept of autonomy in the context of CPS (ACPS). We explore the example application of ACPSs across a variety of domains and its challenges. We also covered the level of autonomy (LoA) which provides an overview on the evolution of automation into autonomy, and the different aspects of autonomy that are used to classify autonomy into different levels in different domains. All of this provides the context information for us to provide a definition of autonomy that we believe is appropriate and useful in the domain of CPS.

In this chapter, we have introduced the concept of ACPS, its associated benefits, and its application across a wide range of domains. However, ACPS comes with non-trivial challenges due to its requirement to adapt to unexpected environments, the ability to learn new behaviour, and incomplete specifications. As ACPSs are increasingly deployed in safety critical scenarios, there is a great reliance on its dependability. Several methods have been proposed in the literature that typically focus on increasing the quantity and quality of testing and runtime monitoring systems. However, the first step to implementing such methods is to identify autonomy requirements in ACPS.

Section 2.4 explored MBSE as a popular and effective approach to designing CPSs. It introduces many concepts and tools that will later on be a critical part in the research, such as system, architecture, architectural framework, and SysML. Lastly, Section 2.5 introduced the concept of collaborative modelling and co-simulation as a method to analyse CPSs. It introduces tools such as Overture (VDM-RT) and 20-sim as modelling tools that will be used later on in our work. Justification for tools selection will be made later on after the research methodology.

3 Research Approach & Methodology

In this chapter, we start by defining the scope of our research in Section 3.1 that enables the selection of our research methodology. In Section 3.2, we utilise a framework to choose a research methodology based on the nature of our research. The methodology chosen is Design Research Methodology, and its application in our research is discussed in Section 3.3. This methodology captures the research requirements and objectives as Documentation of Intended Support as discussed in Section 3.4. The whole chapter is then concluded in Section 3.5 with a summary.

3.1 Scope

Our research is interested in developing and evaluating MBSE approaches that are intended to facilitate the architecting of ACPSs. Overview of the state of the art in relation to our research scope is shown in Figure 3.1.

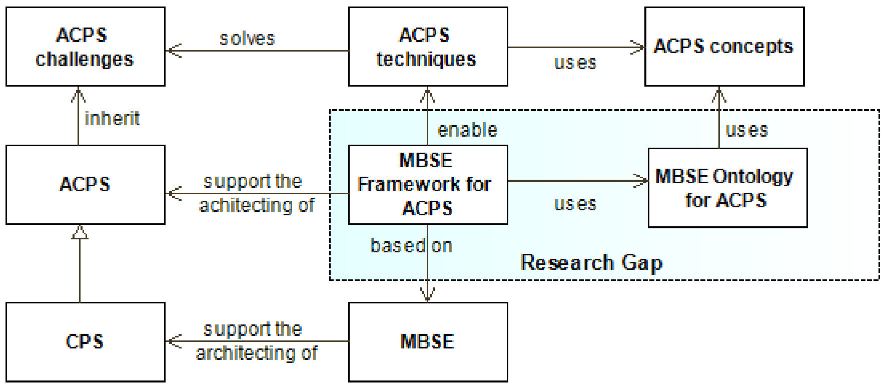


Figure 3.1: Overview of state of the art and scope of thesis

Although MBSE has been used to support the architecting of CPSs, it does not provide the same level of support for the architecting of ACPSs. Here, ACPS is considered a subset of CPS that exhibits some form of autonomy and is different from CPS as ACPS inherits challenges that are unique to autonomous systems. Therefore, MBSE's current state of the art does not contain an approach that addresses these challenges. As covered in Section 2.3.2, ACPS challenges can be summarised as follows:

1. How can we have assurance in the behaviour of ACPSs even in uncertain environments?
2. How can we have assurance in the behaviour of an ACPS even when the system's behaviour changes at runtime?

Challenges of this nature can be found in many autonomous systems. However, we are only concerned about these challenges in the context of CPSs, and we refer to these as the ACPS challenges. Although these are relatively novel challenges, efforts can be found in the literature, where several solutions have been proposed to address these challenges such as runtime monitoring approach [17, 16] and automated test case generation [19, 273, 200].

However, these techniques are not readily transferable to the MBSE processes. This gap is caused by the lack of a common ontology and MBSE framework to capture autonomy characteristics in ACPS required to use the ACPS techniques (explained in further detail within Section 3.3.4.1). We explicitly describe this gap between MBSE and ACPS as the research gap. One solution to this issue would be to develop MBSE-specific techniques for ACPS. However, the more effective solution would be to advance MBSE's state of the art such that it could be integrated with already available ACPS techniques.

Bridging this gap would include the development of MBSE ontology for ACPSs that integrates ACPS concepts with MBSE concepts and the development of an MBSE framework that can capture the characteristics of autonomy in ACPS to enable the use of existing ACPS techniques within the MBSE processes. We conjecture that by employing MBSE Ontology and Framework that is specialised for autonomy, the MBSE approach can be used with existing ACPS techniques to design a dependable ACPS. Case studies will be used to validate this conjecture, demonstrating and evaluating the use of an MBSE framework in the design of autonomous CPS.

3.2 Choice of Methodology

Research can be seen as a process of creating new knowledge. Good research should have a sound methodology, and rigorous research should take a clear stand on its philosophical position as the practice of automatic reliance on experimentation or atheoretic pragmatism is likely to impede the discipline's growth [274]. This is especially important when conducting research for systems engineering as the discipline's methodology has yet to mature. One method for choosing a suitable research methodology for research is by using a framework designed to help this process. Ferris (2009) [275] proposed a framework that utilises a series of questions to suggest suitable methodology based on the nature of the research. The framework is especially suitable for our research as it is targeted for systems engineering research. The following points are the six questions based on the taxonomy of research methods that may be used to assist in the selection of a research method.

- **Desiderata.** Is the research intended to make a significant contribution to the theory, practice, or both theory and practice of the field? Our research intends to contribute to both the theory and practice of the field. The research question itself is theoretical, but the research involves the creation of artefacts (i.e., MBSE Ontology and Framework) that can be used by ACPS engineers.
- **Relation to knowledge.** Are the knowledge aspects in the proposed project primarily desired as the main goal, as a means to achieve practical application, or is the knowledge both a goal and means to achieve practical application? Knowledge is both a goal and means to achieve a practical application in our research. We believe that certain knowledge is only meaningful if it can be translated into practical application, and the practicality of an application can only be improved with sufficient knowledge.
- **Person who benefits.** Is the primary beneficiary of the knowledge expected within the proposed project the researcher themself, or people other than the researcher? The primary beneficiaries of our research are ACPS engineers, a group of people not limited to the researcher themselves.
- **View of certainty of knowledge.** Does the proposed project presuppose that the knowledge to be developed concerns matters which objectively exist, or are they constructs of the community? This is a difficult question to answer because we believe that there are two sides to the research that have to be considered. From the engineering perspective, we assume a more realist perspective that there is one truth. A car can be considered faster than a bicycle simply by comparing their potential maximum acceleration and velocity. However, this view might not hold true in social science where the interaction of humans using said technology has to be considered: a car can move faster than a bicycle, but only if the driver knows how to operate the car.

  The work reported in this thesis is largely confined to the engineering perspective of the technology. This does mean that there will be social aspects that are not addressed, such as how long it takes for a domain expert to learn and adopt the newly-proposed technology, and what training approaches are most effective. This is considered a subject for future work and is discussed further in Section 7.3.2.
- **View of tradition.** Does the proposed project presuppose that the existing framework of the field should be used as a foundation, or be rejected? Existing framework of the field should be used as a foundation because it is difficult to contribute something that progresses the knowledge and practicality of the field without it. This means that where applicable, our research should reuse existing MBSE processes when building the ontology and framework.
- **Objective of life.** Is the subject matter of the proposed project to enjoy knowing, enjoy practice, or enjoy both knowing and practice? This is closely related to the desiderata and relation to knowledge. Our re-

search is to enjoy both the knowing and practice as we consider the theory and practical value of knowledge to be two sides of the same coin.

Based on these answers, the suggested research methodology that best suits our research is 'Design Research'. Design Research is described as an engineering research methodology that addresses a problem which is important and novel through the activity of designing a solution. In the next section, we present a short literature into Design Research and how it is applied in our research.

3.3 Design Research Methodology (DRM)

The emergence of Design Research as a recognisable field of study can be traced back to the 1960s, initially marked by the Conference on Design Methods initiated in 1963 [276]. Design Research is concerned with the 'formulation and validation of models and theories about the phenomenon of design' [277]. This would also include the development of any knowledge, methods, and tools grounded in those theories. Frankel and Racine [278] suggests that design research can be broken into three categories: Research for Design; Research through Design; and Research about Design.

- **Research for Design.** Research for Design is described as primarily prescriptive research methods for specific and feasible design solutions [279]. Archer [280] states that "there are circumstances where the best or only way to shed light on a proposition, a principle, a material, a process, or a function is to attempt to construct something, or to enact something, calculated to explore, embody, or test it".
- **Research through Design.** In this approach, the emphasis is on the research objective of creating design knowledge, not the project solution. The most important aspect of research through design is that it seeks to provide an explanation or theory within a broader context [281]. Research through Design creates new knowledge through an action-reflection approach, and much of the subject matter has been derived from social sciences, business, and marketing [282].
- **Research about Design.** This area of research searches for an explanation through the experience of designers and those who use the products, which help to address the nature of design activity, design behaviour, and design cognition. The study in defining the design problem or "problem setting" is one such example, relating to Rittel and Webber's "wicked problem" that references problems lacking clarity in both their aims and solutions [283][284].

Frankel and Racine [278] state that the three categories of design research are interrelated and represent different levels of design knowledge. Depending on the initial question or hypothesis, one can engage in a variety of category combinations at any level. It is important to note that while we will generally refer to our research method as Design Research, our research method is based on Research for Design, where an artefact is designed and discoveries are made through the activities of designing.

Although the philosophy behind the Design Research is relatively well understood, existing knowledge, methods, and tools to support effective application of the methodology are still lacking [285, 286]. Blessing and Chakrabarti [277] present one of the relatively limited contributions on the engineering of Design Research as a systematic methodology. The Design Research terms and methods used in this research are based on their work in [277] and will be referred to as DRM.

DRM can be used to support all three categories of Design Research [278] (i.e., Research for Design, through Design, and about Design) by selecting the appropriate research type from the 7 research types (see Figure 3.3) in DRM. DRM proposes a methodological framework that categorises Design Research activity into four different stages: Research Clarification, Descriptive Study I (DS-I), Prescriptive Study (PS), and Descriptive Study II (DS-II). The framework notes that each research project typically focuses on one or two of the stages, and that iterations between stages will take place.

The Research Clarification stage aims to identify the research goal by focusing on the research problems, questions, and hypotheses. This stage helps to provide an overview on the initial picture of the existing and of the desired situation. Descriptive Study I (DS-I) aims to obtain a better understanding of the existing situation and identify factors that influence the research question. Prescriptive Study (PS) is concerned about the development of the actual Support that is intended to address the identified need (i.e., building the solution). Descriptive Study II (DS-II) focuses on the evaluation of the designed Support, identifying whether the Support can be used for the task it is intended for and provide the expected effect. The following sections will elaborate further on the four main stages of DRM and how they are realised in our research.

3.3.1 Research Clarification

The first stage of DRM is the Research Clarification stage [277], which goes through an overview of the research to be undertaken. The Research Clarification stage is split into several smaller parts:

1. Identifying overall topics of interest
2. Clarifying current understanding and expectations
3. Clarifying criteria, main questions, and hypotheses
4. Selecting type of research
5. Formulating research plan

The overall topics of interest have been covered in Chapter 1, which sets the domain of interest around Autonomy, CPS, and MBSE. Chapter 2 covers the parts regarding current understanding and expectations through a literature review. Therefore, we will cover the remaining three sections of the Research Clarification stage in DRM within the following sections.

3.3.2 Clarifying criteria, main questions, and hypotheses

We conclude from the literature review that there is a need to improve on the existing engineering methodology of ACPS to meet the growing demand and complexity of ACPSs. One of the more common conclusions found within literature on the methods to achieve this is typically through model engineering framework to support common theory, methods, processes, and paradigms. Based on this initial motivation, we can derive an Initial Reference Model, which is a diagram that maps research factors and criteria shown in figure 3.2.

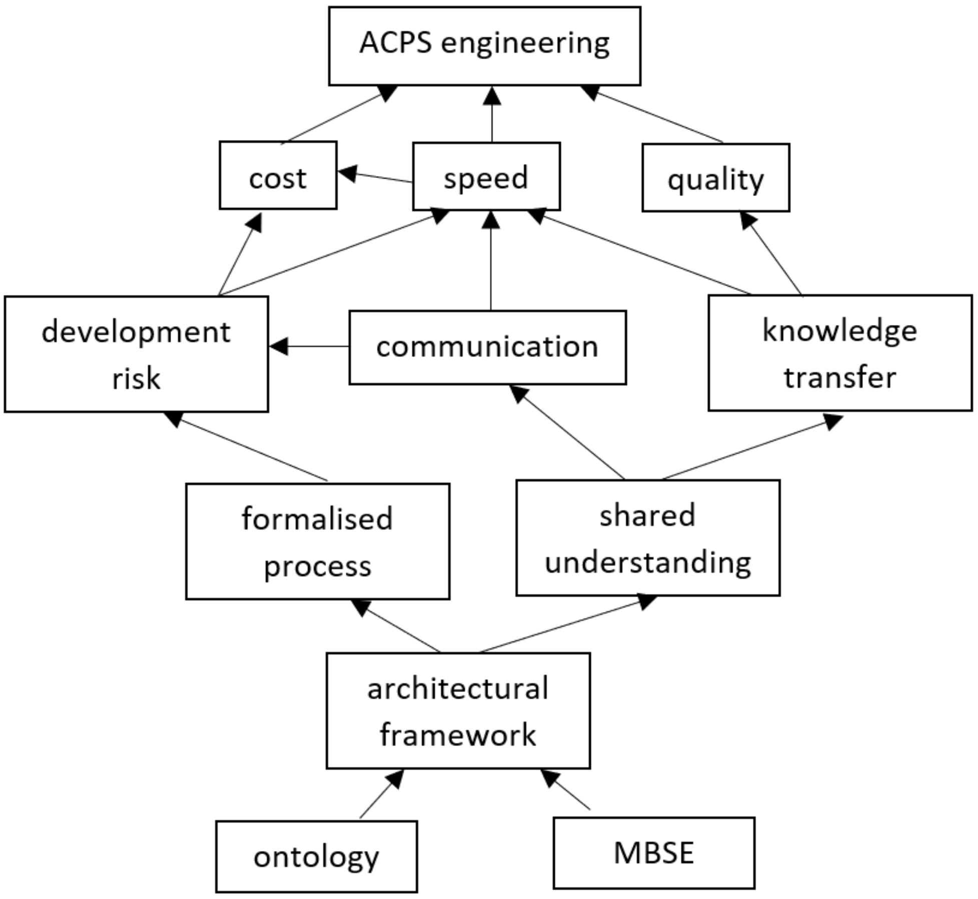


Figure 3.2: Initial Reference Model

In DRM, criteria is used to aid the measurement of research success by creating a link between the research's factor of interest and its success factor. In this phase, two types of criterion should be defined. The first criterion is the *criterion of success*, which is typically a high level and relatively abstract criterion. The second type is called *measurable criterion*. As its name implies, this criterion is capable of being measured against the results of the research.

1. **Criterion for success:** Improve the quality and efficiency of ACPS engineering projects. This is the

high-level goal of the research, which requires a more granular, measurable factor.

2. **Measurable criterion:** Improve the communication, knowledge transfer, and reduction of development risk on ACPS engineering projects. These are the criteria that can be used to measure the success of the research.

Based on these clarifying criteria, we can summarise our research question and hypotheses as follows:

1. How can the engineering process of ACPS be supported to increase project performance?
2. What are the artefacts needed to support the systems engineering of ACPS?

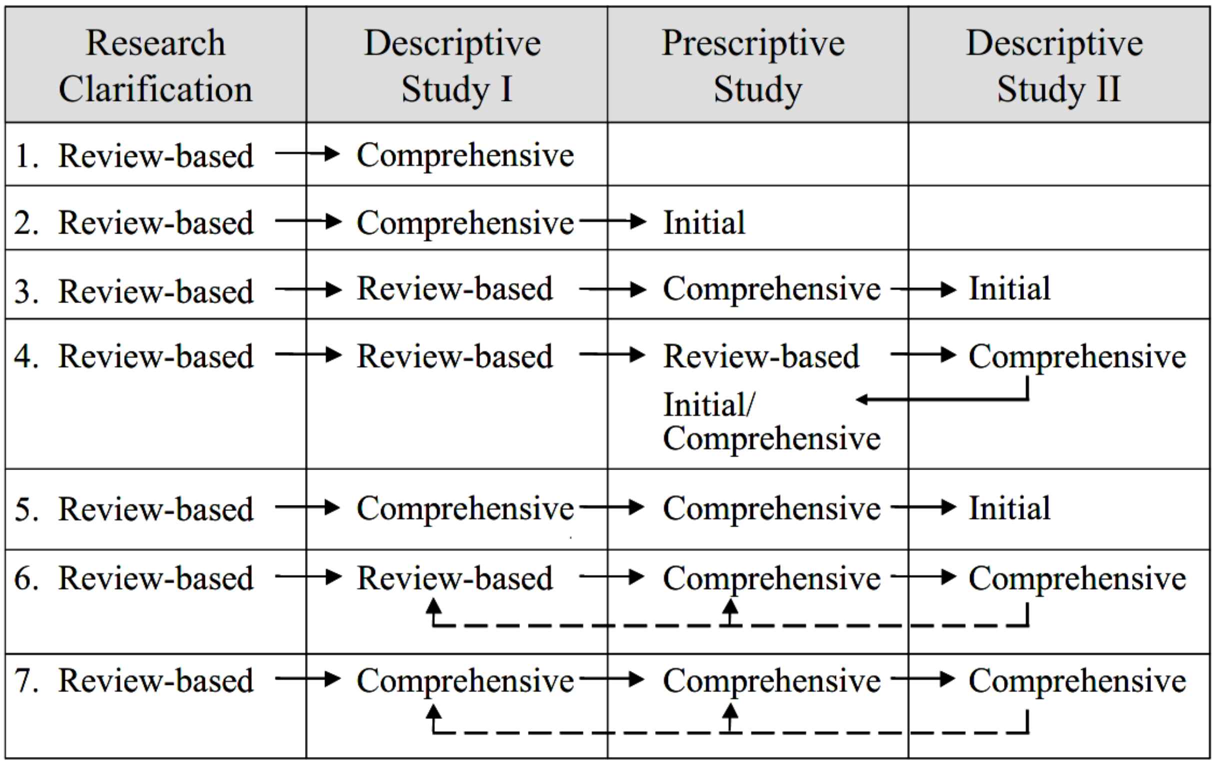
Our hypothesis is that "Using Architectural Framework for ACPS improves the quality and efficiency of the engineering process of ACPS.". This is achieved by improving communication, providing knowledge transfer, and reducing development risk through the use of formalised processes and shared understanding of ACPS engineering processes, which is provided by the architectural framework specifically designed for ACPS.

3.3.3 Selecting type of research

The next step is to identify the type of research suitable to answer the chosen research questions and verify the hypotheses. The DRM is based on four main stages of research development: the Research Clarification, Descriptive Study I, Prescriptive Study, and Descriptive Study II. DRM suggests that depending on the literature's state of the art and the aim of the research, there are seven different types to DRM research as shown in Figure 3.3.

To select the most suitable DRM type for our research, we need to select the appropriate research type to be conducted on each of the four research stages.

- Research Clarification - Review-based. The Research Clarification stage is always based on review-based research, so no decision-making is required here.
- Descriptive Study I - Review-based. Comprehensive Descriptive Study I is typically done when there is low confidence in the factor of research interest. For example, due to non-existent or insufficient literature on related topics. This is not the case for us as there is sufficient literature that suggests the benefits of a model engineering framework for ACPS; therefore, a review-based method has been selected for this stage.
- Prescriptive Study - Comprehensive. The focus of our research will be on the development of the intended Support (i.e., ACPSAF), so Comprehensive has been selected for this stage.
- Descriptive Study II - Initial. We believe that a comprehensive Descriptive Study II would go beyond the resources of our current research. To make meaningful validation (i.e., to measure whether ACPSAF improves the quality of ACPS engineering), the research output should be implemented and measured on



Research Clarification	Descriptive Study I	Prescriptive Study	Descriptive Study II
1. Review-based →	Comprehensive		
2. Review-based →	Comprehensive →	Initial	
3. Review-based →	Review-based →	Comprehensive →	Initial
4. Review-based →	Review-based →	Review-based → Initial/ Comprehensive ←	Comprehensive
5. Review-based →	Comprehensive →	Comprehensive →	Initial
6. Review-based →	Review-based →	Comprehensive →	Comprehensive
7. Review-based →	Comprehensive →	Comprehensive →	Comprehensive

Figure 3.3: Types of Design Research based on DRM

real industrial and/or commercial use cases. However, since industrial ACPSs are typically large, expensive projects, new technology will typically have to go through smaller case studies and improvements before being approved for use. As the focus is on the development of research support at this stage, it is sufficient enough to perform an initial research to Descriptive Study II, which will serve as a proof-of-concept rather than complete validation.

Based on our selection made for each research stage, the type our research falls into is the third row from the table entry in Figure 3.3, referred to as Type 3 in DRM. This also means that our research method falls in the Research for Design category of Design Research, where we make discoveries through the activity of designing a potential solution.

3.3.4 Formulating research plan

As we follow the DRM research methodology, our research plan would be in relation to conducting each DRM research stage from the Type 3 research shown in Figure 3.3. The first stage of DRM, the Research Clarification stage, was covered earlier in section 3.3.1. The rest of this section will elaborate further on the other three DRM stages and how they apply to our research.

3.3.4.1 Descriptive Study I (DS-I)

Review-based DS-I consists of two steps: reviewing literature and drawing an overall conclusion. Both of these steps have been covered in Chapter 2, where partial conclusions are made at the end of each section about the literature's particular topic. In this section, we develop a theory on what factors influence the measurable criterion. In our case, we mentioned that existing ACPS techniques are not currently transferable into MBSE processes, allowing us to develop a theory by answering why this problem holds. A combination of literature reviews and observations are used to propose the following theories:

1. **Lack of MBSE ontology for ACPS**. ACPS techniques use concepts, theory, and terms that can be traced to domains such as control theory, artificial intelligence, autonomic computing, and software engineering. Meanwhile, MBSE is rooted in the Systems Engineering methodology and would naturally inherit concepts from the domain of systems engineering. For example, the notion of requirement, behaviour, and environment might differ between the two domains.
2. **Lack of a MBSE framework that identifies autonomy characteristics in ACPS**. The literature on ACPS techniques typically assumes that certain autonomy characteristics have been identified. For example, literature on the ACPS techniques propose a method to verify requirements in relation to autonomy behaviour in ACPS. However, this is based on the assumption that the autonomy requirements for the ACPS have been identified. While MBSE could capture requirements utilising RE processes, it does not have the means to identify autonomy requirements. Observations from our literature review shows that this is not an issue specific to MBSE. The concept of autonomy for autonomous systems in general does not have a unified view beyond the definition of the word autonomous. While some specific domains, such as the automobile, are able to form a standard (i.e., SAE LoS) that gained widespread recognition, the definition and measurement of autonomy in other domains are still unclear.

3.3.4.2 Prescriptive Study (PS)

In this stage, we attempt to generate a scenario of the desired situation based on the factors described in DS-I. The desired scenario for our research would be:

1. **There exists an MBSE ontology for ACPS**.
2. **There exists an MBSE framework that identifies autonomy characteristics in ACPS**.

We consider MBSE Ontology and MBSE Framework as essential to our criterion of success, and DRM refers to this as the Key Factor. To satisfy this Key Factor such that the criterion of success can be fulfilled, our research would be designing a Support. Using the terminology from DRM, Support refers to artefacts, such as methods we designed, to achieve the criterion of success.

PS can be further categorised into three types based on the intended maturity level of the developed Support.

Initial PS results in the description of the intended Support, where only the concept is developed. *Comprehensive PS* results in a Support where its core functionality can be evaluated against the purpose for which it was developed. **Review-based PS(s)** are carried out on existing Support, where reconstruction is required to get it to the next phase. In other words, review-based PSs rework existing Support when it has not been carried out adequately. The activity involved during the PS stage of DRM is as follows:

1. **Task clarification.** This step is to establish the problem to be solved by clarifying its requirements and defining the desired situation. There is no specific method to record the requirements that have to be satisfied. However, DRM uses the checklist-based requirement list presented by Roozenburg and Eekels [287], which is the format that we will adopt as well. The requirements produced here are referred to as the Intended Impact Model.
2. **Conceptualisation.** In this step, functions and sub-functions for the Support are identified, together with concepts of how these functions could be fulfilled. Once the functionalities are decided, we create a concept plan by introducing the Support, and this concept plan considers the life-cycle phases of the Support. It is referred to as the Intended Introduction Plan.
3. **Elaboration.** At this stage, we should be able to describe the Support in terms of the problems to be addressed, goals and objectives of the Support, its elements, underlying concepts, theories, assumptions, rationale, and how it is to be realised. This description is referred to as the Intended Support Description. Together with the Intended Impact Model and Intended Introduction Plan, these three documents are called the Documentation of Intended Support.
4. **Realisation.** Due to restrictions such as time or expertise, the Support produced is often only a subset of what was planned in the Documentation of Intended Support. On top of the development of the Support itself, this stage should also reflect on how the Support stands in comparison with the Intended Introduction Plan and Intended Impact Model. The result of this comparison can be used as the basis for Documentation of Actual Support, which is composed of the Actual Introduction Plan, Actual Impact Model, and Actual Support Description that reflect the actual Support designed.
5. **Support Evaluation**. Support Evaluation involves verification that the Support fulfils the requirements. This stage typically uses the documents produced during the previous stages against the Support created to check for consistency and completeness.

As our research interest is to develop the method to fill the research gap between ACPS and MBSE, it is safe to say that this stage will be one of the focus of our thesis. Although we have conducted a preliminary study to come up with both the Criteria definition and Descriptive Study I, those two stages are not our main focus. For example, if our research interest is on Descriptive Study I, this thesis would instead be focused on identifying factors causing the gap between MBSE and ACPS, followed with gathering empirical evidence showing that the lack of MBSE Ontology and MBSE Framework have direct causality with the gap between MBSE and ACPS.

The research activity for the mentioned five stages is distributed at different parts in this thesis. The Task Clarification, Conceptualisation, and Elaboration stage will be covered in this chapter in the next section. The Realisation for MBSE ontology is covered in Chapter 4, while Realisation for the MBSE framework is covered in Chapter 5. At the end of both chapters, evaluation of the Support developed will be carried out that would constitute the Support Evaluation stage.

3.3.4.3 Descriptive Study II (DS-II)

The second descriptive study is used to evaluate the application of the Support developed in PS. This is carried out with two types of tests. First is the application evaluation that assesses the functionality from the perspective of a user. The second test is the success evaluation, which is to check whether the Support developed achieves the criterion for success. DRM states that most Design Research projects consist of demonstrators or concepts, where evaluation is aimed at obtaining a proof-of-concept rather than full realisation and deployment in the field, which is suitable for the scope of this research.

DS-II is classified into Initial and Comprehensive DS-II. Initial DS-II is an initial evaluation of the research project that provides an indication of the applicability, usability, and issues of the Support developed. It would also provide some suggestions as to how a proper evaluation can be carried out as future work. Meanwhile, Comprehensive DS-II would be an extensive evaluation that assesses the effects of the proposed Support and informs further development.

For our research, we would carry out Initial DS-II. DRM states that Initial DS-II focuses on Application Evaluation, and the link to the Measurable Success Criteria is argued by using the literature itself. One method of carrying out an Application Evaluation for Initial DS-II is for the researcher to act as the user and create a use case as a starting point and evaluation benchmark. The conclusions that can be drawn with this method are limited as no real target users are involved. Therefore, no conclusions can be drawn whether future users can or are willing to adopt the Support. Despite the limitations of these initial evaluations, researchers would still gain useful insights into the problems and issues related to the proposed Support. Evaluations and findings from Initial DS-II should be treated as:

- an indication of the applicability, usability, and usefulness of the Support;
- an indication of the issues, factor, and links that need detailed evaluation;
- a suggestion for a proper Evaluation plan.

Although the DRM literature is quite clear on what constitute a Comprehensive DS-II, the description of Initial DS-II is that it follows the step of Comprehensive DS-II but on a “lesser extent or in lesser detail”, and that the “level of detail the steps have be executed in an Initial DS-II depends on the aim of the research project”. This methodology leaves the researcher utilising Initial DS-II the freedom to conduct their validation without any minimal standard on the level of rigour. Needless to say, the lesser the rigour of the validation, the lesser the extent of indicative claims that can be made about the Support.

In our research, Initial DS-II is carried out in Chapter 6 in the form of two case studies demonstrating the application of the developed Support. These case studies are selected and conducted by the author, which makes it impossible to validate the actual effectiveness of ACPSAF as it will not have been validated against its intended use case. This is the complex engineering of ACPS that typically involves multiple stakeholders and engineers. We will attempt to mitigate some of these weaknesses, for example, by selecting only existing, well-established autonomy challenges in the CPS domain rather than coming up with a novel challenge in ACPS to prevent trying to solve a non-existing problem.

At the end of each case studies, an evaluation will be made to indicate its applicability and issues, accounting for the limitation of the validation due to the weakness of the Initial DS-II conducted. An overall conclusion from the two case studies and a suggestion for a proper Evaluation plan will be covered in Chapter 7.

3.3.5 Conclusion

In this section, we have covered DRM as our research methodology of choice. DRM research is classified into four main research stages: Research Clarification, Descriptive Study I (DS-I), Prescriptive Study (PS), and Descriptive Study II (DS-II). In section 3.3.1, we cover the first stage of DRM, the Research Clarification, and where each component making up the Research Clarification is located in the thesis, which includes section 3.3.2, 3.3.3, and 3.3.4.

Section 3.3.2 provides an overview on the research questions and hypotheses. Section 3.3.3 covers the type of DRM research method that will be conducted for our research, and in Section 3.3.4, we formulate tasks to do on each DRM research stage.

It is important to note that since the type of DRM research selected is the Type 3 Development of Support, there will be significant limitations to the claims that can be made on the success of the designed Support. As indicative of the characteristic of Initial DS-II, the conclusions derived from Initial DS-II validation can only be used as an indication on the effectiveness of the designed Support rather than proof, and the main contribution for this type of research would be the designed Support itself. The detailed evaluation on the validation result will be covered in the concluding sections of each case study and in Chapter 7.

3.4 Documentation of Intended Support

This section presents the Documentation of Intended Support, which is composed of the Intended Impact Model, the Intended Introduction Plan and the Intended Support Description. This documentation is carried out as a part of Prescriptive Study in DRM, covering the Task Clarification, Conceptualisation, and Elaboration stage.

3.4.1 Intended Impact Model

This section presents the Intended Impact Model for this research. The definition, purpose, and framework used to create this model are described in Section 3.3.2. The intended Impact Model for the MBSE ontology is shown in Table 3.1 and Table 3.2 for the MBSE framework. The first column in the Intended Impact Model indicates the level of requirement, where *D* indicates a demand for something that has to be achieved, and *W* indicates a wish for something that is desirable but not required. Note that from this point on, we refer to the MBSE Ontology Support as ACPS Ontology and the MBSE framework support as ACPSAF (Architecture Framework for ACPS).

Problem statement:	
Develop an MBSE ontology for ACPS that supports the characterisation of autonomy to be used in a MBSE framework.	
Performance:	
D	The Support should be described in MBSE modelling language.
W	The Support should provide an additional description in formal ontology language.
D	The Support should cover key concepts in the MBSE life-cycle from requirements to V&V.
D	The Support should include explicit characterisation of autonomy.
D	The Support should not be restricted to limited types of autonomy.
D	The Support should include concepts of CPSs.
D	The Support should not be restricted to certain CPS domains.
D	The Support should be transferable to the MBSE framework.
W	The Support should include concepts relating to co-modelling and co-simulation.
Ergonomics:	
D	The Support should be usable by experienced systems engineers.
Rigour:	
W	The Support should be defined with a framework.
D	The Support should base its ontology on an existing standard when possible.

Table 3.1: Intended Impact Model for ACPS Ontology

3.4.2 Intended Introduction Plan

The Intended Introduction Plan is concerned with documenting the intended process so that the Support can be introduced, installed, used, and maintained by the intended end users. This section presents the Intended Introduction Plan for both of the Supports we intend to develop in this research.

ACPS Ontology. The main use of ACPS Ontology defined in this research is primarily targeted as a basis to build the MBSE framework for ACPS in Chapter 5. One of the requirements in the Intended Impact Model states that the Support should be described in an MBSE modelling language. This requirement ensures that ACPS Ontology can be readily adapted into an MBSE framework. How the ontology would be applied in the

Problem statement:	
Develop MBSE framework that supports the architecting of ACPSs.	
Performance:	
D	The Support must use the MBSE ontology for ACPS defined in Chapter 4.
D	The Support must be readily deployed with MBSE modelling language.
D	The Support must be focused on system architecture (as opposed to the low level implementation).
D	The Support must be an Architectural Framework.
W	The Support should provide framework description in MBSE modelling language.
D	The Support must facilitate the MBSE life-cycle from requirements to V&V.
D	The Support must be able to identify autonomy characteristics in ACPS.
D	The Support must incorporate the separation of concerns between autonomy and non-autonomy architecture.
D	The Support must be able to identify and capture autonomy requirements in ACPS.
D	The Support must be able to express the architecture of ACPS.
D	The Support must provide a method to support the V&V of ACPS.
D	The Support must not be restricted to limited types of autonomy.
D	The Support must not be restricted to certain CPS domains.
Ergonomics:	
D	The Support must be usable by individual experienced systems engineers.
Rigour:	
W	The Support should be defined with a framework.
D	The Support must reuse components from existing frameworks where applicable.

Table 3.2: Intended Impact Model for ACPSAF

framework is dependent on the framework itself. However, we can suggest three different ways of using this ontology in the MBSE framework.

1. Ontology as shared knowledge. The ACPS Ontology can be used as a basis to establish common knowledge of the terms and concepts used in the MBSE project. As MBSE activity tends to be a collaborative work involving multiple disciplines, the ontology will reduce the risk of miscommunication and misunderstanding in the project. We would suggest that the ACPS Ontology be modified to adapt to the circumstances of each individual project.
2. Ontology as an element of the MBSE framework. On top of being a platform for shared knowledge, the ontology can be directly integrated as part of the MBSE framework. In our work, the ACPS Ontology is an element of ACPSAF. This means that all concepts used in the framework must be included in the ontology. As we design both the ontology and ACPSAF using MBSE modelling language, we could even use tool support to check for model consistency and traceability.
3. Ontology as properties of the domain model. This type of application is dependent on the MBSE frame-

work as the framework should define how the ontology can be used in conjunction with the domain model. In our work, the domain model of ACPS is modelled in SysML. We introduce the elements in the ACPS Ontology as a Stereotype that can be added to the model. The Stereotype would define how an existing meta-class may be extended; however, this functionality would need to be defined by the framework.

ACPSAF. Parts of the Introduction Plan for ACPSAF are included within the framework itself. ACPSAF defines several Viewpoints indicating the purpose of the framework, the target users, and internal rules to using the framework. Here we include the Intended Introduction Plan that is not covered within the framework.

1. Target user: ACPS designer. *ACPS designer* is used loosely here to include anyone that has an interest to design, model, and engineer an ACPS. The term is not limited to a single individual, and there is no level of qualification required. However, understanding of the modelling language is required (see below).
2. Required knowledge: SysML. ACPSAF guides framework users to design an ACPS model with SysML as the modelling language of choice. ACPSAF assumes the user has knowledge of modelling with SysML as a prerequisite. The Viewpoints created in ACPSAF follow the convention of SysML 1.3. The latest version of SysML at the time of writing is SysML 1.5. Changes from SysML v1.3 to v1.5 are minor changes and do not affect the compatibility of ACPSAF. However, this is not guaranteed for the future version of SysML.
3. Required tools: Modelling tools for SysML. ACPSAF is designed to support the architecting of an ACPS model in SysML. To use ACPSAF, users would need to have a modelling tool that allows SysML models to be created. The only requirement for the SysML tool is that it must support all SysML 1.x diagrams. Community-maintained websites such as SysMLTools [1] maintain a short list of available SysML tools that readers can choose from.
   There are plenty of SysML tools available that satisfy our requirement, such as Rational Rhapsody, Enterprise Architect, and MagicDraw. For our work, the tool we have selected is Modelio [2] (open source version). The exact version is Modelio 3.7 with SysML Architect 3.7.02 module installed. Modelio is chosen here as it is a free, open source version option that still provides all the required features for the purpose of our work. Although the framework is not restricted to any particular modelling tools, note that different tools may render slightly different SysML graphics. As a result, SysML diagrams generated by tools other than Modelio may look slightly different to the SysML diagrams in this thesis.
4. Additional tools: Co-modelling and co-simulation tools. ACPSAF supports the use of co-model and co-simulation as a method to support verification and validation. If the user is interested in this feature, co-modelling and co-simulation tools will be needed. A list of tools that support co-simulation through FMUs are recorded in the fmi-standard website[3]. There are many eligible tools for the purpose of our work, such

[1]SysMLTools, https://sysmltools.com/, Last accessed: Aug 5, 2021
[2]Modelio, https://www.modelio.org/, Last accessed: Nov 6, 2019
[3]Fmi-standard, https://fmi-standard.org/tools/, last accessed: Nov 6 , 2019

as the OpenMedalica[4], one of the more mature modelling and simulation tools. The tool chosen for our work is the INTO-CPS[5]. This tool satisfies the main requirement of supporting the co-simulation of FMUs and is developed with primary focus on CPS.

We do not have strict requirements for the development tools of CT and DE based models as long as they support the FMI standard (i.e., being able to generate FMU models). In our work, we have chosen Overture[6] for the development of DE based models and 20-sim[7] for CT models as these tools have first-party integration support with INTO-CPS tool.

5. Disclaimer: ACPSAF is the result of an effort to advance the state of the art of MBSE for ACPS. It has not been accredited for industrial use and should not be treated as one without further formal evaluation.

3.4.3 Intended Support Description

Intended Support Description describes the Support in terms of the need or problem addressed, the goals and objectives of the Support, its elements, how it works, the underlying concept, theories, assumptions and rationale, and how it is to be realised. DRM does not specify a specific method for documenting an Intended Support Description. Therefore, our method to the Intended Support Description would be to answer the questions mentioned. Further elaboration on these descriptions will be covered in the chapters dedicated for each respective Support.

ACPS Ontology

- **Problem addressed:** The lack of MBSE Ontology that captures ACPS concepts.
- **Goal and objectives:** The goal is to create an MBSE Ontology that can be used in the MBSE framework to support the architecting of ACPS. Characterisation of autonomy needs to be explicitly defined, and relationships need to be drawn with MBSE concepts across the life-cycle, from requirements to V&V.
- **Elements:** Concepts from Ontology Engineering, MBSE, Autonomy, Systems Engineering, Requirements Engineering, Architectural Framework, Control Theory, Co-modelling, and Co-simulation.
- **How it works:** ACPS Ontology can be used in multiple ways. Its main use purpose in this research is to develop ACPSAF. However, refer to Section 3.4.2 to see several ways this ontology can be adopted for other purposes.
- **Concept and theory:** Same with Elements.
- **Assumptions:**N/A
- **How to realise:** A framework will be used to aid the development of the ontology. This is covered in detail in Chapter 4.

[4]OpenModelica, https://www.openmodelica.org/
[5]INTO-CPS, https://into-cps.org/
[6]Overture Tool, https://www.overturetool.org/
[7]20-sim, https://www.20sim.com/

ACPSAF

- **Problem addressed:** The lack of an MBSE framework that supports the architecting of ACPS.
- **Goal and objectives:** The goal is to create an AF that supports the architecting of ACPS through the MBSE life-cycle from requirements to V&V. This would include: Identification of autonomy requirement, architecture description to achieve autonomy requirement, traceability to co-model and V&V scenario for co-simulation.
- **Elements:** Perspective for AF, Requirements, CPS, Autonomy, and Co-modelling.
- **How it works:** ACPSAF provides principles and practices for creating and using the architectural description of ACPSs. Users should follow the instructions provided in each Viewpoint as that would help guide them in developing the ACPS model.
- **Concept and theory:** MBSE, Autonomy, Systems Engineering, Requirements Engineering, Architectural Framework, Control Theory, Co-modelling, and Co-simulation.
- **Assumptions:** We assume the user has knowledge of SysML and access to modelling tools.
- **How to realise:** A framework will be used to aid the development of ACPSAF. This is covered in detail in Chapter 5.

3.5 Summary

In this chapter, we have defined the scope of our research and use it to find a suitable methodology based on the nature of our research. As DRM is chosen for the research methodology, our research will be completing the four research stages specified by DRM: Criteria Definition, Descriptive Study I, Prescriptive Study, and Descriptive Study II. A review-stage research is carried out for Criteria Definition and Descriptive Study I, and the summary is included in this chapter. For Prescriptive Study, a Documentation of Intended Support is completed in this chapter, while realisation of the Support will be carried in the following thesis chapters. An initial stage will be carried out for Descriptive Study II and be covered in Chapter 6 in the form of case studies.

4 MBSE Ontology for ACPS

This chapter proposes an ontology for ACPS to be used in the context of MBSE activities. The approach focuses on ‘Ontology’ as described in Section 2.4 and will be used as a basis to build the framework in Chapter 5. The first two sections of this chapter are a preliminary introduction to the ontology engineering process that we follow as part of the ACPS ontology creation. Section 4.1 provides a brief literature review on the typical development stages to ontology engineering. The methodology chosen to build the ACPS ontology is covered in Section 4.2, and the stages of building the ontology are categorised into ontology requirements specification and lightweight ontology building. Section 4.4 covers the ontology requirements specification stage, which includes the definition of scope, purpose, intended use, and competency questions as ontology requirements. The lightweight ACPS ontology is then presented in five sections, organised based on its domain of interest: requirements engineering in Section 4.4, systems engineering in Section 4.5, extended concepts (e.g., machine learning, system-of-systems) in Section 4.6, co-modelling and co-simulation in Section 4.7, and architectural framework in Section 4.8. This chapter is then closed with an evaluation that includes ontology verification against requirements in Section 4.9.

4.1 Literature Review on Ontology

Ontology is a branch of philosophy dealing with fundamental concepts such as reality and existence. Within computer science, it has come to refer to a representation of concepts, entities, their relationships, and their inter-dependencies [288]. AI researchers in the 1970s recognised that the key to developing a powerful AI system lies in the development of knowledge engineering [289, 290], which concerns itself with the aspects involved in building, maintaining, and using knowledge-based systems [291, 292]. Application of knowledge-based systems at the time were primarily in the form of expert systems [293] but has since grown into a wider range of applications such as the semantic web [294, 295]. It is through this process that the knowledge engineering community began to use the term ’ontology’ to refer to both the theory of a modelled world and a component of knowledge-based systems [296, 297].

There have been many attempts to provide a formal definition to ontology. Some of the most cited definitions are proposed by Gruber [298], in which ontology is defined as an 'explicit specification of a conceptualisation', and by Borst [299] as a 'formal specification of a shared conceptualisation'. Studer et al [300] combines the two former definitions, defining it as a 'formal, explicit specification of a shared conceptualisation'. However, these definitions often leave readers with further questions, such as what 'specification' and 'conceptualisation' truly mean in this specific context. This can be seen from the effort by Guarino et al. [301] to clarify and formalise those keywords used in the definition of ontology. More informally, ontology can be seen as a set of definitions and relationships of concepts within a domain. Ontologies are typically human-intelligible and machine-interpretable, and they are often used as standardisation of terminology for a group or community.

Aside from its formal definition, it is equally important to understand the purpose of an ontology. Traditionally, an ontology is developed as part of a knowledge engineering process [302] to build knowledge-based systems (or expert systems) that attempt to emulate the behaviour of a human expert in a given field. An example of this would include Semantic Web as its primary goal is to make internet data semantically machine readable. We refer to this particular method of ontology usage as 'ontology for machine'.

The use of an ontology is not limited to knowledge engineering. Uschold and Gruninger [303] state that an ontology would improve communication between people and interoperability between systems as many of these issues stem from the lack of a shared understanding. In a project where people from different backgrounds must work together, the use of different jargon, concepts, and viewpoints can contribute to poor communication and interoperability. A study carried out by NIST estimates that an annual cost of \$15.8 billion towards the capital facilities industry in 2002 is attributed to inadequate interoperability [304].

In this context, an ontology is seen as a way to define an explicit and unambiguous terminology that promotes interoperability primarily between humans. For example, an ontology approach was used to create information models of SE processes based on a process description in ISO 15288 [305]. Compared to 'ontology for machine', this ontology is not meant for knowledge engineering of a machine. Instead, it is for people to produce systems in an explicit and consistent way; in other words, it is an 'ontology for human'.

There is a lack of work within the literature that formally distinguishes ontology for machines and ontology for humans. However, the two types of ontology have different target users. As the name implies, ontology for machines typically develop ontologies to be used by a knowledge-based system. Therefore, the ontology has to be machine readable. From extreme points of view, it is theoretically possible to develop a successful ontology that is totally meaningless to the domain expert as long as it is usable with the intended AI system. This is possible because methodology in ontology engineering typically advises that the ontologies must be defined by ontology experts rather than domain experts. This is because ontology experts would interview numerous domain experts, interpret meanings between different terms used, and construct a machine interpretable ontology. Meanwhile, ontologies for humans are concerned with the collaboration and interoperability between humans. Therefore, this means that it is also theoretically possible to develop a successful ontology that is not machine readable at all. As the two types of ontologies have different purposes, their engineering processes differ as well.

Ontology is defined using ontology languages, which are typically formal languages. This is especially important for machine ontology as the machine needs to be able to understand the semantics of the ontology [301]. Nevertheless, a less formal language (e.g UML, SysML) can still be used when it comes to ontology for humans. Following the structure of the language, ontology language can be classified [306] as:

- Logical language
  - First-order predicate logic (e.g Common logic [307])
  - Rule based logic (e.g RIF [308])
  - Description logic (e.g OWL [309])
- Frame based (e.g F-logic [310])
- Graph based (e.g IDEF5 [311], OntoUML [312])

The process of developing an ontology has been heavily researched and is often referred to as ontology engineering. Ontology engineering is formally defined as the 'set of activities that concern the ontology development process, the ontology life cycle, and the methodologies, tools and languages for building ontologies' [313].

There are numerous existing methodologies for ontology engineering. On-To-Knowledge [314], Ontology Development 101[315], and DILIGENT [316] are some examples. Each methodology has its own strength and limitations, and there are studies that provide comprehensive analysis of the mentioned methodologies [317, 318, 319].

METHONTOLOGY is a methodology to build ontologies from scratch [320]. It defines the set of activities that need to be carried out to build an ontology. This includes the identification of the ontology development process (life cycle based on evolving prototypes) and some techniques to carry out management, development-oriented procedures, and support activities. The activities to building an ontology as suggested by METHONTOLOGY can be summarised as follows:

- **Specification**
  - The purpose of the ontology, including its intended uses, scenarios of use, end-users, etc.
  - Level of formality of the implemented ontology, which depends on the formality that will be used to codify the terms and their meaning. (e.g., SysML is 'semi formal' following the definition from [303])

  - Scope, which includes the set of terms to be represented, its characteristics, and granularity.
- **Knowledge Acquisition.** (e.g text analysis, interview with experts)
- **Conceptualisation.** (Glossary of terms)
- **Integration** with existing ontologies to prevent re-inventing the wheel.
- **Implementation.** Creation of the ontology using the ontology language.

METHONTOLOGY is one of the most cited methodologies for ontology building. A survey [321] carried out in 2009 points out that approximately 20% of ontologies from their survey data set were developed using it. Nevertheless, there have been concerns raised about METHONTOLOGY being too outdated to address newer challenges as the practice of ontology engineering spreads to a wider range of domains and applications. In particular, Suárez-Figueroa et al. [322] mention that METHONTOLOGY lacks the notion of collaboration and a detailed guide to manage versions. It also mentions that METHONTOLOGY lacks support for the re-use and re-engineering of existing ontologies into the new ontology. NeOn addresses these limitations by providing a scenario-based methodology. Several scenarios are suggested, which take into account the possibility of: creating the ontology from scratch, reusing, re-engineering, and merging both ontological and non-ontological resources.

Although there are various differences between ontology engineering methodologies, there are key activities that appear in most of them. Simperl et al. [321] summarises the key activities of ontology engineering as follows:

- **Ontology Management**, which concerns itself with the initial feasibility study to define the problems, opportunities, potential solutions, and economic feasibility of the project.
- **Ontology Development and Support**, which can be further refined into the following activities:
  - **Domain analysis** which defines the motivation scenario and competency questions.
  - **Conceptualisation** stage, which defines the relevant concepts, integrating with existing solutions.
  - **Implementation** of the formal model using an ontology language.
- **Ontology Use.**
  - **Maintenance** is carried out to adapt the ontology to new requirements or changing understanding of the domain knowledge.

One important aspect to explore is the use of Competency Questions (CQs) as a way to capture the requirements of the ontology. Based on the formality of the CQs, it can be used to support ontology development in a different way. Firstly, CQs can be defined formally to be used as a test to check the ontology against requirements written in natural language. These type of CQs are typically interrogative sentences that target the ontology classes and their relations [323, 324], and are created before the development of the ontology to be used as a form of functional requirement. The CQs can be defined in specific patterns such that CQ validation can be done automatically with tools [325]. These CQs are best utilised after the ontology has been implemented to validate its correctness.

The second application of CQs is when they are defined more informally as a way to describe what is required for the ontology to achieve. This type of CQ is gathered at an early stage of development and is used as a guide to what should be included in the ontology. As the CQs can be defined informally, they allow room for descriptive questions. This is especially useful when the requirement of the ontology depends on the input from multiple stakeholders (e.g., ontology users). Depending on the nature of the questions, some informal CQs can be validated against the ontology as well. Similar to the formal ontology, questions that can be validated are typically ones that target the ontology components and its relationship. CQs will be used in our research as a way to capture requirements for the ACPS ontology.

4.2 Methodology for Ontology Engineering

It is important that we identify our requirements to justify our choice of methodology. By understanding what type of ontology we would like to produce, we can avoid using an outdated or unsuitable domain-specific methodology. The requirements our methodology must support are as follows:

1. **Domain ontology.** Ontologies can be categorised into domain ontologies or upper ontologies. Domain ontologies represent concepts and relations that are domain-specific, while upper ontologies represent common concepts and relations used across a wide range of domains. For example, the term *model* can be commonly found across a wide range of domains. However, we are only defining terms in the context of MBSE for ACPS. This means that the ontology we develop will be a domain ontology, and the methodology we use must support the definition of domain level ontologies.
2. **Middle-out approach.** A top-down approach in ontology engineering is one in which complex questions are decomposed into simpler ones, while a bottom-up approach [326] uses simple questions that are organised to form complex ones [303]. We favour the middle-out method that mixes between the top-down and bottom-up approach. This is because the ontology building will start with a few key concepts that must be included (e.g., CPS, Autonomy) that could be either top or bottom level concepts. These concepts would need to be abstracted (bottom-up) or refined (top-down) as necessary, so the methodology must support the combination of both.
3. **Semi-formal, lightweight ontology.** Lightweight ontologies are used to illustrate the terms and relations in an ontology using a widely accepted and easily understandable format such as SysML. As our goal is to define an ontology for an architectural framework, we do not intend to implement the complete, formal ontology. This means that the methodology we select must support separation of concerns between lightweight and formal ontology, preferably with explicit support for the definition of lightweight ontology.

Based on our requirements, we would be employing a methodology proposed by Hildebrandt et al. [327]. Hildebrandt's methodology satisfies the first criteria as it is developed specifically to support the ontology design in the domain of CPS. It also satisfies the second criteria as the methodology proposes four mechanisms to extend terminology in the ontology that works as a middle-out approach. The methodology has the notion of lightweight and heavyweight ontology, and explicitly defines them with two separate processes which fulfils the third criteria. Hildebrandt's methodology for ontology engineering is defined into three stages:

1. **Ontology Requirements Specification.**
   - Definition of scope & purpose. This step is concerned with understanding the goal of the ontology to be built and is covered in Section 4.3.
   - Identification of intended uses. Similar to the previous point, this is concerned with the scope of the ontology and is covered in Section 4.3.
   - Competency Questions (CQs) as requirements. Requirements for the ontology are recorded in the form of CQs. The CQs are initially recorded in Section 4.3.
   - Grouping and prioritisation of requirements. There will be no prioritisation of requirements. However, the CQs will be grouped by its domain in Section 4.3.
2. **Lightweight Ontology Building.**
   - Search for information resources. The information resources used to define the ontology is covered in Section 4.4 to 4.8. The resources are spread out across five sections that are organised based on the domain of interest.
   - Building ontology based on information resources. The ACPS Ontology is defined and presented together with the information resources in Section 4.4 to 4.8. Hildebrandt suggests the use of semi-formal language such as UML for the definition of lightweight ontology. Our language of choice for the lightweight ontology is SysML. SysML is chosen over UML because the ontology will be used in the context of MBSE. On top of that, all of the basic UML components needed to describe the lightweight ontology also exist in SysML, so there is no issue with using SysML.
   - Verification of ontologies (to requirement). Ontology will be verified against the CQs in Section 4.9.
3. **Heavyweight Ontology Building.** Heavyweight ontology is beyond the scope of our research, as the lightweight ontology is sufficient for the goal of the ontology (i.e., to be used as part of MBSE framework). However, this may be part of future work, so we conduct some preliminary work towards the heavyweight ontology here.
   - Understanding the problem domain and building of heavyweight ontology. Heavyweight ontology is defined using a formal language. The XML document representing the heavyweight ontology for ACPS written in OWL is covered in Appendix A.
   - Matching and verification of ontology. A heavyweight ontology can be verified by running the ontology

against SPARQL queries for each CQs. This step is not covered in our research.

4.3 Requirements Specification

This section defines the Ontology Requirements Specification for the ontology building approach by Hildebrandt et al. [327]. The activities of the Ontology Requirements Specification are used to define the scope and purpose of the ontology, identify its intended use, build requirements by using competency questions, and group the requirements. Note that in the actual sequence of ontology development, we should first identify all existing ontologies before developing the CQs and designing the new ontology. However, we decide that instead of separating the existing ontology into earlier sections, the introduction to existing ontologies and the selected definition for our ontology will be delivered in the same section (Section 4.4 to 4.8) as it helps readers to see the reasoning behind the selected definition.

- **Purpose:** Ontology about ACPSs to be used with an AF to support the MBSE of ACPSs.
- **Scope:** Autonomy, ACPS, MBSE, Requirements engineering, Co-modelling, and Co-simulation.
- **Level of Formality:** Semi-formal (SysML).
- **Target Group:** Systems engineers that are interested in designing ACPS from scratch or introducing autonomy into existing CPSs (i.e transforming CPS into ACPS) with an MBSE approach.
- **Intended Uses:** Ontology is intended to be used as a basis to define an AF to support MBSE for ACPSs. The ontology will be used to define a framework in Chapter 5. However, readers can define their own separate ACPS Framework using this ontology as well.
- **Competency Questions (CQs).**
  - **Architectural Framework**
    - What are the elements making up an AF?
    - What is the relationship between AF and Viewpoint?
    - What is the relationship between AF and View?
    - What is the relationship between AF and Ontology?
    - What is the relationship between AF and System?
  - **Requirements**
    - What is the relationship between Requirement and Need?
    - What are the elements needed to describe a Need?
    - What is the relationship between Need and System?
  - **CPS**
    - What are the elements making up a System?
    - What are the elements making up a CPS?
    - What is the relationship between CPS and Environment?

- **ACPS**
    - What are the elements making up an ACPS?
    - What is the relationship between ACPS and Autonomy?
    - What is the relationship between Autonomy and Autonomy Requirement?
    - What are the roles of Controller in ACPS?
- **Co-modelling and Co-simulation**
    - What are the elements making up a Co-model?
    - What are the elements making up a Co-simulation?
    - What is the relationship between Co-model and Co-simulation?
    - What is the relationship between FMU and Co-model?
    - What is the relationship between FMU and Co-simulation?
    - What is the relationship between Co-model and System?

- **Pre-Glossary of Terms.** AF, Viewpoint, View, Ontology, System, CPS, Environment, ACPS, Autonomy, Autonomy Requirement, Controller, Co-model, Co-simulation, FMU.

The list above presents the Ontology Requirements Specification for the ACPS Ontology that we propose in this chapter. It defines the purpose, scope, formality of language, target group, intended use, competency questions, and pre-glossary of terms. The defined ontology must be sufficiently completed to answer all of the competency questions (i.e., it must at least define all concepts mentioned in the competency questions in order to be able to answer the questions). The pre-glossary of terms indicate the special terms used in the requirements that would exist in the ontology. Typically, the completed ontology would include more terms than the ones identified here. However, the terms here indicate that they are the core concepts of the ontology and serve as a starting point.

Assumption Most ontologies share structural similarities in terms of the components making up the ontology, which are individuals, classes, attributes, and relations. Classes and relations in our ontology closely relate to the typical ontology. However, our ontology will differ in its terms of other components:

1. **Individuals/instance.** In the context of ontology, Individuals represent instances or objects. Deciding whether something is a concept of an instance is often dependent on the context of the ontology. For example, METHONTOLOGY [320] was used to produce an ontology in the domain of chemistry that contains Individuals in the forms of 103 elements of substances, where an example of Individual would be helium under the type of noble gases. Helium in itself is not an definite example of an Individual as it can be turned into a class to classify all known types of helium isotopes. However, in the context of [320], helium is clearly meant to be used as an Individual. Our ontology will not contain any Individual, as our ontology is not intended to be the full list of taxonomy or classification itself. Our ontology is meant to be used with the framework that aids framework users to do the classification themselves. That means our

ontology would provide concepts such as ACPS, but would not provide any Individual that is under the classification of ACPS itself. It is up to the framework users to either identify or build a system that would fit into the classification.

2. **Attributes.** Attributes represent the properties that Classes may have and is an important aspect of an ontology. However, our ontology will not explicitly contain any attributes and will leave it to be defined in ACPSAF instead. The motivation behind this approach is that unlike other ontologies that are established as a standard, MBSE ontology must usually be adapted to the context of the SE activity to be useful. Consider a class called 'Requirement', basic attributes to this class could be 'id' and 'description'. However, each MBSE project might require different additional attributes to be recorded, such as 'priority' and 'status'. The desirable attributes depend on the needs of the stakeholders, and there is no single correct way of recording these attributes. Allowing the attributes of the ontology to be captured in the MBSE framework has its own advantages and disadvantages. The biggest advantage to this approach is as mentioned: attributes in MBSE ontology typically have to be adjusted with the needs and context of each individual project. Adding an attribute directly in the ontology definition would limit the applicability of the ontology. The disadvantage of this approach is that the ontology attributes need to be defined before the ontology can be utilised effectively. This means that the ontology we propose should only be used with ACPSAF, or after new attributes are introduced for each new MBSE project.
3. **Restrictions and rules.** An ontology is only beneficial when there is a clear method to using it. Restrictions and rules enforce how the ontology should be used. However, restrictions and rules of our ontology will be covered as a part of the accompanying architectural framework instead. When an ontology is produced as a standalone artefact, rules and restrictions are placed as part of the ontology. As our ontology is meant to be strictly used with the accompanying framework, we consider the rules as part of the framework so that rules can be modified separately from the ontology. This means that the framework and its rules can be changed to fit the needs of the framework users without having to interfere with its underlying ontology.

4.4 Requirements Modelling

The first step to designing any system successfully is to understand and capture the requirements correctly. The concern of defining, documenting, and maintaining requirements in the engineering design is a well-established domain of Requirements engineering (RE) process. It is important that we consider existing work so that we do not "re-invent the wheel". We consider standards for Requirements engineering such as ISO/IEC/IEEE 29148 and approaches to designing requirements [328], [329], [330], [331], [332], [333], [334] and [335].

The approach that we choose as the basis for our work is the Approach for Context-based Requirements

Engineering (ACRE)[335]. This was chosen for a number of reasons:

- MBSE approach. The ACRE follows the fundamental MBSE approach in the sense that ontology is formally defined as a basis for views to make up the complete set of requirements.
- Flexibility. ACRE is not designed for a specific system or domain, so it can be used and extended for ACPS. It is also applicable from small to large project scale and non-critical to mission critical systems.
- System-of-Systems application. CPSs have the inherent nature to be a SoS, and ACRE has been adapted for the use of SoS. Although this is not the most important selection criteria, proven work of compatibility to SoS (referred to as SoS-ACRE [336]) means that we can readily expand our work to address SoS by adopting SoS-ACRE.

The following section covers our Ontology for Requirements modelling that is based on ACRE, where minor modification will be made to help capture ACPS requirements. This section will capture the full ontology instead of just the modified parts as we intend for this chapter to fully describe all of the ontology that might be used in the framework in Chapter 5.

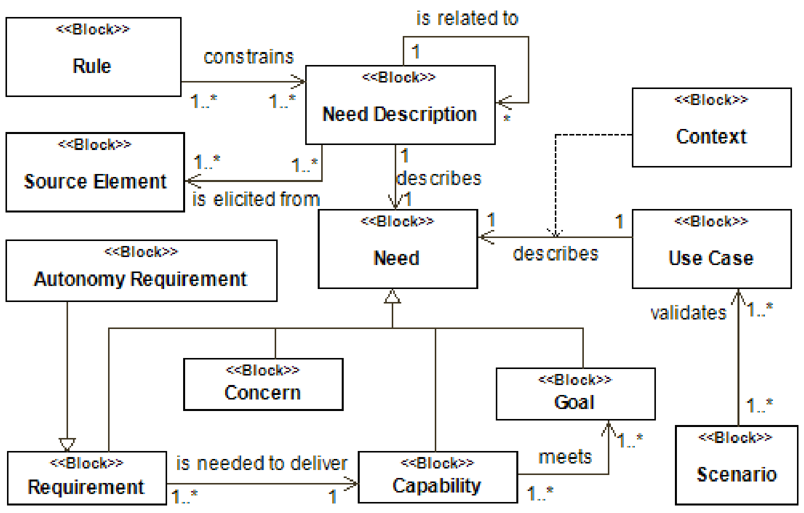


Figure 4.1: Ontology focused on Requirements modelling

The diagram in Figure 4.1 shows that there is a concept of Need at the centre of the ontology. Need is then composed into four types: Requirement, Concern, Capability, and Goal. One or more Goals can be met by one or more Capabilities, and each Capability is delivered by one or more Requirements. There is also a type to Requirement, which is the Autonomy Requirement. Need Descriptions are required to describe each need and are elicited from Source Elements. There are also rules that constrain the definitions of need descriptions. The Context of each Need is described with Use Case and can be validated with Scenario.

The following list describe a short summary of each element and is followed with sections for each elements with more detailed descriptions:

- ‘Need’ - a generic abstract concept that represents something that is required or desirable for the subject of the ‘Context’. ‘Need‘ is the purpose that the ‘System‘ must deliver to the ‘Stakeholder‘ who deploys the ‘System‘.
- ‘Context’ - a specific point of view based on certain characteristics. For example, Stakeholder role, Organisational unit, System hierarchy.
- ‘Goal’ - a type of ‘Need’ whose ‘Context’ usually represents one or more Organisational units. Each ‘Goal’ will be met by one or more ‘Capability’.
- ‘Capability’ - a special type of ‘Need’ whose ‘Context’ will typically represent one or more Project or Organisational units. A ‘Capability’ will meet one or more ‘Goals’ and will represent the ability of an Organisational unit.
- ‘Requirement’ - a property of a ‘System’ that is either required or desired by a ‘Stakeholder’ or other form of ‘Context’. One or mode ‘Requirements’ are needed to deliver each ‘Capability’.
- ‘Need Description’ - a concrete description of an abstract ‘Need’ that has a predetermined set of attributes.
- ‘Source Element’ - the origin of a ‘Need’ that is elicited into one or more ‘Need Description’. A ‘Source Element’ can be anything that inspires and affects a ‘Need’. Examples include a Standard, Project documentation, a System, a publication, an email, and a book.
- ‘Rule’ - constrains the attributes of a ‘Need Description’. A ‘Rule’ can be expressed in many different forms such as equations, reserved words, and grammar restrictions.
- ‘Use Case’ - describe the functionality or behaviour of a system in terms of how it is used to achieve the ‘Need‘ of the various users. It describes what action goes into a specific scenario of use and what outputs are achieved.
- ‘Scenario’ - an ordered set of interactions between one or more ‘Stakeholder’ roles, ‘System’ or ‘System Element’ that represents a specific chain of events leading to a specific outcome. One or more ‘Scenarios’ validates each ‘Use Case’.

4.4.1 The Need Concept

The Need can be considered an abstract concept that describes a Requirement, Capability, Goal, or Concern. Every System that has a purpose will have a set of Needs, regardless of whether they are clearly defined or simply a concept in a person’s mind. It is important to note that the artefacts we produce to describe Needs are not the Need itself, but rather, the representation of the Needs. This is because a Need is abstract and should not be a tangible entity. Holt and Perry [240] argue that a well-defined Need should be identifiable and verifiable, but should also be non-solution specific. A Need may be represented in a multitude of ways, and there is no conclusive classification on types of needs as it is likely to differ depending on the projects, domains, and company. In another word, ’Need’ is the identified needs of the stakeholders that have caused them to be interested in the system development.

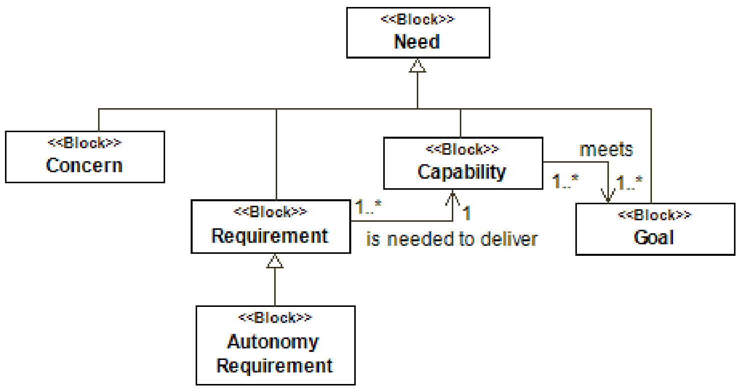


Figure 4.2: Ontology focused on Requirements modelling showing types of Need

The diagram in figure 4.2 shows that there are four types of Needs, which are:

- ‘Goal’ - a type of ‘Need’ whose ‘Context’ usually represents one or more Organisational units. Each ‘Goal’ will be met by one or more ‘Capability’.
- ‘Capability’ - a special type of ‘Need’ whose ‘Context’ will typically represent one or more Projects or Organisational units. A ‘Capability’ will meet one or more ‘Goals’ and will represent the ability of an Organisational unit.
- ‘Requirement’ - a property of a System that is either required or desired by a Stakeholder or other form of Context. One or more ‘Requirements’ are needed to deliver each ‘Capability’.
- ‘Concern’ - is a unique type of Need used in representing an Architecture, a Viewpoint, or an Architectural Framework. It differs from the other types of need as it is used to capture the need of ACPSAF rather than the needs of ACPS. This concept is introduced in our ontology for consistency with FAF ontology, which is used in the making of ACPSAF.

For the purpose of our work, we provide a further classification to the Requirement called the Autonomy Requirement [31, 34, 30].

- ‘Autonomy Requirement’ - a type of ‘Requirement’ that is used to describe Needs in relation to the ACPS’ ability to achieve a certain goal with little to no intervention from a human operator. ‘Autonomy Requirement’ in ACPS is a non-functional requirement that typically involves the ability to measure the environment with a sensor and influence it with an actuator as a result of adaptation in order to satisfy the requirement. The method to identify Autonomy Requirement will be part of ACPSAF.

4.4.2 The Need Description Concept

Since a Need is an abstract concept, it is important to understand that when we attempt to describe a Need, the description does not equate to the Need itself. This description that represents a Need is called a Need Description. It is important that we distinguish between a Need and Need Description because a Need can only be given a meaning by putting it into a Context. While each Need must have a Need Description, understanding a Need Description does not mean that the Need is understood. This is because a Need Description is a high-level description of a Need.

4.4.3 The Source Element Concept

Source Element represents the origin and source of a Need. Needs can be abstracted from anything, but it is important that we establish some form of traceability of what the sources are. Source Element can be from anything, and some examples include requirement documents, conversations, standards, and specifications. The key point would be that while Source Elements can be taken from almost anything, it must be identifiable for it to be usable. In our framework, we would achieve this by attaching some information to each Source Element, such as its version number and location of its record.

4.4.4 The Rule Concept

There is a tendency for ambiguity and misinterpretation when attempting to capture abstract concepts. This is one reason why the systems engineering domain has moved from a document-centric approach to model-based approach as a step to mitigation. Although modelling languages provide improved clarity in comparison to natural language, there is still plenty of room for ambiguity. To minimise this issue, we introduce a number of Rules that may be defined and applied when describing Need Descriptions. Any types of Rules are permissible, and some example of these rules would be:

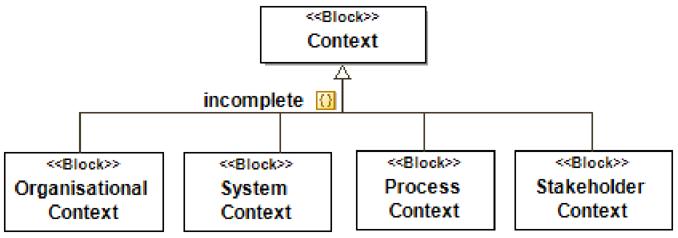
- Rules on the wording used to define a Need Description. Some common examples of these rules would be to restrict the Need Descriptions from using ambiguous words such as ‘approximately’ and ‘should’.
- Constrain Need Description through the value of its attribute. A Need Description could have multiple attributes on top of its textual description. Some examples of additional attributes would be ID, priority, and owner. These values can be constrained by applying a specific pattern or even an enumerated list.

4.4.5 The Use Case Concept

Use Cases are scenarios of use which describe the functionality or behaviour of a system in terms of how they are used to achieve the ‘Need‘ of the various users [337]. Use Cases are commonly used in many different methodologies such as the Object-Oriented Systems Engineering Method (OOSEM) [337], capturing the system requirements as it provides details and clarity that a text requirement such as Need Description cannot express by itself [338]. Identifying Use Cases typically involves identifying actors that use and participate in the Use Cases, and the system that owns and performs the Use Cases [339]. This combination of actors and system will be described as the Context forming each specific Use Case.

4.4.6 The Context Concept

The idea of Context is quite important as it is the way we transition the abstract concept of Needs into something meaningful and measurable. Figure 4.3 shows several types of context - Organisational Context, System Context, Process Context, and Stakeholder Context. The diagram also shows that the shown types of Context are incomplete. This is to highlight that users that use our ontology should not be limited to our examples and should add and adapt the types of Context according to their need. We will not go too deeply into each type of Context shown as they are highly customisable, and will only briefly discuss how each type of Context could differ from others.



<<Block>> Context
incomplete
<<Block>> Organisational Context
<<Block>> System Context
<<Block>> Process Context
<<Block>> Stakeholder Context

Figure 4.3: Ontology focused on Requirements modelling showing types of Concept

For our work, we are only interested in two types of Context that will also be used in our framework: System Context and Stakeholder Context. The Stakeholder Context looks at Needs from the points of view of Stakeholder Roles. Stakeholder Roles could be those of a person or even an organisation that is affected or interested in the system. Context looks at Needs from the points of view of system hierarchy. When a system is broken into subsystems and components, there could be a number of different Needs for each of the different levels of system hierarchy.

The structure of system hierarchy will be covered in Section 4.5, but it is worth noting here that our framework will be considering Needs from the context of Controller. Controller in our model is the element in the system that is responsible for enabling autonomous capability, so this example would fall under 'System Context'. As we are designing autonomous CPS, it is important to consider the requirements from the perspective of the Controller.

Being able to identify the various Stakeholder Roles and the structure of the System hierarchy is key to getting Context right. Some general facts should be kept in mind when considering Contexts:

- The different context elements identified (i.e., Stakeholder roles, System hierarchy) would form the basis for defining the number of Contexts, and each of the elements will have their own point of view or Context.

- The steps of action taken in a System with Context would form the scenario of use, which is the Use Case.
- All Context will form a model, so it must be consistent with one another even if there is a possibility of conflict with each other as they represent different points of view with possibly different interests.

4.4.7 The Scenario Concept

The scenario of use demonstrates that a System could follow a series of actions or behaviours that would lead to a desired outcome as described in the Need Description. Figure 4.4 shows two types of Scenarios, which are Semi-formal Scenarios and Formal Scenarios.

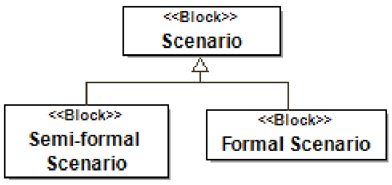


Figure 4.4: Ontology focused on Requirements modelling showing types of Scenario

Following the ACRE approach, Semi-formal Scenarios can be realised with a SysML sequence diagram, showing the System and the interactions between its elements. The Scenario interactions could happen at Stakeholder-level and System-level Scenarios. At Stakeholder-level Scenarios, the focus would be to investigate the interactions between the System and Stakeholder Roles. At System-level Scenarios, the focus would be on the interactions between System elements in the System. The Formal Scenarios are used with mathematical approach to analyse the Use Cases. This can be realised with a SysML parametric constraint block in the constraint diagram. Since our approach uses the modelling and simulation tools, we can also use simulation-based validation that are based on these scenarios. The co-modelling approach provides a powerful method to combine both Semi-formal and Formal Scenarios. This is because the simulation can be set up to run the interactions of Semi-formal Scenarios, and the model checking tools would enforce the correctness of the Formal Scenarios.

4.5 ACPS Modelling

4.5.1 The System Concept

The concept of ‘System’ plays a very important role in the MBSE methodology as it forms the basis for systems engineering and MBSE ontology. The International Organization for Standardization (ISO) states the definition of a system in ISO 15288 [233] as a product and/or service to provide product(s) and/or service(s) for the benefit of users and stakeholders.

In terms of structural composition, a system is composed of a set of interacting system elements that are implemented to achieve their respective purpose or requirement. It also states that “a system’s elements can be hardware, software, data, human, processes ... , procedures ..., facilities, materials, and naturally occurring entities”. Figure 4.5 shows the summary of System-related concepts from ISO 15288 visualised with SysML.

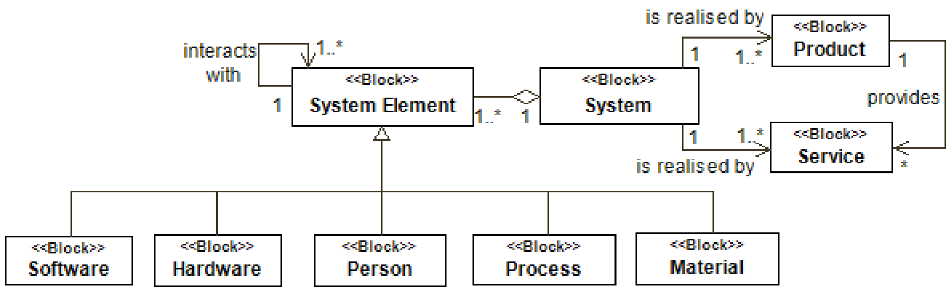


Figure 4.5: SysML diagram summarising the System-related concept from ISO 15288

The definition for system concept provided by the ISO puts forward a strong baseline as system-related concepts defined by other institutions tend to have similar characteristics. For example, the ‘INCOSE systems engineering handbook’ [83] is built on top of ISO 15288. While the US Department of Defense [340] uses a slightly different terminology, the primary characteristic of a system being composed of multiple interacting system elements still persist: “A functionally, physically, and/or behaviourally-related group of regularly interacting or interdependent elements; that group of elements forming a unified whole”.

While we intend to adopt the definitions from ISO 15288 as much as possible, we propose some modification so that the System-related concept is fit for purpose in the context of our framework to support the design of ACPSs.

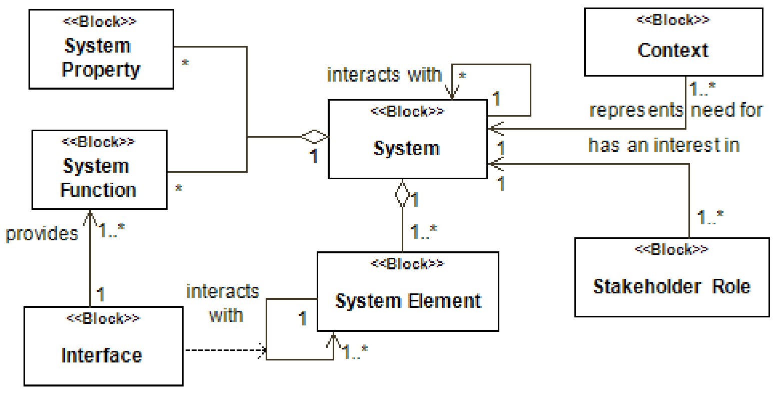


Figure 4.6: System related concept for CPS modelling

Figure 4.6 shows that there is a concept of ‘System’ at the centre of the ontology. Similar to the ISO 15288, the ‘System’ is composed of several interacting system elements. However, we have removed ‘Product’ and ‘Service’. The concept of Product and Service are typically used in relation to the modelling system life cycle and processes. The concept of process is an important one in systems engineering as it is commonly used to capture the activity to produce an artefact that makes up the system. Process modelling for MBSE deserves its own dedicated section. However, we will not explore this in our work as we will focus on the structure and behaviour of the system itself.

Note that in our research, we will be omitting the concept of ‘Process’ from our ontology. This would mean that our MBSE ontology is incomplete for the purpose of full MBSE activity, but this can be resolved by addressing the missing concepts using existing MBSE works from other sources such as MBSEAF [240]. This issue regarding incomplete ontology will be discussed further in Section 4.9 and highlighted as further high-priority work in Section 7.3.

Although we will not cover the concept of ‘Process’ in our ontology, we have a ‘Stakeholder Role’ that is usually covered in the ‘Process’ modelling. The ‘Stakeholder Role’ is usually a person, entity, or organisation that has an interest in the ‘System’. Our ontology requires the ‘Stakeholder Role’ because at the very least, our framework describes the activity of a person designing ACPS. Therefore, the ‘Stakeholder Role’ will be used to represent the people involved in the design activity.

The types of ‘System Elements’ from ISO 15288 are also removed because they are too generic and are an abstraction that could be instantiated in more than one specific kind in the context of a specific system. We propose a more specific type of ‘System Element’ for ACPS, and this will be covered in the next section when we discuss about CPS. The diagram also shows that the system is made of ‘System Property’ and ‘System Function’. It also shows that an ‘Interface’ provides one or more ‘System Functions’ and is used to enable interaction between system elements.

‘System Element’ could also be a ‘System’ depending on the level of abstraction. Consider Figure 4.7 showing an example model of a smart grid system. At the top level, ‘Smart Grid’ is considered a ‘System’ and is composed of three ‘System Elements’: ‘Generator Subsystem’, ‘Transmission Subsystem’, and ‘Distribution Subsystem’. However, when we decompose the subsystem which is a ‘System Element’ to the ‘Smart Grid’, the subsystem can be considered a ‘System’ in its own right. This is shown in the diagram where ‘Generation Subsystem’ is a ‘System’ to its ‘System Elements’: ‘Generator’, ‘Controller’, and ‘Power lines’.

The need to work and model at different levels of abstraction in this manner is common in systems engineering. This is not to be confused with system-of-systems, which is a special collection of systems that deliver a unique functionality not deliverable by any single system composing the SoS. To capture the emergent behaviour of SoS, special concepts need to be introduced into the ontology and will be discussed in Section 4.6.

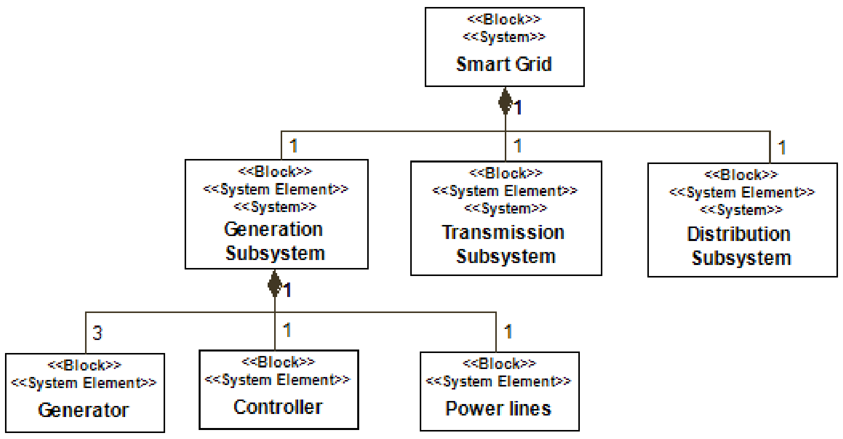


Figure 4.7: Example model showing the use of 'System' and 'System Element'

4.5.2 The CPS Concept

There are multiple definitions of CPS with varying degrees of differences. However, it is commonly accepted that CPSs are the integration of computation, networking, and physical processes where physical processes are monitored and controlled by embedded computers typically with feedback loops, and these physical processes affect computations and vice versa [2]. Our approach to integrating new concepts into our existing MBSE ontology is to pick out primary keywords from the new concept and attempt to connect them to the existing ontology. The keywords making up the CPS from [2] are computation (i.e Cyber) and physical.

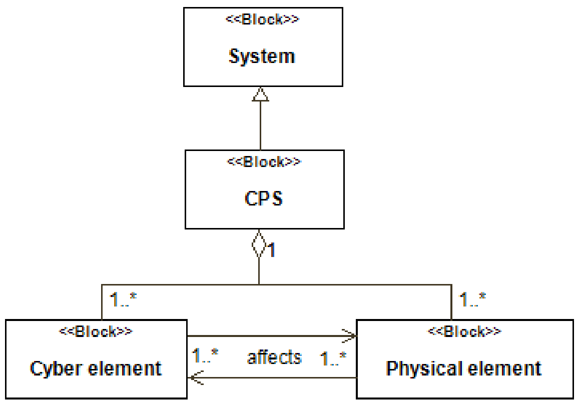


Figure 4.8: SysML diagram summarising parts of CPS related concept from Lee [2]

Figure 4.9 shows that CPS is made of one or more Cyber Elements and one or more Physical Elements. Each Cyber Element in the CPS could affect the Physical Elements in the system and vice versa. While this is a close representation to the CPS definition by [2], we had to remove the concept of System Elements established in Section 4.5.1. The major weakness of this approach is that the concept of System Element is deeply integrated into the systems engineering methodology, and complex systems are typically developed on a component basis that utilises this concept. Breaking up the System Element concept into strictly Cyber or Physical Elements would make our approach incompatible with the long-established and accepted systems engineering method.

Instead of focusing on the Cyber and Physical aspect of the CPS definition, an alternative would be to focus on the activity of the elements making up the CPS. A self-driving car can be taken as an example for observation. The major features that transform a traditional car into a self-driving car is its ability to detect the environment surrounding the car with sensors, plan for its movement with a computer, and control the car's mechanics to follow the planned movements with actuators. From this simple example, we can abstract the key activities of a CPS to be sensing, actuating, and control.

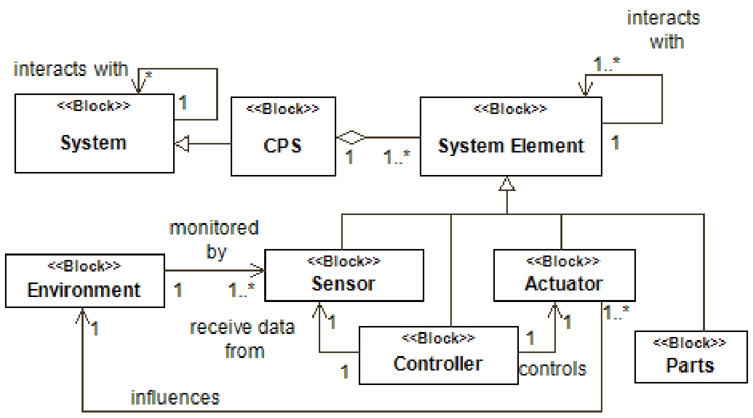


Figure 4.9: CPS related concept for CPS modelling

Figure 4.9 shows the extended concept of System from Section 4.5.1 to include the CPS concept. Some concepts such as the System property and Interface are not shown in order to simplify the diagram for this section. The diagram shows that instead of focusing on the Cyber and Physical aspect, attention is given to the elements that enable the activities making up a CPS.

- 'CPS' - a special type of 'System' that is made up of one or more specific types of 'System Elements', such as 'Sensor', 'Controller', 'Actuator', and 'Parts'.
- 'Sensor' - a special type of 'System Elements' that monitors the Environment by detecting or measuring a physical property.
- 'Actuator' - a special type of 'System Elements' that is responsible for moving and controlling a mechanism or system that influences the physical the Environment.
- 'Controller' - a special type of 'System Elements' that is responsible for the decision-making process of the System. Controller receives data from the Sensor and provides a signal to control the Actuator.
- 'Part' - a special type of 'System Elements' for CPS that can't be categorised as a Sensor, Actuator, or

Controller. Parts are typically a physical entity binding the System Elements together to make up the CPS. For example, the cables making up the transmission lines of a smart grid can be categorised as Parts.

- 'Environment' - represents the state of the physical world surrounding the System. Environment is typically taken in the context of the natural world but is not strictly enforced as CPS could regard other systems as Environment.

Although the concepts of Cyber and Physical are absent from the ontology we proposed, this is not to say those concepts are not relevant for CPS. As we place stronger emphasis on the systems engineering approach, the benefits of following the System Elements approach outweighs the approach shown in Figure 4.8. However, the concept of Cyber and Physical is very important for the co-modelling concepts, which will be covered in Section 4.7.

4.5.3 The Autonomy Concept

Autonomy is closely related to the idea of independence and self-governance. Autonomy by itself can be considered an abstract concept and will only become meaningful when it is put into context. When viewed in the context of CPS, Autonomy refer to the usage of a closed-loop control system to enable self-* capability without the interference of a human operator. Figure 4.10 shows the Ontology for Autonomy related concepts, showing the relationship between 'Autonomy', 'ACPS', and 'Autonomy Requirement'.

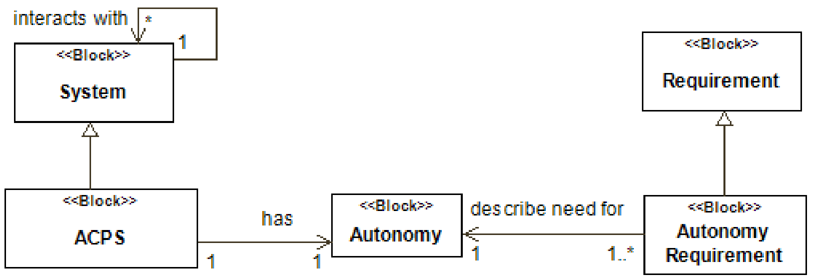


Figure 4.10: Autonomy related concept for CPS modelling

4.5.4 The ACPS Concept

At this point, we are ready to put together all the concepts that we have defined to make up the ontology for ACPS. Figure 4.11 shows the ontology focused on the concept of ACPS and its relationship with the concepts of System, CPS, and Autonomy. It is this version of ontology that will be used to represent the ACPS related concepts and framework in Chapter 5.

ACPS in our architectural abstraction has the same structural make up as CPS. This is why apart from the introduction of ACPS being a special type of CPS, there are no changes to the type of System Elements. The

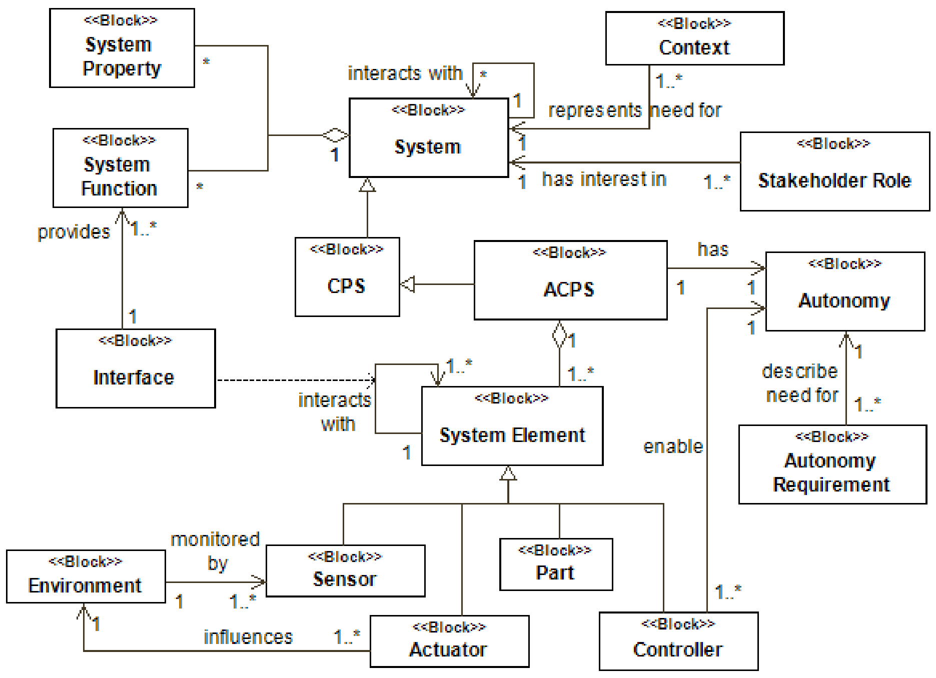


Figure 4.11: ACPS related concept for ACPS modelling

diagram also shows how ACPS is tied to the concept of Autonomy that connects to the Need concept through Autonomy Requirement.

4.5.5 The Interface Concept

Interface is a well studied topic in Systems Engineering. [341] defines Interface as 'The system boundary that is presented by a system for interaction with other systems'. In the context of our ontology, Interface is the means by which System Elements interact with other System Elements in the System. In the case of System-of-Systems, the interacting System Elements could possibly be from other Systems as well.

In our work, we consider an Interface that is operational or service-based in nature. Other forms of Interface such as those based on transfer of data, material, or energy will not be identified explicitly as Interfaces in our ontology. We only consider operational-based Interfaces as we intend to tie the Interface closely with the System Function. Each Interface provides one or more System Functions in the System for interaction. Note that this interface discussion is at the level of one subsystem to another in the system as well as between systems. Interface in systems engineering is also about a system and its environment, which are things outside the system boundary (see Section 2.2.3 for definition). Interfaces between system or subsystem are quite different compared to interfaces between environments as environments are outside the control/management of the system developer.

4.5.6 The System Function Concept

System Function is provided by the Interface to interact with other ‘Systems’ or ‘System Elements’. Each ‘System’ or ‘System Element’ could have an ‘Interface’ and ‘System Function’ that is implemented to the particular ‘Interface’. ‘System Function’ specifies the detailed behaviour of the interaction between ‘System Elements’. Each ‘System Functions’ are owned by one ‘System’. However, one ‘System Function’ could affect the state of multiple ‘System Elements’. This is because a ‘System’ could be composed of multiple ‘System Elements’, and each of the ‘System Elements’ could potentially be considered a ‘System’ in itself, which in turn is composed of more ‘System elements’. Figure 4.12 shows an example of SysML block representing a ‘System Element’ that contains two ‘System Functions’ (i.e., ‘open()’ and ‘close()’) realised as an SysML operation.

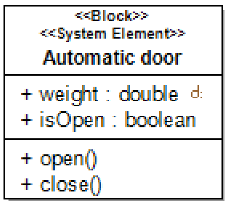


Figure 4.12: Block diagram showing a ‘System Element’ containing ‘System Property’ and ‘System Function’

4.5.7 The System Property Concept

‘System Property’ describes an attribute, quality, or characteristic of a ‘System’ or ‘System Element’. For example, Figure 4.12 shows an automatic door as a ‘System Element’. The door may have several physical attributes that are of interest, such as its weight and whether it is in an open or closed position. Each of these attributes can be represented as ‘System Properties’ as shown on the middle block compartment in Figure 4.12 (i.e., ‘weight’ and ‘isOpen’). A ‘System Property’ typically describes something that is quantifiable in nature (e.g., weight), but it could also be descriptive as well.

4.6 Extended CPS Modelling

The concepts we introduced in Section 4.5 are focused on the autonomy aspect of CPS and do not bring other defining characteristics of CPS into consideration. In this section, we discuss CPS characteristics that are not explicitly captured in our current definition of ACPS related concepts, such as the concept of System-of-Systems, Network, and Human. Unlike the other sections in this chapter, the concepts defined in this section will not be used in our framework for ACPS in Chapter 5. The concepts introduced here serve as an example and suggestion of how our ontology and framework can be extended to accommodate more complex concepts of CPS.

4.6.1 The System-of-Systems Concept

System-of-Systems (SoS) is a well-researched concept in the systems engineering discipline, and the methodology to designing solutions to SoS problems is referred to as Systems-of-Systems Engineering (SoSE). In this section, we provide a suggestion of how our ontology can be extended to accommodate the design of SoS.

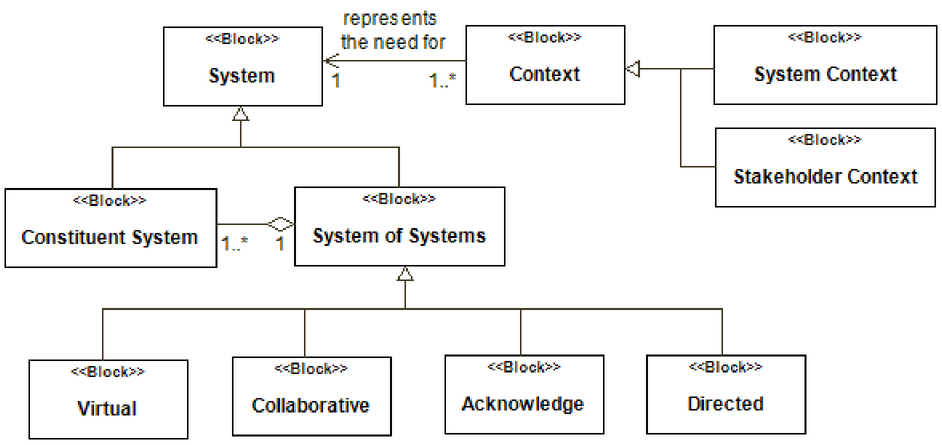


Figure 4.13: Subset of SoS ACRE Ontology focused on System-of-Systems

Figure 4.13 shows the subset of SoS ACRE Ontology that focused on System-of-Systems. The full ontology of SoS ACRE would include the full ontology of ACRE as well, but is omitted for clarity. The concepts in the diagram are defined as follows:

- 'System' - satisfy one or more 'System Contexts'. Where 'System' is a 'System-of-Systems', its element will be made up of one or more 'Constituent Systems'.
- 'Constituent System' - a special type of 'System' that made up a 'System-of-Systems'.
- 'System-of-Systems' - a special type of 'System' that is made up of one or more 'Constituent Systems' that delivers unique functionality not achievable by any single 'Constituent System'. 'System-of-Systems' can be 'Virtual', 'Collaborative', 'Acknowledged', or 'Directed'.
- 'Directed' - a special type of 'System' that has recognised objectives, a designated manager, and resources. Each 'Constituent System' retains its independent operation but not management.
- 'Acknowledged' - a special type of 'System' that has recognised objectives, a designated manager, and resources; however, each 'Constituent System' retains its independent management and operation.
- 'Collaborative' - a special type of 'System' that lacks central management and interacts somewhat voluntarily to fulfil agreed upon central purposes.
- 'Virtual' - a special type of 'System' that lacks a central management authority and a centrally agreed upon purpose for the 'System-of-Systems'.

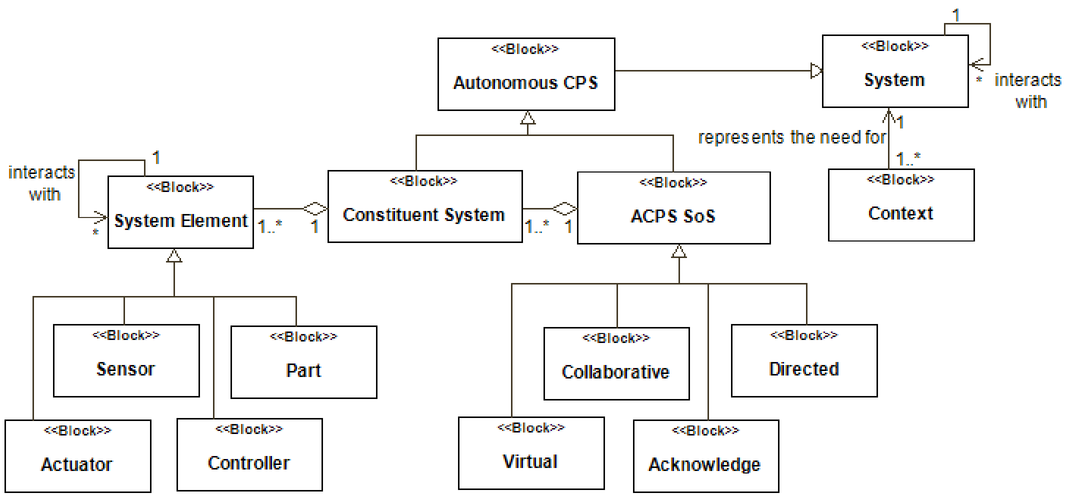


Figure 4.14: ACPS Ontology extended with SoS related concepts from SoS ACRE

Using the SoS concepts defined in SoS ACRE, an example of ACPS Ontology extended to include SoS related concept is captured in Figure 4.14, where:

- 'System' - satisfy one or more 'Contexts'. Unlike ACRE, we do not specify any special type of 'Context' in our ontology.
- 'ACPS SoS - a special type of 'Autonomous CPS' that is made up of one or more 'Constituent Systems'. 'ACPS SoS' refers to autonomous CPS that shows the characteristics of SoS.
- 'Constituent System' - a special type of 'Autonomous CPS' that is made up of a 'ACPS SoS'. Each 'Constituent System' is made up of one or more 'System Elements' that are either a 'Sensor', 'Actuator', 'Part', or 'Controller'.

It is important to stress that the given ontology is only a suggestion and that there are alternatives to integrating SoS ACRE into our ontology. Our example introduces the concept of 'ACPS SoS' as a special type of 'Autonomous CPS', but an alternative would be to have 'SoS' be a special type of 'System' or 'CPS' and draw an association between 'SoS' and 'Autonomous CPS'.

The purpose of this section is to show that while we do not provide further investigation into the concept of SoS in our research, this demonstration shows how our ontology can easily be extended to accommodate other concepts.

4.6.2 The Network Concept

The concept of network in CPS is one that has not been discussed thoroughly in our ontology. For some applications of ACPS such as the self-driving car and Unmanned Aerial Vehicle (UAV), it is possible to design it as a closed system that does not communicate with other systems, which means that the system can be designed without any network components. However, full-fledged applications of CPSs usually involve the deployment of multiple CPSs (e.g., swarm of UAVs) that communicate with each other to achieve an emergent behaviour. CPSs could also be geographically distributed, such as a smart grid CPS where the sensors are distributed across the grid. Network components are required for this to communicate the data from sensors to the control centre [59].

Given that communication is an integral part of CPS [47, 67], there are two reasons why the concept of network is not included in our ontology. Firstly, the need to model communication in CPSs is closely linked to the need to model real-time requirements, which is beyond the scope of our research [342, 343]. Secondly, our ontology still allows network components to be modelled, and the remainder of this section will demonstrate how this can be achieved.

No Network Component. The first approach is to ignore the network aspect at the architectural level and leave it to the implementation. In our ontology, 'System Elements' can interact with each other. For example, the 'Controller' is supposed to send a control signal to the 'Actuator'. The 'Controller' and the 'Actuator' could be connected through a cable that transfers the control signal or over a network. However, it doesn't matter at the architectural level, and we assume that the communication will be resolved at the implementation level.

Network with 'Sensor' and 'Actuator'. As we have the concept of 'Sensor' and 'Actuator' in our ontology, we could utilise them as the network components to enable communications across the 'System Elements'. The most common medium used for system networks is radio waves, and communication is enabled with the use of a transmitter to generate radio waves and receiver to receive the waves at a different location. The radio wave receiver is commonly accepted as a form of sensor, so it makes sense to use the 'Sensor' to represent this. However, the term actuator is traditionally used to refer to the components that move a mechanism and is considered to be different from a transmitter. Although there would be an etymology issue, it is possible to treat the transmitter as an 'Actuator' that generates radio waves in the 'Environment'. These waves would in turn be captured by a 'Sensor' that is observing the 'Environment', thus enabling communication across distributed 'System Elements'.

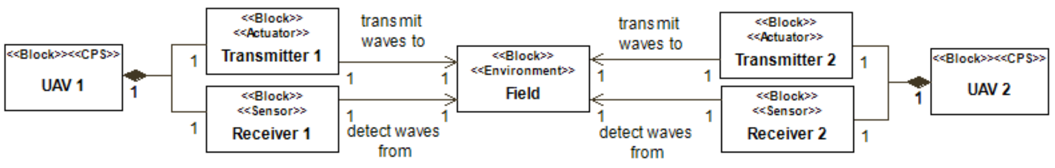


Figure 4.15: Example CPS model using 'Actuator' and 'Sensor' to model network components

An example of this application is shown in Figure 4.15. The example model shows that there are two CPSs in the form of a UAV and each UAV has a transmitter that is an 'Actuator' and a receiver that is a 'Sensor'.

The 'Actuator' and 'Sensor' interact with each other through a 'Field', which is an 'Environment', to enable two-way communication.

'Part' as Network. An alternative to using 'Sensor' and 'Actuator' is to use 'Part'. In our ontology, 'Part' is defined as a special type of 'System Element' that is not a 'Sensor', 'Actuator', or 'Controller'. Although it will predominantly be used to model the physical parts of a system, we can utilise this to model the network component.

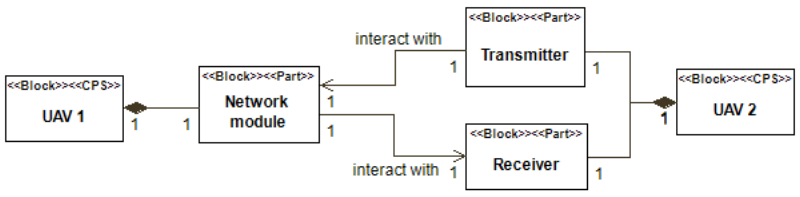


Figure 4.16: Example CPS model using 'Part' to model network components

Figure 4.16 shows an example of a model using 'Part' to describe the network component of the system. The diagram shows that 'UAV 2' is composed of two 'Parts', a 'Transmitter' and 'Receiver' that is used to enable communication. It is also possible to use 'Part' to model a network module that is treated as a black box. This is not an uncommon practice as the design of complex systems tend to integrate fully functioning network modules to serve as their network component.

Which approach should be used out of the three mentioned above would depend on the focus and the importance placed upon the network architecture of the target domain and model. If there is no need to model the network system into the model, then the network component can be left out. In situations where it is desirable to model the specific inner working of the networking nodes (e.g, model and simulate a faulty receiver on one of the network modules), the second approach of using 'Sensor' and 'Actuator' could be a suitable approach. The third approach is best when it is desirable to model the network elements, but the 'Sensor' and 'Actuator' stereotype is not suitable for the component.

Introduce the concept of 'Network'. Figure 4.17 shows an example of how our ontology can be extended to include the concept of 'Network' as a special type of 'System Element'. In this approach, 'Network' is used to represent any element that is used to enable the communication aspect of CPS. This approach might be especially useful if the ontology is needed to support real-time operations that are inherent in networks.

4.6.3 Machine Learning Concept

The concept of machine learning, especially in the context of ACPS, is an area of ongoing research. Although there has been literature on the application of ML in ACPS, creating an ontology to describe the architecture of ML in the context of ACPS will be challenging. Our research activity does not stop at creating an ontology,

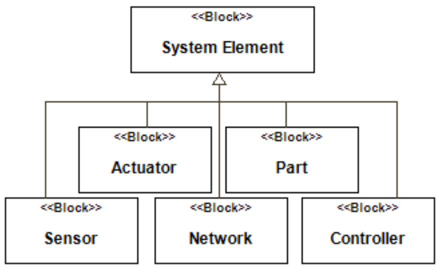


Figure 4.17: Example introduction of ‘Network’ concept into existing ontology

but to define an AF that uses the ontology to aid the MBSE of ACPS. This means that to include ML in the research would include an ontology that describes the general architecture of ML in ACPS, define a framework to support the realisation of such architecture in the ACPS model, and create a case study of ACPS with ML to validate the proposed approach.

Defining MBSE methods to realise such architecture in the model design and create such a model for a case study requires a novel approach that is beyond the scope of this thesis. The ability to support the MBSE of ML in ACPS will be part of planned future work. Here, we propose a simple ML-based ontology as a preliminary plan towards MBSE of ML in ACPS.

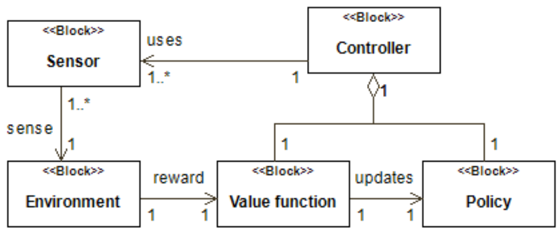


Figure 4.18: Example introduction of ML concept into existing Ontology

Figure 4.18 shows the addition of new ontology concepts in relation to existing ontology concepts. The new concept includes *Value function* and *Policy*. This ontology is based on architecture proposed to support the description of reinforcement learning [344]. Note that as the ontology above is based on reinforcement learning, it may be incomplete (i.e., in relation to addressing other types of ML such as supervised and unsupervised learning) and has not been verified for correctness.

A *value function* can be described as an element that evaluates the reward for an agent to reach a certain state. It is equal to the expected total reward for an agent starting from the state. *Policy* is a set of functions that map a state to an action. The policy is used to dictate the behaviour of the system depending on the state of the system and/or environment.

The ontology presented in Figure 4.18 can be said to be an incomplete one. When trying to describe the purpose of *Value function* and *Policy*, we had to use terms such as ‘state’ that are not represented in the ontology. A good ontology should capture all the terms and concepts that are correlated. As mentioned, the ML ontology here is meant to be a preliminary for future work. It can be used as a reference and starting point for future ontology work, but not for use in AF.

4.6.4 The Human Concept

Although the goal of fully ACPS is to operate without the intervention of a human *operator*, it doesn’t mean that it is meant to operate independently from interaction with a human *user*. For example, a fully autonomous self-driving car does not need the assistance of any human driver to safely reach its destination. However, it will still need to have an interface that allows the human passenger to set the destination that the passenger would like to go to. From this perspective, it is important that the concept of humans is taken into consideration in the CPS design.

The study of humans in CPS has been given a few different terms, such as Human-in-The-Loop CPSs (HiTL-CPS) [345], Cyber Physical Social Systems (CPSS) [346] and Cyber-Physical-Human Systems [347, 348]. At a more superficial level, humans can be treated as a special type of environment. For example, the case study in [349] is concerned with the CPS interacting with humans in the form of crowd-sensing. When humans are treated as a crowd like how self-driving cars would treat humans on the road, it is sufficient for the CPS to detect and treat humans as an environment as there is no real interaction between humans and CPSs.

However, for CPSs that are required to have a more complex interaction with humans (e.g human passenger in self-driving car), it might be necessary to include the concept of humans into the ontology. Nunes et al. [345] explore the concept of humans as sets of sensors, as well as humans as actuators (along with humans as communication and processing nodes). As we have the concept of ‘Sensor’ and ‘Actuator’ in our ontology, it might be possible to adopt some of this approach. An alternative would be to include the concept of ‘Human’ as a special type of ‘System Element’ (see Figure 4.19).

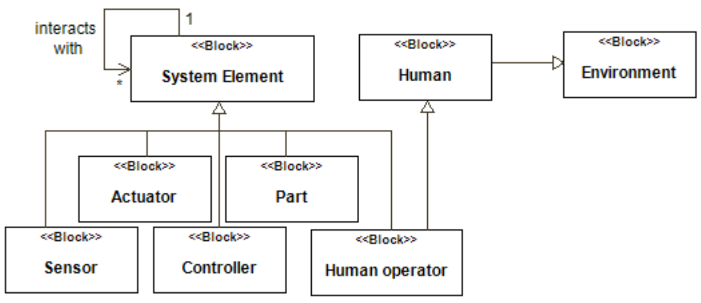


Figure 4.19: Example introduction of ‘Human’ concept into existing ontology

It is easy to assume that 'System Elements' might be exclusive for manufacturing matters, but this is not the case as 'System Elements' can be anything that makes up the 'System'. And if a human is an integral part making up the 'System', it is perfectly justifiable to have 'Human' as a 'System Element'. Figure 4.19 shows that 'Human' could be both a special type of 'Environment' as well as 'System Element'. However, the type of human that is part of the 'System Element' is a special type of 'Human' called the 'Human operator'. In this approach, 'Human operator' is a special type of 'Human' that would have a special type of interactions with other System Elements.

This is an approach that would need to be explored further. If there is a distinction between a human that is considered part of the system and another that is considered to be part of the environment, how does the system make the distinction between the two? One way to achieve this could be some form of identification and authentication technique, but this will depend on the context of ACPS. For our AF, we treat humans exclusively as part of the environment, and just like any type of environment, it can still interact with ACPS through the sensors available in the ACPS.

4.7 Co-modelling and Co-simulation Modelling

The activity of modelling and simulation is an integral part of MBSE approach, so we need to define modelling to understand it. The diagram in Figure 4.1 shows that there is a concept of Model at the centre of the ontology. The concepts in the diagram make up the ontology for the co-modelling approach that we take in our research. The following sections discuss each of the ontology elements in detail.

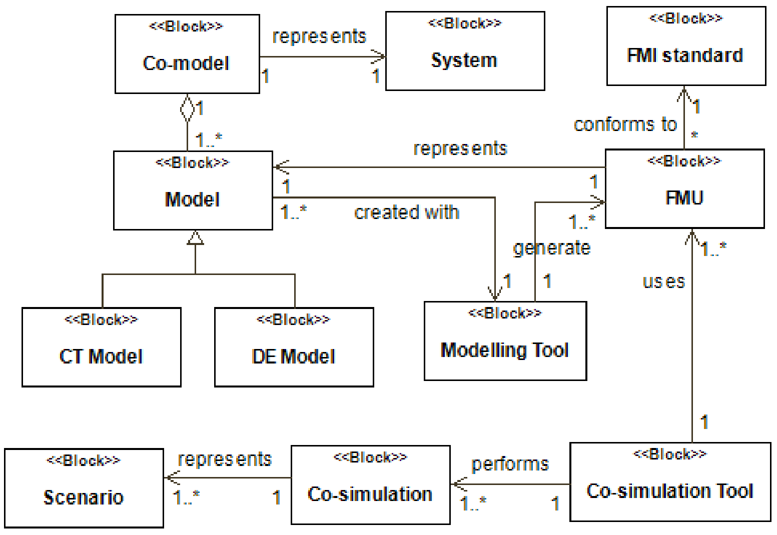


Figure 4.20: Ontology focused on Model related concept

- 'Model' - a 'Model' represents a part of a 'System'. There are two special types of 'Models': 'CT Model' and 'DE Model'.

- 'CT Model' - a type of 'Model' that is modelled with continuous-time formalism, where state of model changes continuously through time.
- 'DE Model' - a type of 'Model' that is modelled with discrete-event that views values of variables as occurring at distinct, separate points in time.
- 'Co-model' - is made of one or more 'Models' and uses the collective 'Model' to represent a 'System' as a whole.
- 'Modelling Tool' - is used to create one or more 'Models'. Each 'Modelling Tool' is typically used to either create a 'CT Model' or 'DE Model', but not both.
- 'FMU' - represents a 'Model' making up the 'Co-model' that represents a 'System'. One or more 'FMUs' are generated by a 'Modelling tool' to be used by the 'Co-simulation Tool'.
- 'FMI standard' - sets the standard and rules of how 'FMU' can be used to interact with one another.
- 'Co-simulation tool' - uses one or more 'FMUs' to perform one or more 'Co-simulations'.
- 'Co-simulation' - is the computer execution of models that represents one or more 'Scenarios'.

The concept of Model has been used across many different disciplines, and different organisations within a discipline could have slightly different definitions about Models as well. A Model can be seen as 'a representation of one or more concepts that may be realised in the physical world' [350]. [351] describe Models as 'an abstraction of a system, aimed at understanding, communicating, explaining, or designing aspects of interest of that system'. And one of the simpler definitions is to treat a Model as a simplification or reality [352].

Although different words were used to define a Model, there is a common understanding that a Model is a representation of 'something', and that 'something' in our case would be a system, or more specifically, an ACPS. There are many things that can be considered a Model, such as mathematical models, physical models, visual models, and text models. The Models that we are interested in are Models that are used in the context of modelling and simulation, in which computers are used to represent physical phenomenons in the form of simulated mathematical equations.

It is important that the Model does not oversimplify the system it is trying to represent. Oversimplification means that the Model loses its connection to the reality of the system, meaning that anything we learn from the Model would be meaningless. In Section 2.5 we cover the literature stating that CPS models should utilise both continuous-time formalism and discrete Models to correctly model CPS. This approach of modelling, where multiple different domain-specific modelling methodologies are used, is called Co-modelling. A Co-model refers to the collection of Models created to represent a system.

In terms of MBSE ontology, a 'Model' is a domain specific representation of a part of a 'System'. One or more 'Models' make up a 'Co-model' that is used to represent a 'System' as a whole. There are two special types of Models that we are interested in. One is 'CT Model', that is modelled with continuous-time formalism, and the other is 'DE Model', for discrete event modelling.

4.8 Architecture Modelling

The concept of an architecture is the foundation of any systems engineering activity. The definition or architecture could differ between different industries or stakeholder groups. As we aim to propose an Architectural Framework as a part of our research, this section will be covering the MBSE Ontology for Architecture related concepts, which includes the Architectural Framework. The definition for architecture related concepts that we use are taken from ISO 42010 as mentioned in Section 2.4.1.

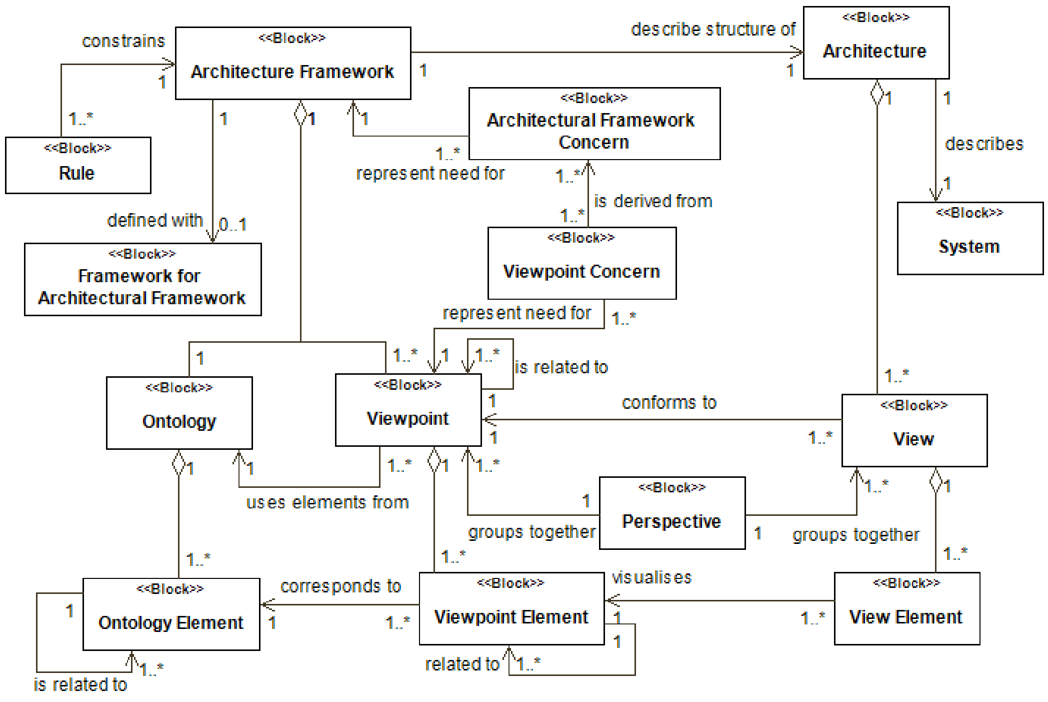


Figure 4.21: Ontology focused on Architecture and Architectural Framework related concept

Figure 4.21 shows the MBSE Ontology for Architecture and Architectural Framework related concepts that we will use throughout our research. We do not intend to create a new set of Ontology for Architecture related concepts as this concept has been well established in the Systems Engineering domain. The ontology we use is taken from [240] with a minor modification, namely the addition of the 'Framework for Architectural Framework' block. Although the ontology proposed in [240] is aimed for systems in general, we believe that the architecture concept can still be effectively applied for ACPS.

- 'Architectural Framework' - describes the structure of an 'Architecture' from the points of specific stakeholders or industries. It is made of an 'Ontology' and one or more 'Viewpoint(s)'. An 'Architectural

Framework' can be defined with or without 'Framework for Architectural Framework'.

- 'Framework for Architectural Framework (FAF)' - is a framework that guides the creation of 'Architectural Framework'. This framework can be used to ensure that the defined 'Architectural Framework' follows a standard or good practice of systems engineering. This ontology is not part of [240], but is introduced here as the FAF will be used as the basis of our architecture framework creation.
- 'Architecture Framework Concern' - represents the Need that the 'Architectural Framework' aims to realise.
- 'Ontology' - defines one or more 'Ontology Element(s)', the concepts and terminologies relating to the 'Architecture' that the 'Architectural Framework' aims to describe.
- 'Ontology Element' - is the concepts and terms making up an 'Ontology'. 'Ontology Elements' are needed to define each 'Viewpoint' through 'Viewpoint Elements'. Each 'Ontology Element' is related to one or more 'Ontology Elements'.
- 'Viewpoint' - defines the structure and content of one or more 'View(s)'. 'Viewpoint' derives its structure and content using one or more 'Ontology Elements' that make up the 'Ontology'. 'Viewpoint' is required to fulfil the Needs defined by one or more 'Viewpoint Concerns'. One or more 'Viewpoints' can also be grouped together into a 'Perspective'. As 'Viewpoint' defines the template for 'View', each 'View' must conform to its 'Viewpoint'.
- 'Viewpoint Concern' - defines the Need that a 'Viewpoint' has to address. One or more 'Viewpoint Concern(s)' are derived from one or more 'Architectural Framework Concern(s)'.
- 'Viewpoint Element' - is a set of elements that corresponds to 'Ontology Element' from 'Ontology' that is part of the 'Architecture Framework'. One or more 'Viewpoint Elements' make up a 'Viewpoint', and each 'Viewpoint Element' can be related to each other.
- 'Architecture' - is a description of a 'System'. 'Architecture' is made of one or more 'View(s)'.
- 'View' - represents an artefact that is produced as part of a project. In terms of ontology, 'View' visualises parts of the 'System' in terms of its 'Architecture' that conforms to its associated 'Viewpoint'. Each 'View' is made up of one or more 'View Element(s)'
- 'View Element' - visualises a 'Viewpoint Element' that corresponds to 'Ontology element' making up a 'Viewpoint'. One or more 'View Elements' make up a 'View'.
- 'Perspective' - groups together one or more 'Viewpoints' in relation to their intended purpose. This grouping means that 'Perspective' also groups together one or more 'Views', as 'Views' have to conform to a 'Viewpoint'.
- 'Rule' - constrains how the 'Architectural Framework' is used to describe an 'Architecture'. For example, there could be a rule stating which of the 'Viewpoints' are compulsory or optional when using the 'Architectural Framework'.

4.9 Evaluation

This chapter defines ACPS Ontology to be used with an AF to support the MBSE of ACPSs. The ontology describes all the concepts related to the MBSE of ACPSs and maps a relationship between each concept to ensure the definitions are consistent. Here, we evaluate the ontology in terms of its ability to answer the competency questions set as the requirement for the ontology defined in Section 4.3.

- What are the elements making up an AF?
  An AF is made of an Ontology and one or more Viewpoints.
- What is the relationship between AF and Viewpoint?
  One or more Viewpoints are required in an AF.
- What is the relationship between AF and View?
  AF describes the structure of an Architecture, and an Architecture is made of one or more Views.
- What is the relationship between AF and Ontology?
  An Ontology is needed to build an AF.
- What is the relationship between AF and System?
  AF describes the structure of an Architecture, and an Architecture describes a System.
- What is the relationship between Requirement and Need?
  Requirement is a type of Need.
- What are the elements needed to describe a Need?
  Need is described with a Need Description, and Need Descriptions are elicited from a Source Element.
- What is the relationship between Need and System?
  The need of a System is represented by its Context. Context is applied to the Need to derive Use Cases.
- What are the elements making up a System?
  A System is made of one or more System Elements.
- What are the elements making up a CPS?
  A CPS is made of one or more System Elements. These elements can be categorised into Sensor, Actuator, Part, and Controller.
- What is the relationship between CPS and Environment?
  CPS uses Sensor to monitor the Environment and Actuator to influence the Environment.
- What are the elements making up an ACPS?
  ACPS is a special type of CPS that inherits the structure of CPS. ACPS is made of one or more System Elements, and these elements can be categorised into Sensor, Actuator, Part, and Controller.
- What is the relationship between ACPS and Autonomy?
  ACPS is capable of Autonomy.
- What is the relationship between Autonomy and Autonomy Requirement?
  Autonomy Requirement describes the need for Autonomy.

- What are the roles of Controller in ACPS?
  Controller receives data from Sensor and controls the Actuator. A Controller is required to enable Autonomy.
- What are the elements making up a Co-model?
  A Co-model is made of one or more Models. There are two types of Models, the CT Model and DE Model.
- What are the elements making up a Co-simulation?
  Co-simulation is not made of any elements. However, it is provided by Co-simulation Tools.
- What is the relationship between Co-model and Co-simulation?
  Co-model and Co-simulation are related through FMU.
- What is the the relationship between FMU and Co-model?
  Co-model is used by the Modelling Tool to generate FMUs.
- What is the the relationship between FMU and Co-simulation?
  FMUs are used by the Co-simulation Tool to perform Co-simulation.
- What is the relationship between Co-model and System?
  Co-model represents a System.

The competency questions are designed to cover concepts from AF, MBSE, CPS, Autonomy, CPS, Requirements, Co-modelling, and Co-simulation. This ontology would allow the AF described in the next chapter to use these concepts as well. Next, we evaluate the ontology based on the Intended Impact Model described in Chapter 3.

Actual Impact Model

The Actual Impact Model will be presented as a list, where each element in the list is taken from the Intended Impact Model shown in Table 3.1. For each of the elements in the Impact Model, we will evaluate whether the element has Need realised in the ACPS Ontology. Note that the evaluation does not indicate whether the framework has achieved what it is intended to do (this can only be achieved with a validation method). Instead, it evaluates whether the design process of ACPS Ontology has been carried out as planned.

- The Support should be described in the MBSE modelling language - [SATISFIED]
  The ontology is defined with SysML, which is a modelling language commonly used for MBSE.
- The Support should provide additional description in formal ontology language - [SATISFIED]
  The SysML ontology is translated into a formal ontology defined in OWL. Details of this ontology are covered in Appendix A. However, ontology verification (e.g., evaluating competency questions as SPARQL queries against the ontology) is not provided and is instead included as further work.
- The Support should cover key concepts in the MBSE life-cycle, from requirements to V&V - [SATISFIED]
  The ontology required to capture concerns in the MBSE life cycle are concepts relating to requirements (Section 4.4, system architecture (Section 4.5), and co-simulation for verification and validation (Section

4.7)

- The Support should include explicit characterisation of autonomy - [SATISFIED]

  Autonomy is described explicitly in Section 4.5.3 and is an element in the ontology.
- The Support should not be restricted to a limited types of autonomy - [N/A]

  It is difficult to prove that our ontology does not restrict any type of autonomy. One way to demonstrate this is by using the ontology to model several ACPSs with different types of autonomy. This means that the evaluation can only be done after the case studies in Chapter 6. However, Section 4.6.3 does show that the ontology might need to be specialised to support autonomy features such as ML.
- The Support should include concepts of CPSs - [SATISFIED]

  Section 4.5 includes the concepts of CPSs and combines it with Autonomy concepts to derive concepts for ACPSs.
- The Support should not be restricted to certain CPS domains - [SATISFIED]

  We ensure that the ACPS ontology is not domain specific by avoiding the use of domain specific concepts. This ensures that the ACPS Ontology is applicable across domains such as transportation, manufacturing, and agriculture.
- The Support should be transferable to the MBSE framework - [SATISFIED]

  The ACPS Ontology is defined in SysML, which means that the ontology can be directly used as part of the AF for the MBSE of ACPS.
- The Support should include concepts relating to Co-modelling and Co-simulation - [SATISFIED]

  Section 4.7 provides the ontology for Co-modelling and Co-simulation related concepts.
- The Support should be usable by experienced systems engineers - [SATISFIED]

  Here we assume that the experienced systems engineers have knowledge of SysML. As the ontology is designed in SysML, it should be usable by systems engineers.
- The Support should be defined with a framework - [SATISFIED]

  The method proposed by Hildebrandt et al. [327] is used for the construction of this ontology. We were not able to fulfil all the recommended methods stated in [327]. This will be evaluated in detail in Chapter 7.
- The Support should base its ontology on an existing standard when possible - [SATISFIED]

  Each concept proposed in the ontology is based on an existing standard or published work. The literature that the concepts are based on is mentioned in each section of the ontology.

5 Architectural Framework for ACPS

This chapter presents the Architectural Framework for ACPS (ACPSAF) as part of the Support required in the Prescriptive Study stage of the research following the DRM. Section 5.1 introduces the scope of ACPSAF, describing what and how the AF should be realised. The ACPSAF itself is made up of Perspectives covered in Sections 5.2 to 5.10. Section 5.2 presents the AF Perspective, which contains self-explanatory Viewpoints. It describes the purpose of ACPSAF, the ontology used, rules to use ACPSAF, and a summary of the framework content.

Section 5.7 is the Need Perspective, which is concerned with the requirements engineering activity in the MBSE process. Section 5.8 is focused on the CPS Perspective, concerned with capturing the structure and non-autonomous behaviour of ACPS. Section 5.9 is Autonomy Perspective, which is concerned with ensuring that autonomy requirements in ACPS can be satisfied and highlighting the autonomy behaviour to achieve this. Section 5.10 is the Co-modelling Perspective, which is concerned with using the Co-modelling and Co-simulation approach as a method to support V&V of a system model. The chapter is then closed with an evaluation in Section 5.11.

5.1 Scope

This section specifies all the framework, methods, and tools used to create the ACPSAF. It also revisits the Intended Impact Model the ACPSAF should fulfil, which was defined earlier in Chapter 3 and provides an overview to the structure of ACPSAF.

5.1.1 Framework and methods

Architecture Framework Framework (FAF)

A meta-AF would be used to define the ACPSAF. Utilising a framework to build our AF would ensure that the AF is holistic and aligns with good practice in the MBSE domain. For that purpose, the Architecture Framework Framework (FAF) [240] was chosen to aid the ACPSAF development as the FAF specifies a set of Viewpoints that should be followed when defining an AF. As a result, the overall structure and functionality of ACPSAF was greatly influenced by FAF. Further description on how FAF was used to develop ACPSAF is covered in Section 5.1.4.

Approach to Context-based Requirements Engineering (ACRE)

ACRE is a framework-based, MBSE approach to requirements engineering [241]. ACRE defines seven Viewpoints to elicit and analyse requirements, identify its context, define acceptance criteria, and establish traceability. ACRE is designed to work for any system in general, so there are no issues with engineering ACRE for requirements engineering of ACPS. However, one Viewpoint from ACRE has been updated such that autonomy requirements can be identified in ACPS.

MBSEAF, DODAF, and MODAF

ACPSAF contains a perspective known as the CPS Perspective, which is related to capturing the structure and non-autonomous behaviour of ACPSs. CPS Perspective is analogous to a system viewpoint that can be found in other AFs. For MODAF, it is similar to the Systems Viewpoint that describes the physical implementation of operational and services of defence planning. Meanwhile, it is also similar to the Systems View that captures systems and services supporting the DoD functions in DODAF. In MBSEAF, it is similar to the System Perspective that captures the structure and behaviour of generic systems. The CPS Perspective developed in ACPSAF takes inspiration from these three AFs, with MBSEAF being the most influential on how the Viewpoints in the CPS Perspective are defined. Nevertheless, as CPS Perspective is aimed for ACPS and uses the ACPS Ontology, it differs greatly from the aforementioned AFs in terms of its domain target.

INTO-SysML

The Co-modelling Perspective is concerned with capturing the Co-modelling and Co-simulation approach to support V&V of a system model. INTO-SysML [353] is an UML based profile that customises SysML for architectural modelling of FMI co-simulation. It specialises in SysML blocks to represent different types of components in a CPS and presents two diagrams to capture the architecture and connections of the CPS co-model for co-simulation purposes. Our work in Co-modelling Perspective is influenced by the work done in INTO-SysML. However, our approach is to generalise the concept and principles taken in INTO-SysML as ACPSAF is a framework.

5.1.2 Tools

SysML 1.2

When it comes to SysML usage in ACPSAF, there are two aspects we should pay attention to. Firstly, ACPSAF is described in both natural language and SysML 1.2. Future iterations of SysML might change the semantics of the language, but it should not be affected by minor updates since ACPS is described using very basic building blocks. For example, the latest version of SysML at the time of writing is 1.5, and all changes have not affected any meanings from the diagrams.

The second aspect is the fact that ACPSAF expects users to use ACPSAF with SysML to model ACPS. Although users can use other modelling languages with ACPSAF to model ACPS, ACPSAF is designed with SysML 1.2 as its modelling language of choice. When using another language with ACPSAF, users have to find the equivalent expression between SysML 1.2 and their modelling language of choice. Needless to say, ACPSAF is most effective when used with SysML 1.2. One important SysML update to keep in mind is the changes to Ports since SysML 1.3. To be specific, Flow Ports and Standard Ports are deprecated in favour of Proxy Ports and Full Ports. This means that for each Viewpoints in ACPSAF that uses Ports, users will have to find the equivalent representation in their version of choice.

Modelio

The only requirement for the SysML modelling tool is that it must support all 9 SysML 1.x diagrams. The modelling tool we used to create the SysML model and diagrams is an open source version of Modelio 3.7.01 which supports the SysML diagrams requirement. Modelio needs an additional module known as the SysML Architect module to enable modelling in SysML. The SysML Architect module used is 3.0.02, which allows users to model in SysML 1.2. This is also the main reason why the SysML version chosen is 1.2.

5.1.3 Requirement and Contribution

The requirement acquisition process for ACPSAF has been carried out in Chapter 3 when carrying out the preliminary stage of the Prescriptive Study. This resulted in a document known as the Documentation of Intended Support covered in Section 3.4. To serve as a reminder, we present some content from the Documentation of Intended Support again in this section. In particular, the Intended Impact Model provides a clear list of requirements that ACPSAF should fulfil.

RDM also states that the core contribution should be identified before the implementation of the Support itself. This is to ensure that most of the resources, such as time and attention, are allocated to the critical area. For ACPSAF, the core contribution would be the Autonomy Perspective. As covered in Section 5.1.1, the other Perspectives in ACPSAF are built on top of existing work that has been re-engineered. However, the Autonomy Perspective is an original contribution where autonomy characterisation is used in the context of

Problem statement:	
Develop MBSE framework that supports the architecting of ACPSs.	
Performance:	
D	The Support should use the MBSE ontology for ACPS defined in Chapter 4.
D	The Support should be readily deployed with MBSE modelling language.
D	The Support should be focused on system architecture (as opposed to low-level implementation).
D	The Support should be an Architectural Framework.
W	The Support should provide framework description in MBSE modelling language.
D	The Support should facilitate MBSE life-cycle from requirements to V&V.
D	The Support should be able to identify autonomy characteristics in ACPS.
D	The Support should incorporate separation of concerns between autonomy and non-autonomy architecture.
D	The Support should be able to identify and capture autonomy requirements in ACPS.
D	The Support should be able to express the architecture of ACPS.
D	The Support should provide a method to support the V&V of ACPS.
D	The Support should not be restricted to limited types of autonomy.
D	The Support should not be restricted to certain CPS domains.
Ergonomics:	
D	The Support should be usable by individual experienced systems engineers.
Rigour:	
W	The Support should be defined with a framework.
D	The Support should reuse components from existing frameworks where applicable.

Table 5.1: Intended Impact Model for ACPSAF

MBSE. Apart from Autonomy Perspective, other core contributions would be the Requirement Description Viewpoint that has been adapted to identify autonomy requirements, and the System Traceability Viewpoint designed to establish traceability between SysML model to FMU co-models.

5.1.4 ACPSAF Structure

ACPSAF is developed following the method stated in FAF, introduced in Chapter 2.4.1. Concepts that are important to FAF, as well as the relationship between them, are described in an ontology and used to describe the FAF views. The concepts and relationships included in the FAF ontology relate to the architectures and AFs, and are based on concepts defined in ISO/IEC/IEEE 42010 [35].

When creating an AF, the use of FAF is intended to ensure that:

- The requirements of the AF are properly understood.

- Terminology and concepts that can be modelled using the AF are defined
- Necessary views of a system are identified
- The requirements of each view are understood in relation to the overall requirements of the AF
- Each view is defined based on terminology and concepts identified as key to the AF
- Any rules constraining the AF are identified

In order to satisfy these requirements for the description of an AF, the FAF states that an AF shall be composed of six Viewpoints. Figure 5.1 shows the example of the Viewpoint Relationship View diagram showing all six Viewpoints as described by FAF and the relationship between each Viewpoint.

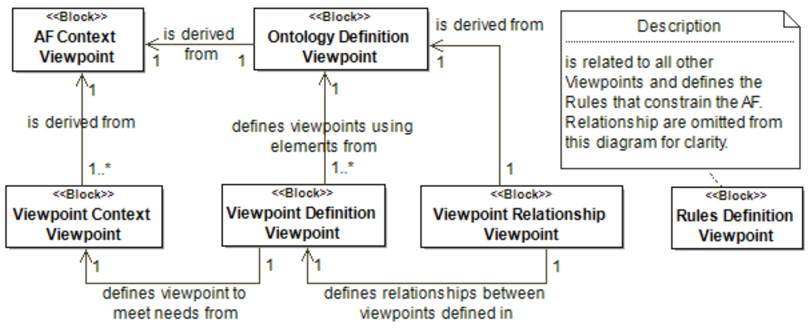

<<Block>>
AF Context Viewpoint
is derived from
1
1
<<Block>>
Ontology Definition Viewpoint
is derived from
1
Description
is related to all other Viewpoints and defines the Rules that constrain the AF. Relationship are omitted from this diagram for clarity.
1
is derived from
1..*
defines viewpoints using elements from
1
1..*
1
<<Block>>
Viewpoint Context Viewpoint
<<Block>>
Viewpoint Definition Viewpoint
<<Block>>
Viewpoint Relationship Viewpoint
<<Block>>
Rules Definition Viewpoint
1
1
1
1
defines viewpoint to meet needs from
defines relationships between viewpoints defined in

Figure 5.1: Example Viewpoint Relationship View for FAF

AF Context Viewpoint (AFCV) Defines the Context for the Architectural Framework. Allowing Needs the AF addresses to be captured, establishing why the AF is needed. The AFCV for our AF will be covered in Section 5.2.

Ontology Definition Viewpoint (ODV) Defines the ontology for the AF. Defining the concept, terms, and the relationship between them for the domain where the AF is to be used. The ODV for our AF will be covered in Section 5.3.

Viewpoint Relationships Viewpoint (VRV) Shows the relationship between Viewpoints that make up an AF and group them into Perspectives. The ODV for our AF will be covered in Section 5.4.

Rules Definition Viewpoint (RDV) Defines a number of rules constraining the AF. The ODV for our AF will be covered in Section 5.5.

Viewpoint Definition Viewpoint (VDV) Defines a particular Viewpoint, showing the ontology that appears on the Viewpoint. This is to ensure each Viewpoint can be defined consistently, ensuring its conformance to the ontology. The VDV for our AF will be covered from Sections 5.7 to 5.10.

Viewpoint Context Viewpoint (VCV) Defines the Context for a particular Viewpoint. To ensure that each Viewpoint is fit for its purpose and meets the overall Needs for the AF. It is important that the Needs for each Viewpoint are clearly understood. The VDV for our AF will be covered together with the VDV from Section 5.7 to 5.10.

Note that the first four Viewpoints (AFCV, VRV, ODV and RDV) are meta-viewpoints that are used to describe ACPSAF itself. For most of these Viewpoints, only one diagram is needed to capture the View (except VRV). However, VDV and VCV are different as it states that two that two types of Views must be created for each Viewpoint that we create for ACPSAF. That means that if ACPSAF defines ten Viewpoints to support the architecting of ACPS, at least twenty diagrams are needed to capture the DVD and VCV of each Viewpoint. As a result, each Viewpoint in the ACPSAF will be structured as such:

1. **Title and introduction.** Each Viewpoint starts as a subsection, followed by a brief introduction to the Viewpoint.
2. **VCV.** Following the FAF, the context of the Viewpoint is introduced, and the VCV diagram is presented.
3. **VDV.** Following the FAF, the definition of the Viewpoint (in relation to ACPS Ontology) is introduced, and the VDV diagram is presented.
4. **View Discussion.** Additional information to accompany the Viewpoint.

5.2 AF Context View

According to FAF, an AF should start by presenting meta-information about itself. The first View is the AF Context View (AFCV) which defines the Context for the AF. This means that it should represent and put the Concerns of the AF into context - why the AF is needed. The AFCV exists solely to define and illustrate the Needs (i.e., AF Concern(s)) of the AF. Therefore, the AFCV should be able to demonstrate:

- ‘Identify AF needs’ - Identify the Needs that the AF must address.
- ‘Understand the relationship between Needs’ - Understand the relationship between the Needs that the AF intends to address.
- ‘Identify AF Stakeholder Roles’ - Identify the Stakeholder Roles involved in the AF that are concerned or impacted by the AF Needs.

The AF Context View for ACPSAF is shown in Figure 5.2 through a Use Case diagram. The diagram shows the concerns of the AF as Use Cases and its relevant Stakeholders. It shows that the main Architectural Framework Concern that the ACPSAF must address is to ‘provide architectural framework for modelling autonomous cyber-physical systems’. The types of modelling that the Architectural Framework must support are indicated by the four Use Cases that constrain the main Concern:

- ‘Must support requirement, design, verification, and validation’ - Architectural Framework should be suitable for the mentioned MBSE life cycle.
- ‘Comply with best practice’ - systems engineering places a strong emphasis on compliance to best practice to ensure the quality of work undertaken. This applies to the ACPSAF that we define as well; it must comply with best practice in the form of Architectural Framework Standards.

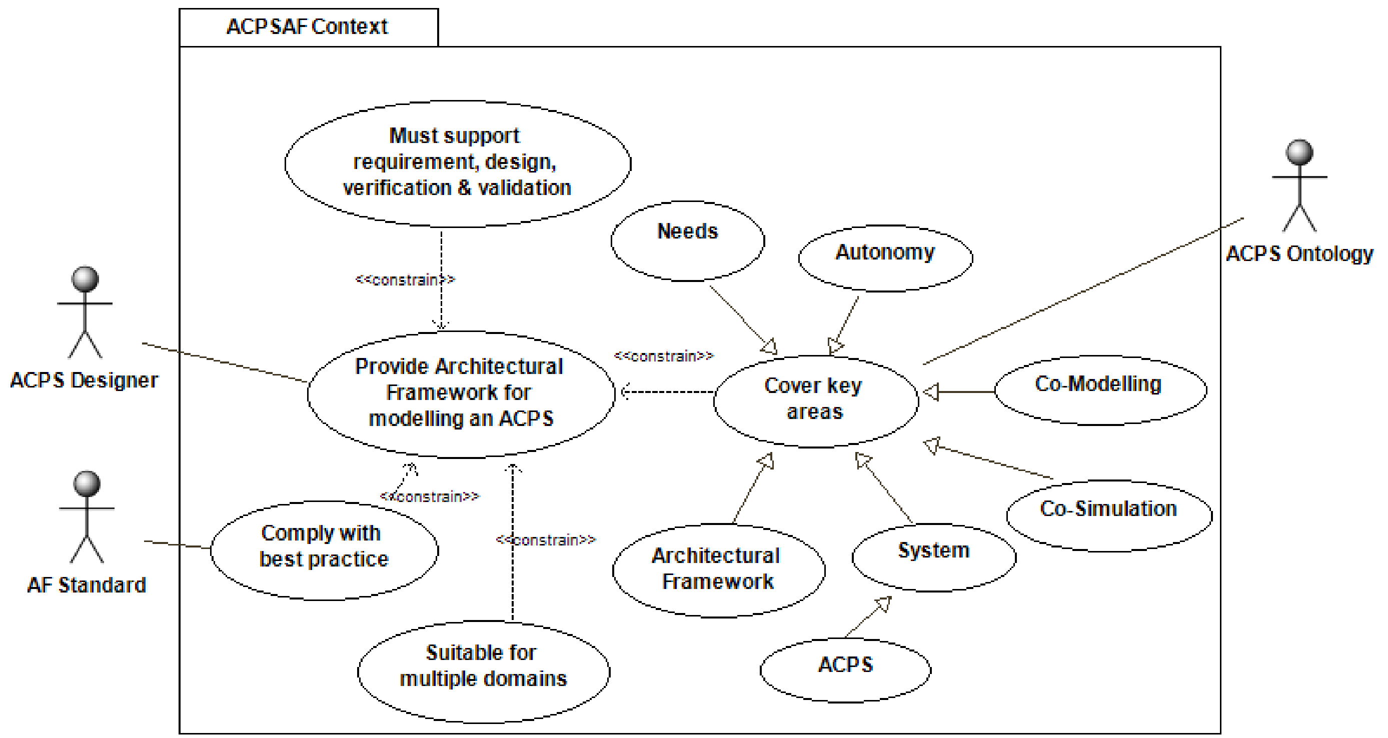


Figure 5.2: AF Context View for the ACPSAF

- ‘Suitable for multiple CPS domains’ - an Architectural Framework that can be used in multiple domains means the time-consuming activity of defining a new framework for every new CPS domain can be avoided. In general, it is better to create AF that is applicable across multiple domains whenever possible. The exception would be when a more domain-specific AF is needed to address design challenges that could not be captured otherwise.
- ‘Cover key areas’ - the key areas that our research aims to address has been covered in our research objectives and ACPS Ontology. Our AF shall use the Ontology defined and cover those key areas.

It should be noted that there are other benefits to the Architectural Framework that are not covered in the AF context view, such as benefits that are common to AFs in general (e.g., increase interoperability and development speed, increase product acceptance/approval, and reduced risk). Figure 5.2 shows that the Use Cases are derived from only three different stakeholders. In practice, having more Use Cases derived from more varied stakeholders may improve the framework in some ways. For example, regulators of the ACPS domains may impose certain constraints or requirements to suit their needs or standard. Having those Use Cases influencing the design of ACPSAF could significantly improve the acceptance rate of framework adoption by the involved stakeholders. As part of future work to increase the Technology Readiness Level of ACPSAFs, the ACPSAF Context may be modified to include more relevant stakeholders, which includes an update on the framework to satisfy those needs.

5.3 Ontology Definition View

The Ontology Definition View (ODV) defines the Ontology for our AF. FAF states that ODV should demonstrate the following key points:

- ‘Identify ontology elements’ - The ODV should identify the Ontology Elements that will be used by the AF.
- ‘Identify ontology relationship’ - There could be a relationship between each Ontology Elements, and the ODV should capture these relationships.
- ‘Identify ontology areas’ - Related Ontology Elements should be grouped together as this will help the grouping process for Perspective, Viewpoints, and View of the AF.

The Ontology defined in the ODV forms the basis for the AF as the concepts used in the AF have to be captured by an Ontology Element. Figure 5.3 shows all the Ontology Elements for ACPSAF and the relationship between each Ontology Element. The diagram shows all of the Ontology Elements for ACPSAF. However, some relationships between the Ontology Elements were omitted for clarity. Multiple Ontology Definition Views can be created for ODV as it can be difficult to arrange a large number of Ontology Elements into a single diagram. However, we will not define an additional Ontology Definition View diagram to capture the full relationship as this has been done in Chapter 4.

While some of the ontology used here is heavily influenced by our work (e.g., Autonomy Requirement), most of the ontology is reused from certain standards. For example, FAF (which is used to create ACPSAF) dictates that an AF must have an AF perspective (a meta-Perspective to describe itself, how the framework is used, etc). One of the most commonly referenced standards for this is ISO-42010, from where the terms 'view', 'viewpoint' and 'architecture' are derived. Unless absolutely necessary (i.e., to capture the correct context), we believe that reusing commonly known and accepted ontologies is beneficial to the framework as the purpose of an ontology is reusability for shared knowledge.

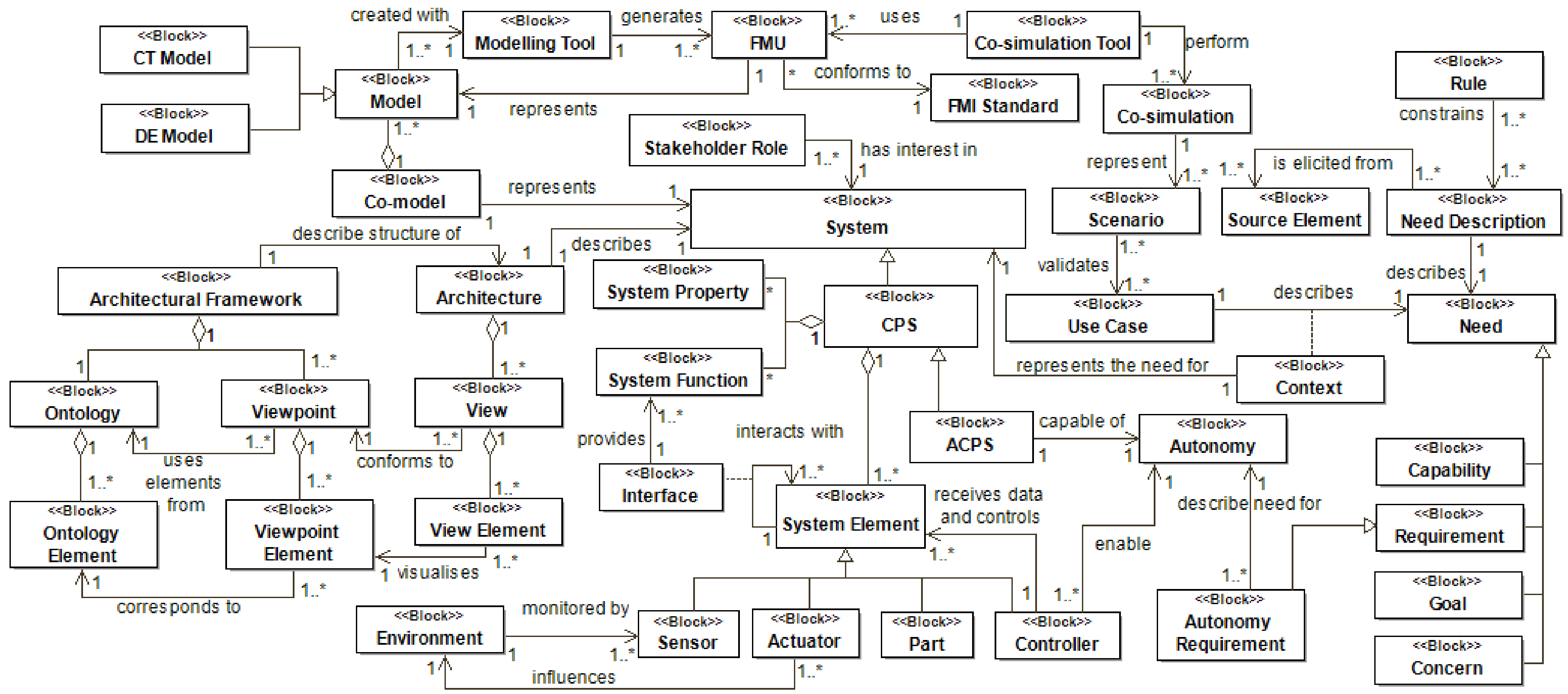


Figure 5.3: Ontology Definition View for the ACPSAF

5.4 Viewpoint Relationships View

The Viewpoint Relationship View (VRV) shows the relationship between the Viewpoints making up the ACPSAF, and the Viewpoints identified here must meet the Need for ACPSAF as stated in the AFCV. A collection of one or more Viewpoints would be called a Perspective, and VRV should show the grouping of Viewpoints into Perspectives as well. The VRV for ACPSAF is shown in Figure 5.4.

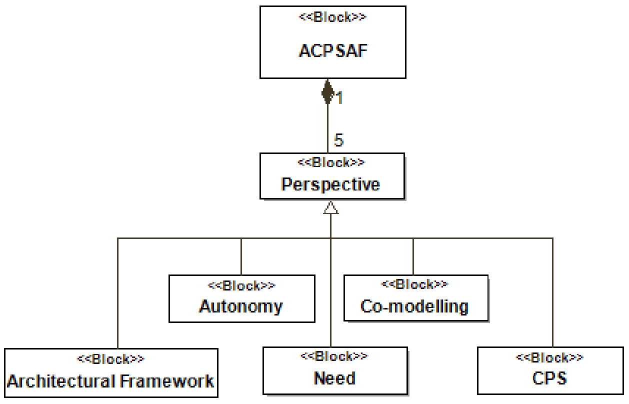


Figure 5.4: Viewpoint Relationship View for the ACPSAF showing the Perspectives

The Viewpoints making up each of the Perspectives is shown in Figure 5.5. It shows four of the five Perspectives, excluding the Architectural Framework Perspective. The Architectural Framework Perspective is shown earlier in this chapter as Figure 5.1. This Perspective is excluded here as it is a meta-Perspective that is used to define ACPSAF itself. Therefore, it does not have a direct relationship with other Viewpoints.

- 'Need Perspective' - defines Viewpoints that allow aspects of System Needs to be captured. The Need refers to an abstract concept that describes a Requirement, Capability, Goal, or Concern of the System. The Need Perspective allows System Needs to be structured in the form of Need Descriptions together with its source information and related Rules. The Need Perspective is concerned with the requirements engineering process and is based on ACRE.
- 'CPS Perspective' - defines Viewpoints that allow aspects of System Needs to be captured in the context of CPS architecture. CPS Perspective is an extension of Need Perspective; it looks at Requirements in the context of System Elements. The CPS Perspective is concerned with capturing the structure and non-autonomous behaviour of ACPS.
- 'Autonomy Perspective' - defines Viewpoints that allow aspects of the Autonomy in ACPS to be captured. It ensures the autonomy requirements can be satisfied and demonstrates their behaviour.
- 'Co-Modelling Perspective' - defines Viewpoints that allow the Co-modelling and Co-simulation approach to be used as a way to support verification and validation of ACPS models. Although the Perspective is called the Co-modelling Perspective, it is intended to cover both Co-modelling and Co-simulation. This is because our work uses co-modelling explicitly in the context of co-simulation. Therefore, they are both

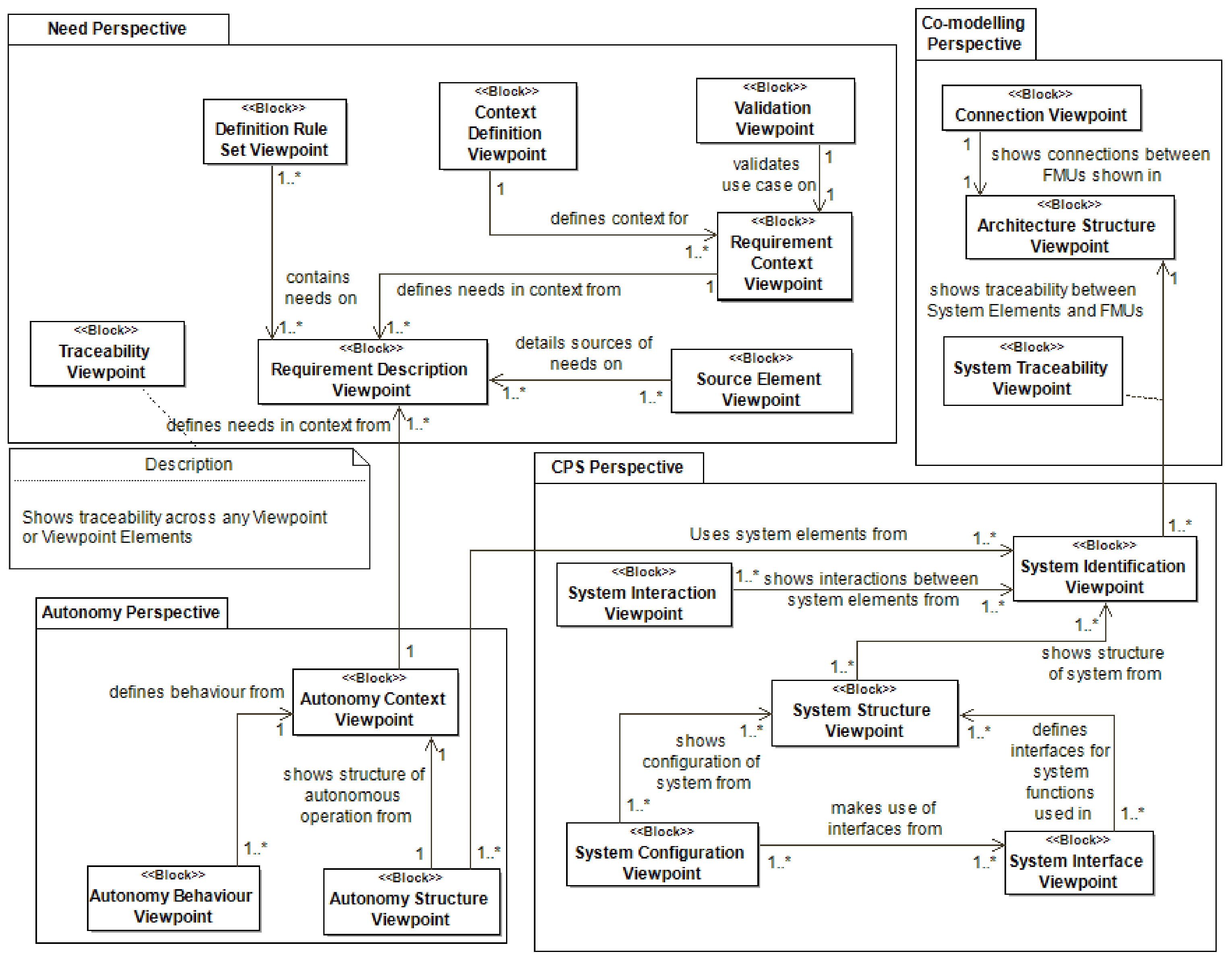
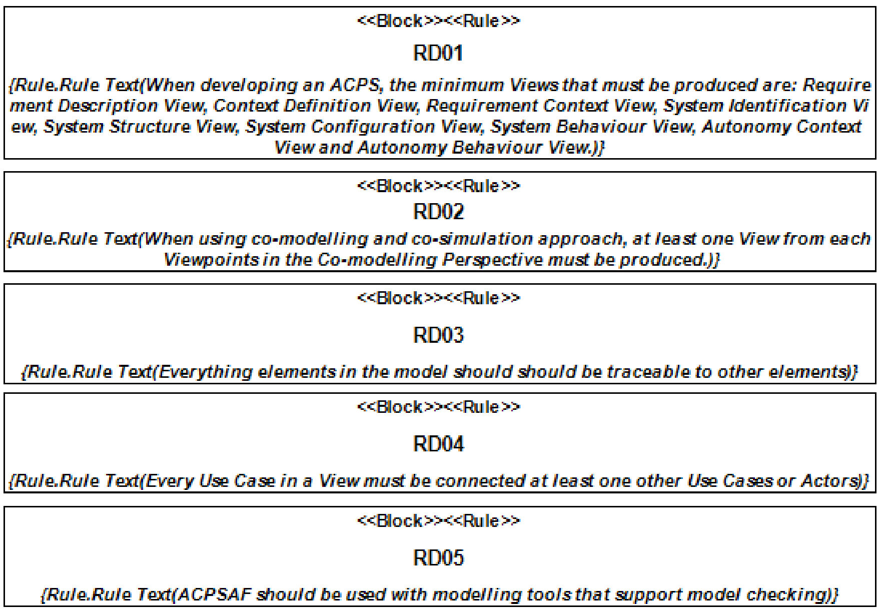

Need Perspective
<<Block>> Definition Rule Set Viewpoint
<<Block>> Context Definition Viewpoint
<<Block>> Validation Viewpoint
validates use case on
defines context for
<<Block>> Requirement Context Viewpoint
contains needs on
defines needs in context from
<<Block>> Traceability Viewpoint
<<Block>> Requirement Description Viewpoint
details sources of needs on
<<Block>> Source Element Viewpoint
defines needs in context from
Description
Shows traceability across any Viewpoint or Viewpoint Elements
Co-modelling Perspective
<<Block>> Connection Viewpoint
shows connections between FMUs shown in
<<Block>> Architecture Structure Viewpoint
shows traceability between System Elements and FMUs
<<Block>> System Traceability Viewpoint
CPS Perspective
Uses system elements from
<<Block>> System Interaction Viewpoint
shows interactions between system elements from
<<Block>> System Identification Viewpoint
shows structure of system from
<<Block>> System Structure Viewpoint
shows configuration of system from
defines interfaces for system functions used in
makes use of interfaces from
<<Block>> System Configuration Viewpoint
<<Block>> System Interface Viewpoint
Autonomy Perspective
defines behaviour from
<<Block>> Autonomy Context Viewpoint
shows structure of autonomous operation from
<<Block>> Autonomy Behaviour Viewpoint
<<Block>> Autonomy Structure Viewpoint
1
1..*

Figure 5.5: Viewpoint Relationship View for the ACPSAF showing Viewpoints in each Perspective

contained in one Perspective and named this way for simplicity.

5.5 Rules Definition View

The Rules Definition View (RDV) defines the Rules that constrain ACPSAF. When using ACPSAF, users should follow these rules as they are intended to help users define a complete, consistent, and traceable ACPS. The RDV can be visualised with a block definition diagram, showing the rules and relationship between rules. Each Rule must have an ID and a description. Figure 5.6 shows the Rules for ACPSAF represented as blocks, identified with the stereotype «Rule». The name of the block represents the ID of the Rule, and the Rule Text contains the description of the Rule.



<<Block>><<Rule>>
RD01
{Rule.Rule Text(When developing an ACPS, the minimum Views that must be produced are: Requirement Description View, Context Definition View, Requirement Context View, System Identification View, System Structure View, System Configuration View, System Behaviour View, Autonomy Context View and Autonomy Behaviour View.)}

<<Block>><<Rule>>
RD02
{Rule.Rule Text(When using co-modelling and co-simulation approach, at least one View from each Viewpoints in the Co-modelling Perspective must be produced.)}

<<Block>><<Rule>>
RD03
{Rule.Rule Text(Everything elements in the model should should be traceable to other elements)}

<<Block>><<Rule>>
RD04
{Rule.Rule Text(Every Use Case in a View must be connected at least one other Use Cases or Actors)}

<<Block>><<Rule>>
RD05
{Rule.Rule Text(ACPSAF should be used with modelling tools that support model checking)}

Figure 5.6: Rules Definition View for the ACPSAF showing the Perspectives

RD01 and *RD02* are intended for model completeness as they specify the Views that AF users have to produce when using ACPSAF. This is to ensure that the minimum required elements to describe an ACPS architecture is included in the model. *RD03* and *RD04* are intended for model traceability. MBSE puts strong emphasis on traceability, where every element in the model should be traceable to all other elements in the model. If traceability cannot be established to an element, that means the element does not contribute to anything in the system and should be removed instead. *RD03* is intended for consistency, as model checking helps users to automatically check the model against SysML syntax.

5.6 Viewpoint Context View and Viewpoint Definition View

Unlike the previous AF Viewpoints, Viewpoint Context View (VCV) and Viewpoint Definition View (VDV) must be defined for each Viewpoint that makes up the ACPSAF. VCV defines the Context for a particular Viewpoint and is usually visualised through a Use Case diagram. Viewpoint Context View is concerned with capturing the Viewpoint Concerns that a Viewpoint is created to address (i.e., the purpose of the Viewpoint). FAF states that VCV shall be used to:

1. 'Identify Viewpoint Needs' - Identify the Needs that the Viewpoints are created to address. This is the purpose of the Viewpoint.
2. 'Understand the relationship between Needs'
3. 'Identify the Stakeholder Roles of the Viewpoint' - Identify the Stakeholder Roles involved in the definition of the Viewpoint. The Stakeholder Role could be anyone that is affected or has an interest in the identified Needs.

The Viewpoint Definition View provides the definition of a Viewpoint and shows the Viewpoints Elements that form the particular Viewpoint. VDV can be visualised with a block definition diagram, the blocks being used to show the ontology involved in the Viewpoint and associations to draw the relationship between them. FAF states that the VDV shall be used to:

1. 'Identify Viewpoint Elements' - Identify the Viewpoint Elements that will be used by the Viewpoint.
2. 'Identify Viewpoint relationships' - Identify the relationship between the Viewpoint Elements in the Viewpoint.

The following sections will cover the Viewpoints making up ACPSAF that support the design process of ACPS. Viewpoints are grouped into Perspectives, with VCV and VDV presented for each Viewpoint.

5.7 Viewpoint Definitions - Need Perspective

This section presents the Viewpoints for the Need Perspective for ACPSAF. The Need Perspective will cover all Need-related concepts as described in the Ontology. As mentioned in Chapter 4, we used the ACRE Framework as the basis for our Need-related Ontology. This means that the Viewpoints presented in this Perspective are also based on the ACRE Framework. Nevertheless, there will be slight modifications in terms of the View produced as the Need-related Ontology for ACPS has been modified from the original ACRE Framework to suit the design of the ACPS.

This section and the rest of the Perspective will be presented in the same fashion. For each of the Viewpoints in the Perspective, we will present the Viewpoint Context View and Viewpoint Definition View with descriptions specifically for that Viewpoint.

5.7.1 Source Element Viewpoint

The Source Element View is used to show all the information needed to capture the Needs of the project. This View is used to ensure that the origin of these Needs are understood. It also serves as a method of establishing traceability, which illustrates the connection between Needs and other aspects of the System.

There are no restrictions on what can be considered as Source Elements. A Source Element can be as abstract as an idea, or as formal as a complete system specification. Common examples of Source Elements include requirement documents, conversations, emails, system specifications, and standards. However, this list is not exhaustive as almost anything could be regarded as a Source Element as long as it is related to the conception of the Needs.

Do note that while this View may appear simple, the process to properly capture the content of the View is difficult and highly risky. Since this is the starting point of the project and the source to guide the elicitation of needs, incomplete or incorrect Source Elements will easily propagate and expand throughout the process of designing the system. Therefore, gathering correct and complete Source Elements requires relevant stakeholders to be identified while balancing the concerns of different stakeholders with different needs and priorities in the deployment of the system.

Viewpoint Context View

The Viewpoint Context View for the Source Element Viewpoint is shown in Figure 5.7. The Viewpoint Concerns that the Source Element Viewpoint is intended to address are shown as Use Cases on the VCV. The main Use Case is to 'Show source information relating to system needs'. This is the primary aim of the Viewpoint - to allow the origin of each Need to be defined. This Viewpoint is used as a primary way of establishing traceability and providing links between the Needs and any other aspects of the System.

There are two inclusions that allow extra information to be captured. The first is 'Relationship between each Source Element'. The second is 'Properties of each Source Element' with three specialised Use Cases, 'Date', 'Type', and 'Status'.

- 'Type' can be used to describe the basic type of source reference, such as a book, project document, system specification, and standard.
- 'Date' in our context would be the date the referenced source was published.
- 'Status' is used to provide control to the configuration of the Source Elements. For example, 'status' could take on values such as 'Approved', 'Draft', and 'Archived'.

Viewpoint Definition View

The Viewpoint Definition View for the Source Element Viewpoint is shown in Figure 5.8. The diagram shows that the 'Source Element View' contains one or more 'Source Elements' and that 'Need Description'

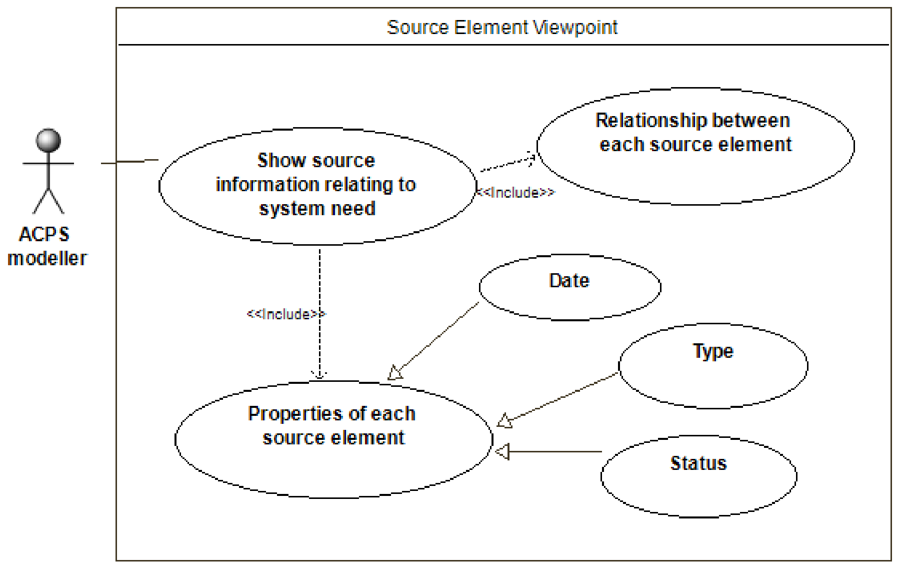

Source Element Viewpoint
ACPS modeller
Show source information relating to system need
Relationship between each source element
<<include>>
<<include>>
Date
Type
Properties of each source element
Status

Figure 5.7: Viewpoint Context View for the Source Element Viewpoint

is elicited from one or more ‘Source Elements’. It also shows that the Source Element View has a relationship with another View in the Framework. This relationship is first shown by the VRV in Figure 5.5, but the diagram here shows further details on this relationship, indicating how the relationship is established through traceability between the ‘Need Description’ and ‘Source Element’.

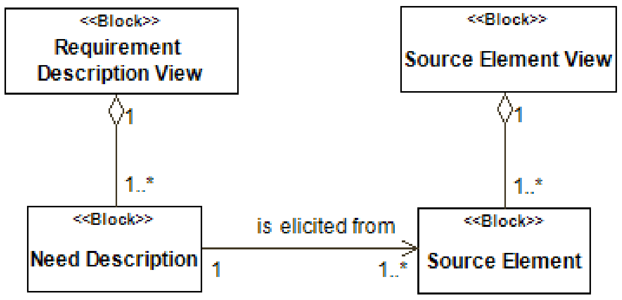


Figure 5.8: Viewpoint Definition View for the Source Element Viewpoint

View discussion

The Source Element View is quite straightforward in terms of its structure and allows a high degree of freedom in terms of what can be included as a ‘Source Element’. The Source Element View can be produced in SysML using a block definition diagram where each Source Element is presented as a block and given the stereotype «Source Element». Due to its simplicity, it is possible to create the Source Element View without using a SysML diagram, such as with a simple table. However, one main benefit of the model-based approach is that it inherently establishes traceability to the model itself. In this case, there would be explicit traceability between the external source and the system model.

In the Viewpoint Context View, we provided three types of properties to the Source Element. These properties are intended as a guide, and framework users are encouraged to modify the properties as necessary for the purpose of the project. For example, users can add new properties such as 'version' and 'location' to introduce version control and a file management system.

5.7.2 Requirement Description Viewpoint

The Requirement Description View is used to describe Needs using Need Description. As mentioned in the Ontology, Need is an abstract concept and Need Description is used to help understand the Need. The Need Description would typically have a set of attributes to help describe the Need, and multiple Need Descriptions would be defined in the Requirement Description View. This View is also used as the first step in identifying which of the defined Need Descriptions relates to the Autonomy Requirement.

Viewpoint Context View

The Viewpoint Context View for the Requirement Description Viewpoint is shown in Figure 5.9. The main Use Case is to 'Describe each need in a structured manner' that includes 'Identify Autonomy Requirements' and 'attributes or features' to capture a detailed description. There are two types of 'attributes or features' introduced in the View, and each Need Description must have all of these attributes defined.

- 'ID' is the unique identifier for Need Description. ID allows a Need Description to be managed by tools and provides a basis for traceability in the system.
- 'Description' is a text that describes the Need Description. The description should be as simple and concise as possible.

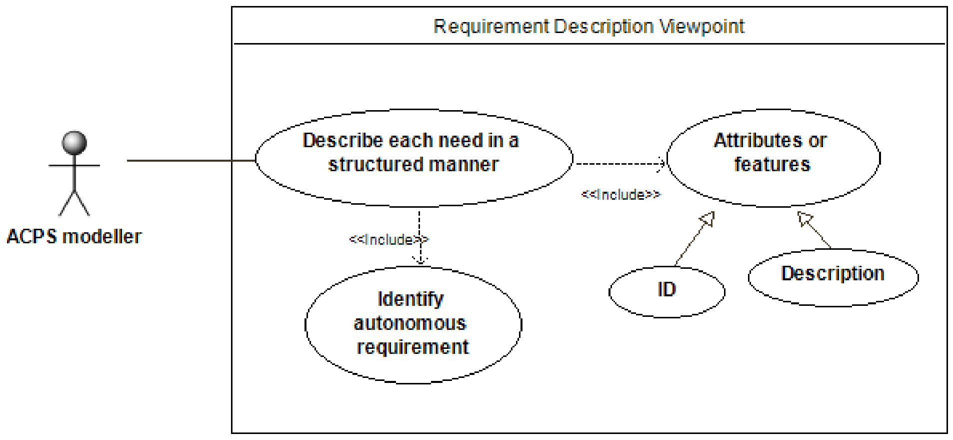
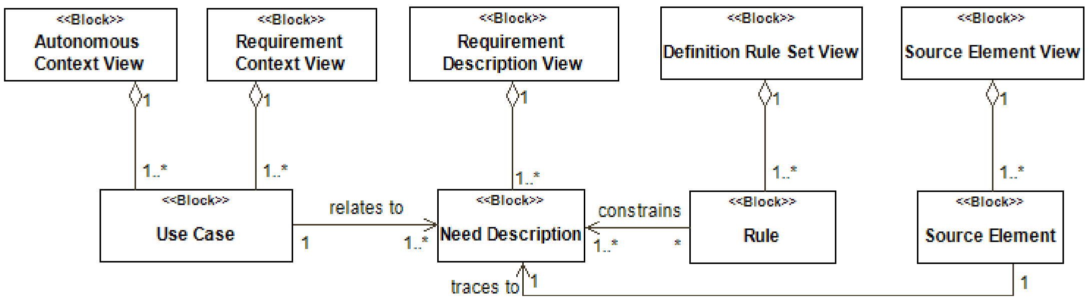

Requirement Description Viewpoint
ACPS modeller
Describe each need in a structured manner
Attributes or features
<<Include>>
<<Include>>
Identify autonomous requirement
ID
Description

Figure 5.9: Viewpoint Context View for the Requirement Description Viewpoint

Viewpoint Definition View

The Viewpoint Context View for the Source Element Viewpoint is shown in Figure 5.10. The diagram shows that the 'Requirement Description View' contains one or more 'Need Descriptions' and that it has a relationship with multiple Views in the Framework. The relationship between this View and the 'Source Element View' has been explained in the previous section. However, there are three more Views. This View is related to the 'Definition Rule Set View' as there might be a 'Rule' that constrains one or more 'Need Descriptions'. This means that if a 'Rule' exists, it must apply to a Need Description. The 'Requirement Description View' also has a relationship with the 'Autonomy Context View' and 'Requirements Context View' through 'Use Case'. Each Need Description must be related to or involved in at least one Use Case from either Views.



<<Block>> Autonomous Context View
<<Block>> Requirement Context View
<<Block>> Requirement Description View
<<Block>> Definition Rule Set View
<<Block>> Source Element View
<<Block>> Use Case
<<Block>> Need Description
<<Block>> Rule
<<Block>> Source Element
relates to
constrains
traces to
1
1..*
*

Figure 5.10: Viewpoint Definition View for the Requirement Description Viewpoint

View discussion

Need Descriptions should be considered individually and its description should be non-contextual. This is because a Need that is considered within a Context is called a Use Case and will be captured by the Requirement Context Viewpoint instead. Similar to the Source Element View, the suitable attributes for Need Descriptions are expected to vary across organisations, domains, and projects.

We suggested 'ID' and 'Text' as the default attributes, but users are expected to add further attributes to fit the Needs of the system. For example, 'Origin' can be added to draw traceability from the Need Description to Source Elements. We did not include 'Origin' here because Section 5.5 states that it is not compulsory to define the Source Element View when using ACPSAF. This means that the definition of Source Element is optional, and the 'Origin' attribute would only be applicable when there are Source Elements.

The Requirement Description View is best realised in SysML using a Requirements Diagram. Each Need Description is represented using a requirement block and its attributes modelled as block properties. The Requirement Description View should be quite simplistic, and the only relationship that will be used in the View would be the containment relationship that shows hierarchy in which one requirement can be decomposed into further requirements.

A block definition diagram can also be used to model the Requirement Description View, using block and block properties to represent Need Description and its attributes. When using a BDD as an alternative to the Requirements Diagram, a custom stereotype would be needed in place of the Requirement stereotype (e.g., refine, satisfy) that comes with a Requirements Diagram. The stereotype «requirement» would be given to identify the block as a requirement, and association to represent the containment relationship.

On top of the generic types of requirements that are stereotyped with «requirement», any Need Description that relates to the system's ability to operate autonomously shall be stereotyped with «autonomous». The «autonomous» stereotype would indicate that the block is an Autonomy Requirement.

As described in our Ontology, we are concerned with the autonomy that enables self-* capability through utilisation of the closed-loop control. To help identify which of the requirements is an Autonomy Requirement, users should go through each requirement and check it against the following characteristics:

- Autonomy Requirement requires control over sensors and actuators to achieve its goals, where
- Actuators are used to influence physical properties in the Environment or the System itself in relation to the Environment, and
- Sensors are used to measure the changes in the physical properties in the Environment, and
- The goal needs to be achieved without the intervention of human operators, relying on the ACPS to make decisions and control the System Elements accordingly.

It is important that Need Descriptions are captured and decomposed into the right level of abstraction as they will affect the requirements engineering process and help identify the Autonomy Requirements. If users are unsure whether a requirement is an Autonomy Requirement, the recommended practice is to treat it as one. This is because the Autonomy Requirements will be refined further in the Autonomy Perspective. It will then be easy to identify if a requirement is not an Autonomy Requirement.

5.7.3 Definition Rule Set Viewpoint

The Definition Rule Set View is used to capture Rules that will be used to constrain the Need Description. The constraint could be text-based rules or in the form of mathematical equations. These constraints are enforced by applying the rules to the attribute values of the Need Description. This means that the Rules defined have to refer to the attributes of a Need Description (e.g., 'txt').

Viewpoint Context View

The Viewpoint Context View for the Definition Rule Set Viewpoint is shown in Figure 5.11. The main Use Case is to 'Define rules applied to each Need Description' that includes 'Rule name or identifier' and 'Rule description', where 'Rule description' could be a 'mathematical equation' or 'list of texts'.

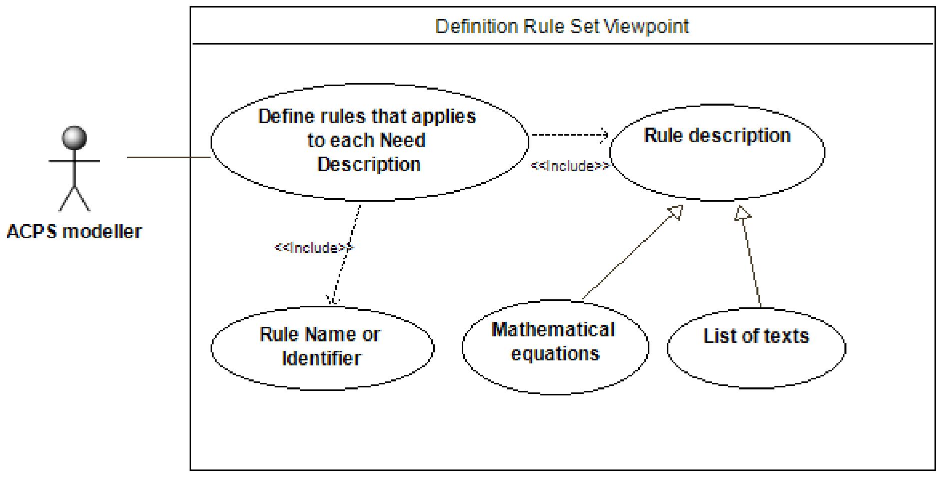


Figure 5.11: Viewpoint Context View for the Definition Rule Viewpoint

Viewpoint Definition View

The Viewpoint Definition View for the Definition Rule Set Viewpoint is shown in Figure 5.12. The diagram shows that 'Definition Rule Set View' contains one or more 'Rules' and has a relationship with the 'Requirement Description View' as one or more 'Rules' constrain one or more 'Need Descriptions' that are contained within the 'Requirement Description View'.

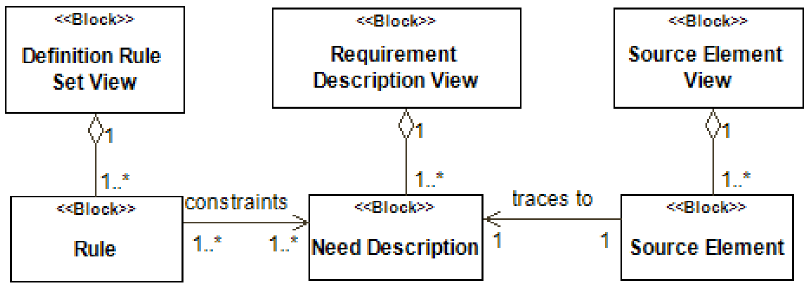


Figure 5.12: Viewpoint Definition View for the Definition Rule Viewpoint

View discussion

The Definition Rule Set View can be realised in SysML using a block definition diagram. Each Rule can be modelled using a block for text-based rules and a constraint block for mathematical-based equations. For equation-based rules, each rule represented by a constraint block would have the attribute 'constraint(s)' that defines the mathematical equations and parameters attributes that represent the parameters used in the equation. For text-based rules, each rule can be represented using a block with the stereotype «Rule» that has an attribute to contain the text description of the rules.

Although it is possible to define specific Rules that target a few Need Descriptions, typically all the Rules defined in the Definition Rule Set View apply to all the Need Descriptions. Rules are typically used to target the 'ID' and 'txt' of Need Descriptions as the two mentioned attributes are typically always used in every project. Examples of text-based rules include 'each requirement must have a unique identifier' and 'the word APPROXIMATELY must not be used in the Requirement Description.'.

It is best to keep each rule as simple and concise as possible, so it is better to split a complex rule into multiple rules when applicable. Each Rule would be identified individually, and there is no need to model the relationship between Rules even when some Rules are related to each other. This means that the Definition Rule Set View would typically be a block definition diagram with a collection of blocks or constraint blocks without any relationship association between diagram elements.

5.7.4 Context Definition Viewpoint

The Context Definition View is used to capture the Contexts or point of views that will be considered in the project. These Contexts can come in a range of types, but we will be concentrating on Contexts focused on Stakeholder Roles and System Elements.

Viewpoint Context View

The Viewpoint Context View for the Context Definition Viewpoint is shown in Figure 5.11. The main Use Case is to 'Define Contexts for the project' that includes 'Identify the types of Context' and 'Define Elements for each type of Context'.

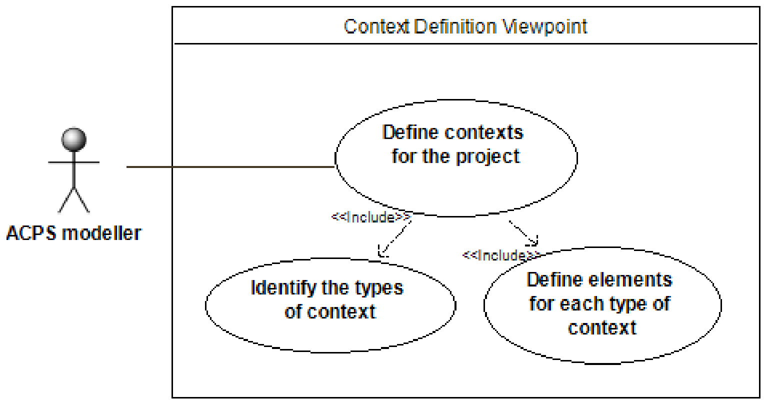
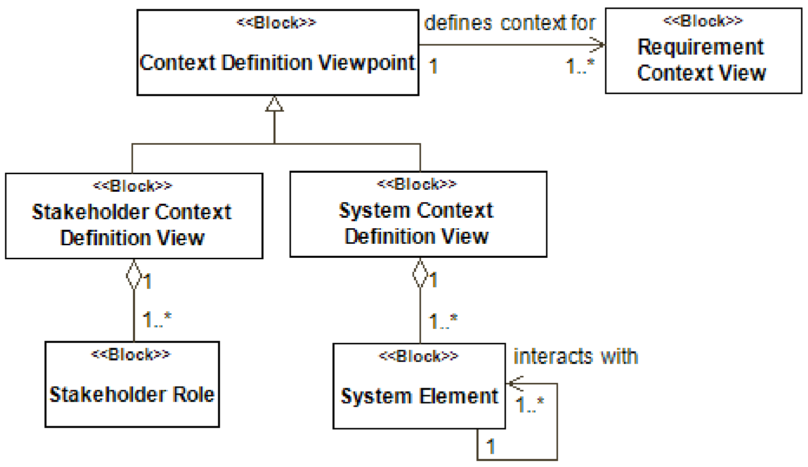

Context Definition Viewpoint
ACPS modeller
Define contexts for the project
<<include>>
<<include>>
Identify the types of context
Define elements for each type of context

Figure 5.13: Viewpoint Context View for the Context Definition Viewpoint

Viewpoint Definition View

The Viewpoint Definition View for the Context Definition Viewpoint is shown in Figure 5.14. The diagram shows that there are two types of 'Context Definition View(s)': the 'Stakeholder Context Definition View' that contains one or more 'Stakeholder Roles', and the 'System Context Definition View' that contains one or more 'System Elements' that interact with each other. It also shows that both types of 'Context Definition View(s)' are incomplete. Therefore, users are expected to add additional types as appropriate for their project.

The diagram also shows that the 'Context Definition View' has a relationship with another View as it defines the context for one or more 'Requirements Context View(s)'.



<<Block>> Context Definition Viewpoint
defines context for
1
1..*
<<Block>> Requirement Context View
<<Block>> Stakeholder Context Definition View
<<Block>> System Context Definition View
1
1..*
<<Block>> Stakeholder Role
1
1..*
<<Block>> System Element
interacts with
1..*
1

Figure 5.14: Viewpoint Definition View for the Context Definition Viewpoint

View discussion

The Viewpoint Context View can be realised in SysML with a block definition diagram where each Context is typically modelled as a block. For the Stakeholder Context Definition View, the Stakeholder Roles can be shown as a taxonomy with a generalisation relationship to define the different types of Stakeholder Roles. For the System Context Definition View, the System Elements are modelled as blocks and defined in the form of a structural hierarchy using a composition relationship.

This View can be difficult to get right for two reasons. Firstly, the Viewpoint Context Viewpoint could have multiple Views, needing the framework user to determine which Views are required for the project. Secondly, identifying and classifying the elements of the Context at the right level of abstraction is not a trivial task. It is possible that further Views are created before a system designer analyses their model and finds out the Context defined needs to be further refined.

5.7.5 Requirement Context Viewpoint

The Requirement Context View is used to capture Use Cases and its relationship to other Use Cases and Contexts in the project. As mentioned in the Ontology, a Use Case refers to a Need that has been put into Context. This View is the first step to establishing relationships and traceability between requirements and related contexts (i.e., stakeholders, systems).

Viewpoint Context View

The Viewpoint Context View for the Requirement Context Viewpoint is shown in Figure 5.15. The main Use Case is to 'Describe Need from a specific Context' that includes 'Show the Use Cases involved in each Context', 'Show relationship between Use Cases', and 'Show relationship between Use Cases in one Context to other Contexts'.

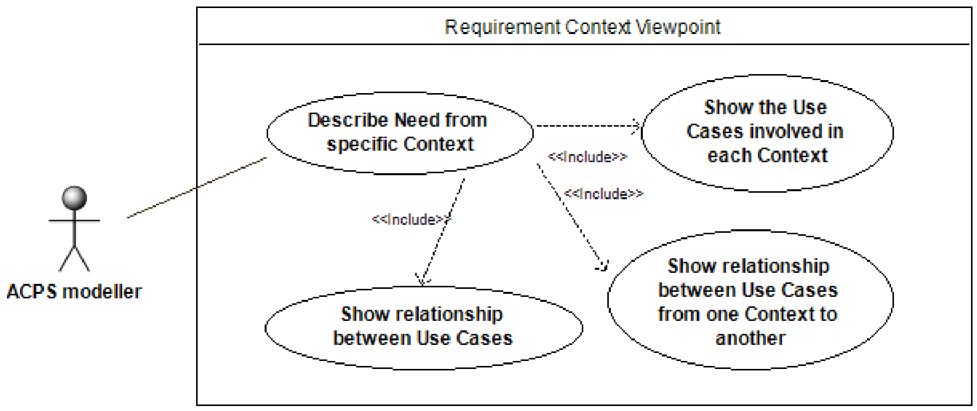

Requirement Context Viewpoint
Describe Need from specific Context
Show the Use Cases involved in each Context
<<Include>>
<<Include>>
<<Include>>
Show relationship between Use Cases
Show relationship between Use Cases from one Context to another
ACPS modeller

Figure 5.15: Viewpoint Context View for the Requirement Context Viewpoint

Viewpoint Definition View

The Viewpoint Definition View for the Requirement Context Viewpoint is shown in Figure 5.16. The diagram shows that there are two types of 'Context Definition View(s)': the 'Stakeholder Context Definition View' that contains one or more 'Stakeholder Roles', and the 'System Context Definition View' that contains one or more 'System Elements' that interact with each other. It also shows that both 'Context Definition View(s)' are incomplete. This means that users are expected to add additional types as appropriate for their project. The diagram also shows that 'Context Definition View' has a relationship with another View as it defines the context for one or more 'Requirements Context View(s)'.

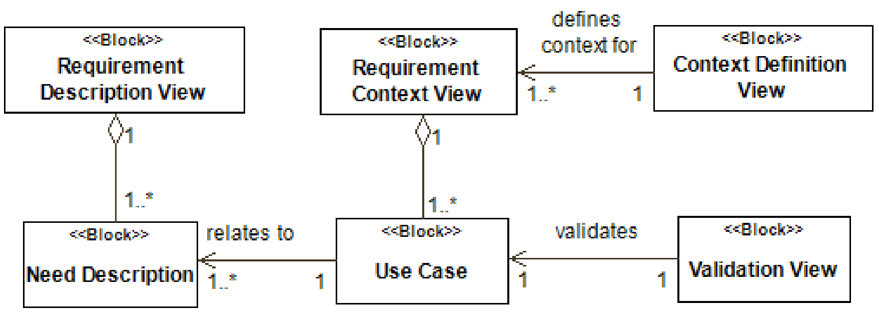

<<Block>> Requirement Description View
<<Block>> Requirement Context View
defines context for
<<Block>> Context Definition View
1..*
1
1
1..*
1
1..*
<<Block>> Need Description
relates to
1..*
1
<<Block>> Use Case
validates
1
1
<<Block>> Validation View

Figure 5.16: Viewpoint Definition View for the Requirement Context Viewpoint

View discussion

The Requirement Context View can be realised in SysML using the Use Case diagram. When creating a Use Case diagram for Requirement Context View, the system boundary in the diagram represents the boundary of a Context (i.e., System Element or Stakeholder Role). The system boundary would then be populated with Need Descriptions that are related to its Context using SysML use case elements. We refer to this Need Description which is put in Context as a Use Case. Notice that the same term is used twice, where 'Use Case' (with upper case) refers to a Need in context as covered in Ontology, and 'use case' refers to the SysML use case element to be used in a Use Case diagram.

After the Use Cases are defined, the relationship between them should also be identified using the appropriate Use Case diagram relationship element. Each Use Case must have a relationship with at least one other Use Case, so there should not be any Use Cases in the diagram that are not connected to other Use Cases in the diagram. On top of the relationship between Use Cases, Use Cases in a particular Context can be related to other Contexts as well. This relationship can be defined by including the external Context as an actor in the diagram and drawing the relationship from the actor to the Use Case inside the system boundary.

As each diagram represents the point of view of one Context, it is possible to create multiple unique Use Case diagrams. The maximum number of Use Case diagrams for each View created for this Viewpoint would be equivalent to the number of Contexts defined in the Context Definition View, where one View is created for each Context. However, it is not necessary to create a Use Case diagram for all the defined Contexts as long as all the Use Cases have been covered.

5.7.6 Validation Viewpoint

As implied by the name, validation views serve as a basis of validation by showing that project Needs can be fulfilled. Validation views are realised on a Scenario basis that is used to validate the Use Cases, which in turn validates the Needs as all Use Cases originate from Needs.

Viewpoint Context View

The Viewpoint Context View for the Validation Viewpoint is shown in Figure 5.17. The main Use Case is to 'Show that Needs can be satisfied by the System' that includes 'Semi-formal Scenario(s)' and 'Formal Scenario(s)', with the former being used to show 'Interactions between system entities that satisfy Use Cases', and the latter showing 'Mathematical-based system properties and the relationship between them'.

Viewpoint Definition View

The Viewpoint Definition View for the Validation Viewpoint is shown in Figure 5.18. The diagram shows that each 'Validation View' represents a 'Scenario', and that one or more 'Scenarios' validate one or more 'Use Cases' as described in the 'Requirement Context View'.

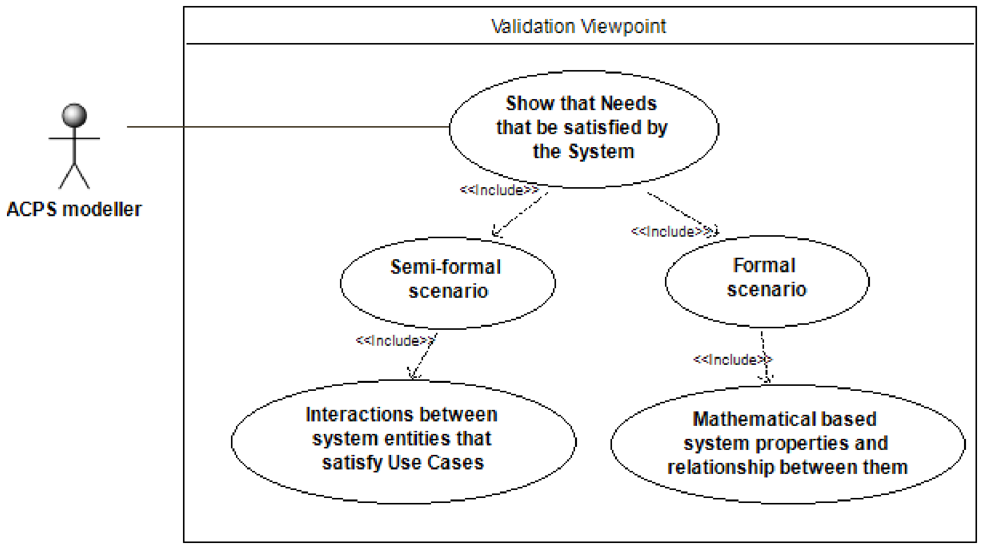

Validation Viewpoint
ACPS modeller
Show that Needs that be satisfied by the System
<<Include>>
<<Include>>
Semi-formal scenario
Formal scenario
<<Include>>
<<Include>>
Interactions between system entities that satisfy Use Cases
Mathematical based system properties and relationship between them

Figure 5.17: Viewpoint Context View for the Validation Viewpoint

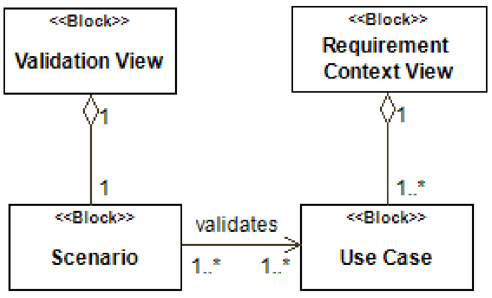


Figure 5.18: Viewpoint Definition View for the Validation Viewpoint

View discussion

As described in the VCV, Validation View can be categorised into Formal Scenarios and Semi-formal Scenarios. The Semi-formal Scenario considers the relationships between entities in the system and demonstrates how their interaction can satisfy the given Use Case. These scenarios can be visualised with SysML using the sequence diagram. It is recommended that each scenario is focused on a single Use Case for clarity.

The one-to-one mapping between Semi-formal Scenario and Use Case approach is only a recommendation as it is possible for a single scenario to satisfy multiple Use Cases and vice versa, where multiple scenarios are used to satisfy a Use Case. However, if multiple scenarios are needed to illustrate a single Use Case, it is worth deliberating whether the Use Case is too complex and could possibly be refined.

The Formal Scenario considers how values of the various properties are being evaluated and how they affect the system. Similar to the Semi-formal Scenario, each scenario in the Formal Scenario is based on Use Cases. Properties of the Formal Scenario can be represented in SysML using constraint blocks, and two different diagrams can be used to demonstrate different aspects of the scenario. A Block Definition Diagram can be used to show the definition of a scenario-related property, each represented with a constraint block. Following the SysML convention, each constraint block should show its parameters and the mathematical-based equation that reveals the relationship between the input and output parameters.

The second type of diagram for Formal Scenarios uses SysML parametric diagrams to show how the defined constraint blocks are connected to each other and applied to the System Elements. In the parametric diagram, each constraint block is connected to either another constraint block or directly to the property of a System Element. The formal approach allows us to reason about the Need and provide a means of measuring performance and effectiveness.

At this stage, it would be beneficial to ensure that the Stakeholder Needs are correct and complete. Here are some examples of how this can be achieved:

- Stakeholder review of the use cases and scenarios. As the stakeholder reviews the scenarios, they would be able to provide feedback from their own perspective on whether their original needs have been sufficiently and correctly captured.
- System Requirements Review (SRR). A formal review can be conducted to determine the complicity and correctness of the system requirement. However, this review would require further documentation materials, such as a Technical Performance Measurement (TPM), that is not covered in this framework.
- Risk assessment methods may help identify off-nominal situations that the system may have to deal with.

5.7.7 Traceability Viewpoint

The Traceability View is used to capture traceability between Needs and other traceable elements in the system. Establishing traceability can be an arduous task. However, one benefit of using a model-based approach is that traceability is inherent in the model. Traceability can serve as a form of validation as it provides a level of rigour in relation to the Needs.

Viewpoint Context View

The Viewpoint Context View for the Traceability Viewpoint is shown in Figure 5.19. The main Use Case is to ‘Provide traceability to and from Needs’ that includes ‘Traceable Elements’. It also shows that the two types of ‘Traceable Elements’ are the ‘View Element’ and ‘View’.

Viewpoint Definition View

The Viewpoint Definition View for the Traceability Viewpoint is shown in Figure 5.20. The diagram shows that each ‘Validation View’ represents a ‘Scenario’, and that one or more ‘Scenarios’ validate one or more ‘Use Cases’ as described in the ‘Requirement Context View’.

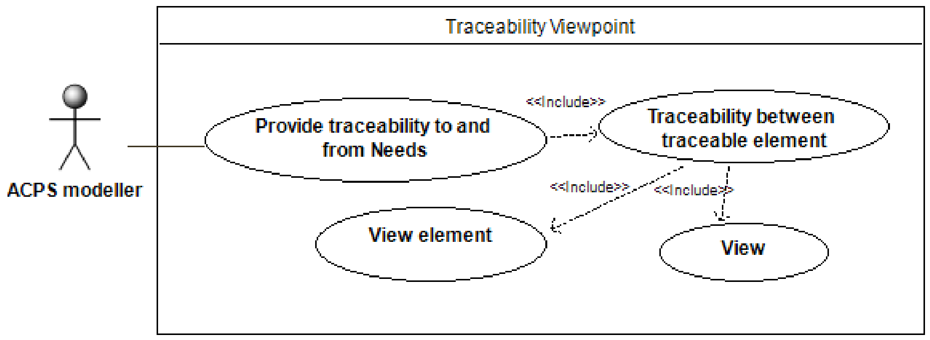


Figure 5.19: Viewpoint Context View for the Traceability Viewpoint

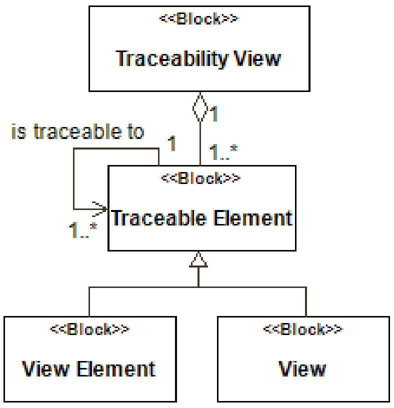


Figure 5.20: Viewpoint Definition View for the Traceability Viewpoint

View discussion

When using a modelling language such as SysML to create a model, there is a default traceability at the meta-model level. For example, there is a relationship between operation from a block to an activity of a state. However, the traceability that this View intends to capture is at the application level of the domain model (i.e., the ACPS model).

These traceability relationships can take many forms. The VCV shows a type of 'Traceable Element' in 'View Element'. The 'View Element' represents any element that exists within the Ontology defined in Chapter 4. Although it should be possible to derive traceability from the Use Case to all elements defined in the Ontology, this View will focus on elements that have a direct relationship with Use Cases.

The elements that have a direct relationship with Use Cases are Scenario and Need. Needs in our model are represented as Requirement and are captured by the Requirement Description View as View Elements. Although Needs (represented as Requirement) is a View Element, the Scenario in our model is not captured as a View Element. The Scenario in our model is captured with the Validation View, with each View representing a Scenario. This is why it is necessary that 'View' is also a type of 'Traceable Element' as this View should draw a relationship between the Validation View and Use Cases.

As mentioned earlier, it should be possible to extend and show traceability between all ontology elements. One way to extend this traceability is to show traceability between elements that have a direct relationship with Requirements, such as System Elements and Source Elements.

The traceability View can be visualised in SysML with a block definition diagram or a requirement diagram. The choice of diagram used would depend on the elements covered in the diagram. The block definition diagram can be used for any type of Traceable Element. However, the requirement diagram might be more suitable for the relationship between Needs and Use Cases.

Diagrams for the Traceability View should contain a set of Traceable Elements and their connections with each other, showing their relationship. This relationship should be a type of dependency. When using a requirement diagram, there are default types of requirement relationships that can be used to draw these relationships, such as the refine, satisfy, and trace relationship. The relationship for this View is not strictly limited to these predefined options. Where necessary, it is encouraged to define a new stereotype to describe the nature of the dependency.

To clarify on the elements involved in building Traceability Views, the following list shows the different types of traceability between Traceable Elements and the example dependency that can be used to describe its relationship. Note that for a complete traceability view, this View should be combined with the System Traceability View from the Co-modelling Perspective.

1. Traceability between Use Cases and Need Description. A refine dependency (or «refine» stereotype) can be used to show how one or more Use Cases are traceable to a Need Description and refines them.
2. Traceability between Use Cases and Validation Views (representation of Scenario). A custom stereotype such as '«validate»' can be used to draw dependency from the Validation View to Use Case to show how each Use Case is validated by a specific Scenario.
3. Traceability between Need Description and Source Elements. The trace dependency (or «trace» stereotype) can be used to show the traceability of a Need Description to the Source Element that it is derived from.
4. Traceability between Need Description and System Elements. Unlike the previous examples on this list, System Elements are not modelled in any of the View from the Need Perspective as this will be covered in the CPS Perspective instead. Therefore, it is important that the necessary Views from the CPS Perspective (i.e., CPS Identification Views) are created before this traceability can be finalised. Once the System Elements have been defined, the satisfy dependency (or «satisfy» stereotype) can be used to show how System Elements can be traced to meet the Need Description.
5. Traceability between Use Case and System Elements. Trace dependency can be used to draw traceability

between System Elements and Use Cases.

6. Traceability between Co-simulation and Validation Views. If co-simulation is used, the co-simulation should be based on part of or multiple Scenarios depicted in the Validation Views. Trace dependency can be used to draw traceability between Co-simulation and Validation Views.

5.8 Viewpoint Definitions - CPS Perspective

This section presents the Viewpoints of the CPS Perspective for ACPSAF. The CPS Perspective will be covering all the System-related concepts for ACPS, focusing on the structure that forms the system. The perspective is called CPS Perspective instead of ACPS Perspective because the Viewpoints covered here are targeted at CPS structure and non-autonomous behaviour. All autonomy-related concerns are addressed in the Autonomy Perspective.

Our decision to treat ACPS as CPS with added autonomy comes with certain benefits and limitations. The main benefit of this approach is that the framework would be relevant and highly-adaptable for projects where there are existing systems, infrastructures, or policies that the ACPS should be built upon. For example, autonomous smart grids are typically built on top of existing power grid systems since the idea of rebuilding the whole infrastructure is not an option. The CPS Perspective allows the designed ACPS to start by designing the existing system and its boundaries before exploring the different designs for autonomous behaviour. Although this would be desirable for projects that start with an existing system, this would be a limitation for projects that want to start with the desired autonomous capability before exploring different design possibilities that have the potential of coming up with a novel ACPS design that is inherently different from the existing system.

The Viewpoints defined in CPS Perspective are based on System Perspective from MBSEAF [240]. Several Viewpoints are generalised, with some others specialised to fit for ACPS purposes. The Viewpoint names in this Perspective favour the term ‘System’ rather than ‘CPS’ (e.g., System Identification Viewpoint) primarily as a way to follow the common convention.

5.8.1 System Identification Viewpoint

The System Identification View is used to identify Systems, CPS, ACPS, System Elements and the hierarchical relationship between them. The focus of this View is to capture all the structural components forming the system. This View should also show the hierarchical structure between each element in the system. When the component identified is at the System Element level, it should also be identified whether the System Element is an Actuator, Sensor, Part, or Controller.

Viewpoint Context View

The Viewpoint Context View for the System Identification Viewpoint is shown in Figure 5.21. The main Use Case is to 'Identify systems and its components' which includes 'System' that can be typed into 'CPS' and then further typed into 'ACPS'. The main Use Case also includes 'Show System Elements' and when the identified system is a 'CPS', it should further identify the System Elements of the CPS as Sensor, Actuator, Part, or Controller. Note that although Figure 5.21 shows that the system is identified by an ACPS modeller, in reality, any stakeholder can contribute to the system identification.

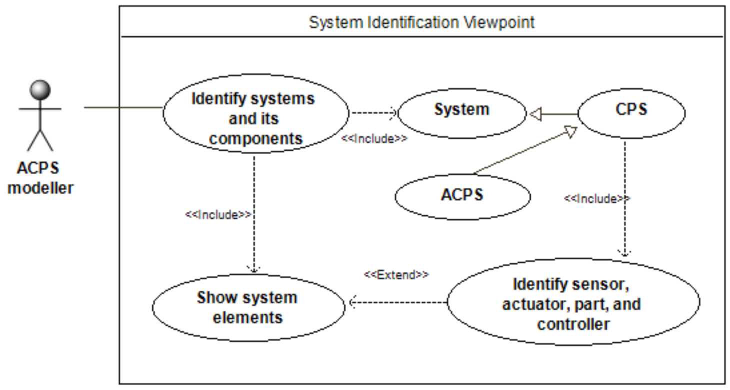

System Identification Viewpoint
ACPS modeller
Identify systems and its components
<<include>>
System
CPS
ACPS
<<include>>
<<include>>
<<Extend>>
Show system elements
Identify sensor, actuator, part, and controller

Figure 5.21: Viewpoint Context View for the System Identification Viewpoint

Viewpoint Definition View

The Viewpoint Definition View for the System Identification Viewpoint is shown in Figure 5.22. The diagram shows that each 'System Identification View' contains one or more 'Systems'. 'ACPS' is a type of 'System', and each 'ACPS' contains one or more 'System Elements'. There are four types of 'System Elements', which are 'Actuator', 'Sensor', 'Part', and 'Controller'. It also shows that the 'System Identification View' has a relationship with the 'System Structure View' through 'System'.

View discussion

The System Identification View can be realised in SysML using the block definition diagram, where each system component is represented by a block. At this stage of development, only the name of the block is required without going deeper into the properties or operations of the blocks. The stereotypes of «System», «CPS», «ACPS», and «System Element» can be given to the block to represent the types of Ontology Element it represents. The System Elements making up the CPS and ACPS should also be further identified into «Sensor», «Actuator», «Part», and «Controller».

For these Views, we are interested in drawing the hierarchical relationship of system components in terms of their composition and types. Our approach regards System Element(s) as a complex type that could have its

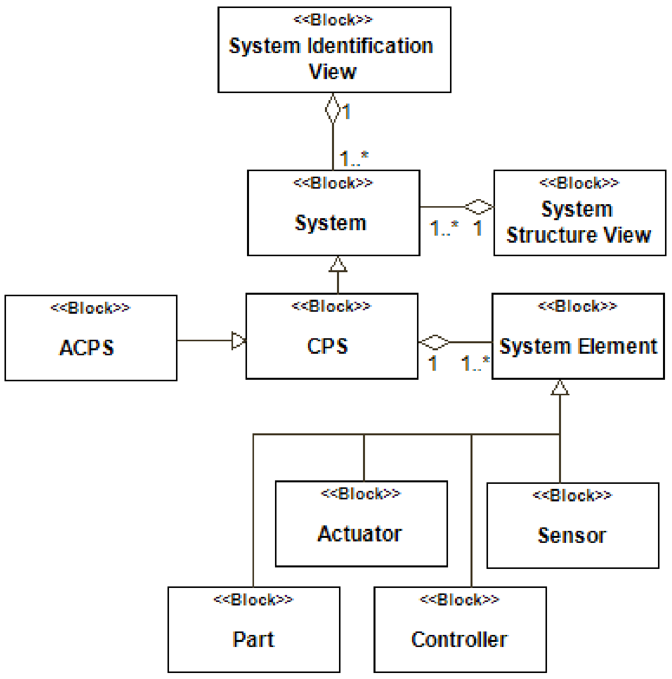


Figure 5.22: Viewpoint Definition View for the CPS Identification Viewpoint

own properties and functions. For that reason, block properties such as parts or reference properties should not be used to represent the hierarchical relationship of such elements. The relationship of these elements should be represented with SysML composition, aggregation, and generalisation. Less 'structural' relationships such as the interactions between System Elements (typically an association relationship) should not be included in these Views.

Any number of block definition diagrams can be created to cover the System Identification Views. When dealing with large systems, attempting to fit the whole model into a single diagram can be counter productive as it gets too complex to be understood. Another reason to use multiple Views is that it may clarify the level of abstraction, which can get complicated when dealing with Systems and System Elements. As mentioned in the Ontology, the approach of System being composed of System Elements can be a recursive when the System Elements are further deconstructed into more System Elements. This means that a block can be both a System and System Element at the same time depending on the context. This is a common occurrence in MBSE, and using several Views strategically at a different level of abstraction could encourage the model to be well understood.

5.8.2 System Structure Viewpoint

Similar to System Identification View, the System Structure View is interested in the structure and relationship of systems and its components. However, this view will explore deeper into each of the System Elements and the interactions between them, accounting their System Properties and System Functions.

Viewpoint Context View

The Viewpoint Context View for the System Structure Viewpoint is shown in Figure 5.23. The main Use Case is to 'Show the structure of Systems and System Elements' that includes 'Show relationships between them' and 'Properties and functions', which is typed as 'System Function' and 'System Property' as covered in the Ontology.

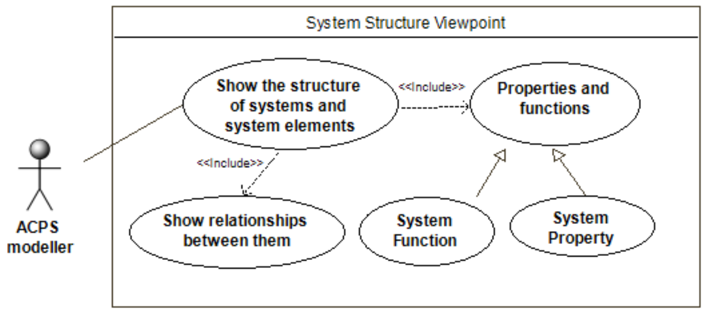

System Structure Viewpoint
Show the structure of systems and system elements
<<include>>
Properties and functions
<<include>>
Show relationships between them
System Function
System Property
ACPS modeller

Figure 5.23: Viewpoint Context View for the System Structure Viewpoint

Viewpoint Definition View

The Viewpoint Definition View for the System Structure Viewpoint is shown in Figure 5.24. Similar to System Identification View, the 'System Structure View' would contain a 'System' that is eventually decomposed into 'System Elements'. However, this View also shows the 'System Property' and 'System Function' for each 'System' and also an indication that 'System' and 'System Element' may interact with each other. The 'System Structure View' has a relationship with three other Views in the CPS Perspective. The first relationship is with 'System Identification View' as it uses the system-related elements identified there. This View also uses a 'System Function' that may be provided by an 'Interface' defined in the 'System Interface View'. The last relationship is with 'System Configuration View' that shows how the systems may be configured.

View discussion

As shown in the VDV, most of the Ontology Elements involved in this View overlap with those in the System Identification View. However, this View places more focus on each system component, focusing on the interactions between System Elements and taking into account their respective System Functions and System Properties.

System Structure Views can be visualised in SysML using a block definition diagram with block elements already defined in the System Identification Views. The main activity in creating this View would be to define the System Function and System Property for each of the model blocks. System Property refers to

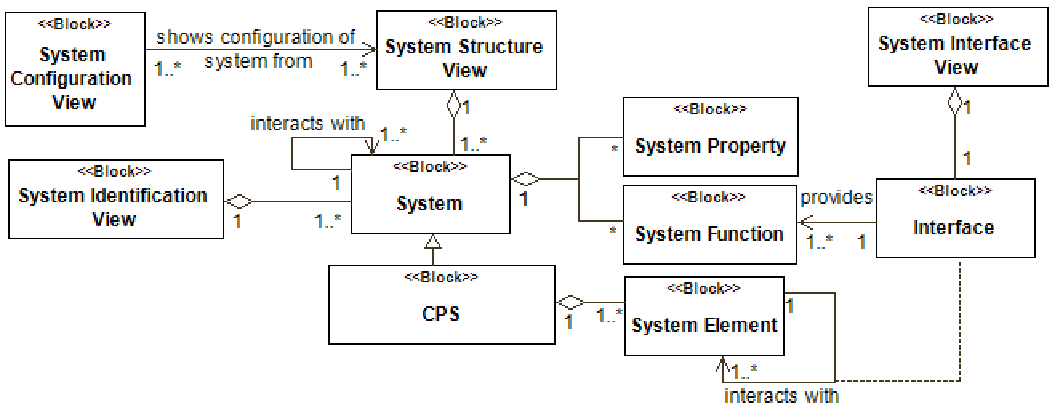


Figure 5.24: Viewpoint Definition View for the System Structure Viewpoint

the quantifiable characteristic of a System or System Element that comes with a form of units or measurable dimensions. Value property can be used to represent the System Property of the respective blocks.

System Function is an action, a task, or activity performed by the System Element to achieve a desired outcome. The System Function is realised in SysML as operations to its respective owning blocks. Each operation has a name with an optional in, out, and return parameter. These parameters need to be modelled here as it shows how one System Element may be connected and interact with other elements in the system. However, explicit implementation of these functions will be covered in the System Behaviour Viewpoints.

Interactions between Systems or System Elements can be represented with an association relationship. The interactions shown in this View are descriptive as the functional interactions between System Elements require the use of 'Interface'. This means that the association relationship would describe the nature of the interactions between System Elements without having to provide further details.

5.8.3 System Interface Viewpoint

The System Interface View defines the Interfaces that will be used by the System (and its sub-types) and System Elements to interact externally. All the interfaces used in the model should be shown in this View.

Viewpoint Context View

The Viewpoint Context View for the System Interface Viewpoint is shown in Figure 5.25. The main Use Case is to 'Define interfaces' that are targeted 'For System and sub-types' and 'For System Element(s)'. The main Use Case also includes 'Operations of each Interfaces', which would include 'parameter' that is typed into 'in', 'out', and 'return'.

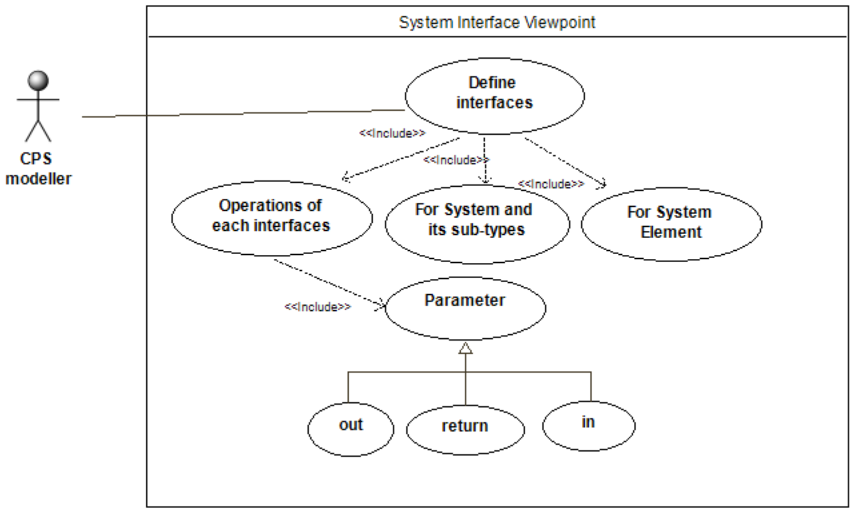


Figure 5.25: Viewpoint Context View for the System Interface Viewpoint

Viewpoint Definition View

The Viewpoint Definition View for the System Interface Viewpoint is shown in Figure 5.26. It shows that 'System Interface View' contains one or more 'Interfaces', giving it a direct relationship with the 'System Configuration View' that uses these Interfaces. This View also has a relationship with 'System Structure View' as the 'Interfaces' provide 'System Functions' that are used in the 'System' covered in 'System Structure View'.

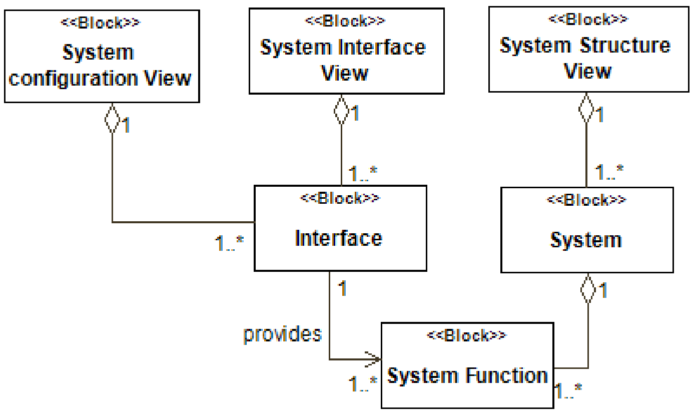

<<Block>> System configuration View
<<Block>> System Interface View
<<Block>> System Structure View
1
1
1
1..*
1..*
<<Block>> Interface
<<Block>> System
1..*
1
1
provides
<<Block>> System Function
1..*
1..*

Figure 5.26: Viewpoint Definition View for the System Interface Viewpoint

View discussion

The System Interface View can be visualised in SysML with a block definitions diagram using interface blocks to represent each interface in the model. Each Interface would consist of a name and an optional set of operations. Similar to the operations for System Functions in the System Structure View, each operation has a name and an optional set of parameters (i.e., in, out, and return). Apart from the three types of parameters, SysML contains a fourth type of parameter known as the inout parameter. This refers to a parameter that could be either in or out depending on the implementation of the operation. As shown in the VCV, our framework does not deal with an inout parameter and will only utilise the other three.

Although similar, an Interface is not equal to a System Function. The VDV states that an Interface provides one or more System Functions. However, an Interface is not simply a collection of System Functions. A System Function can be derived from the owning System Element block or an Interface. If a System Function is directly derived from a System Element, it means that the System Function would only be visible internally within the block and cannot be used to interact with elements outside the block. Meanwhile, System Functions derived from an Interface would be visible to at least two separate System Element blocks. One System Element would be providing the Interface, while the others would require the Interface. It is through this Interface connection that the System Elements may interact with each other.

5.8.4 System Configuration Viewpoint

The System Configuration View is used to show the structural configuration of the overall system, accounting for all the system components together with the Interfaces between them.

Viewpoint Context View

The Viewpoint Context View for the System Configuration Viewpoint is shown in Figure 5.27. The main Use Case is to 'Show configuration of System' that includes 'System and its sub-types', 'System Elements and its sub-types', and 'Interface connection between System Elements'.

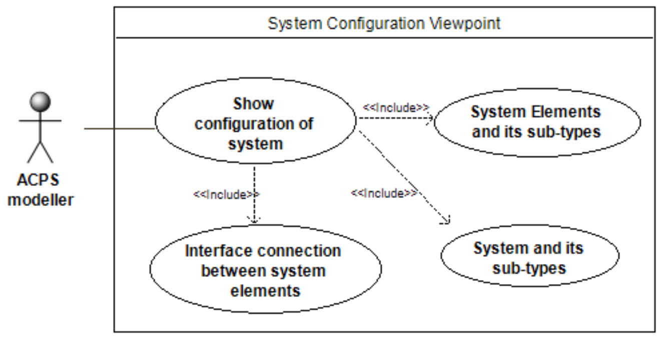

System Configuration Viewpoint
ACPS modeller
Show configuration of system
<<include>>
System Elements and its sub-types
<<include>>
<<include>>
Interface connection between system elements
System and its sub-types

Figure 5.27: Viewpoint Context View for the System Configuration Viewpoint

Viewpoint Definition View

The Viewpoint Definition View for the System Configuration Viewpoint is shown in Figure 5.28. The diagram shows that 'System Configuration View' contains one or more 'Systems' and 'Interfaces'. As explored in the previous Views, this means that the 'System Configuration View' has a direct relationship with the 'System Interface View' and 'System Structure View' as those are the Views where 'Interface' and 'System' structure are defined. ‘System Configuration View’ has a relationship with ‘System Structure View’ instead of System Identification View because although the System Identification View identifies the System, ‘System Structure View’ defines the structure of the system in terms or interaction between System Elements, and the ‘System Configuration View’ shows the realisation of such structure by connecting System Elements with Interfaces.

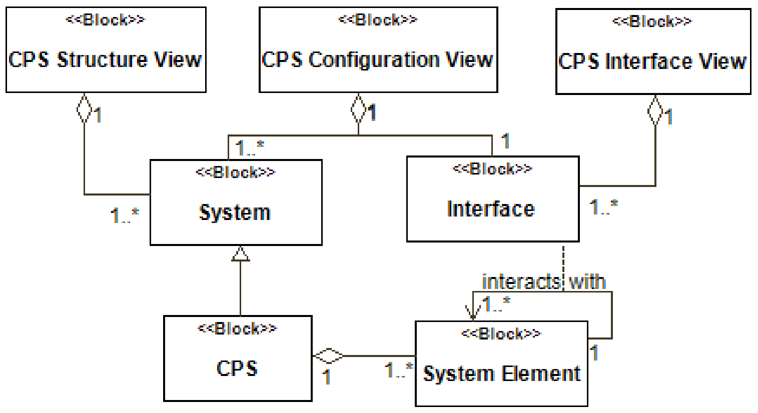


Figure 5.28: Viewpoint Definition View for the System Configuration Viewpoint

View discussion

The System Configuration View can be visualised in SysML with an internal block diagram. System Elements are represented using Parts connected via Interfaces or Item Flows. Ports are used as the connection points for these Interfaces and item Flows. Ports can have required or provided Interfaces attached and be attached multiple times into an Interface.

The System Configuration View should allow users to visualise how systems are actually configured. For example, a car could have a System Element for ‘Wheel’. The System Configuration View should show four instances (or Parts) of type ‘Wheel’ and show how they are connected to the main body of the car.

5.8.5 System Behaviour Viewpoint

The System Behaviour View is used to define the behaviour of System and System Elements. This includes the behaviour of System Function as well.

Viewpoint Context View

The Viewpoint Context View for the System Behaviour Viewpoint is shown in Figure 5.29. The main Use Case is to ‘Define behaviour of System and System Elements’ that includes ‘System Functions’, ‘System constraint’, ‘Flow of activity’, ‘System state’, and ‘Interaction between System Elements’.

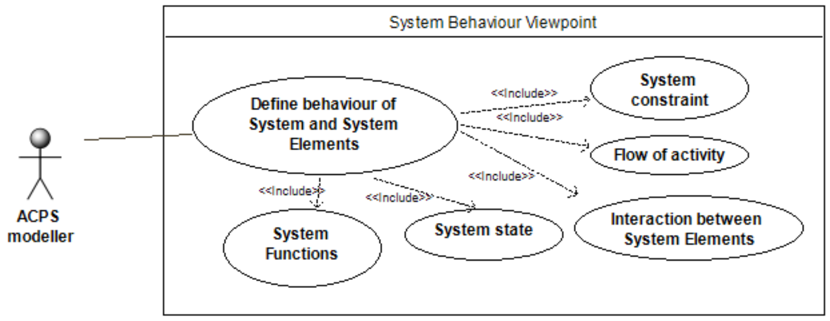

System Behaviour Viewpoint
ACPS modeller
Define behaviour of System and System Elements
System constraint
Flow of activity
Interaction between System Elements
System state
System Functions
<<Include>>

Figure 5.29: Viewpoint Context View for the System Behaviour Viewpoint

Viewpoint Definition View

The Viewpoint Definition View for the System Behaviour Viewpoint is shown in Figure 5.30. The diagram shows that 'System Behaviour View' contains one or more 'Systems', which also mean that it could contain one or more 'System Elements', 'System Functions', and 'System Properties'.

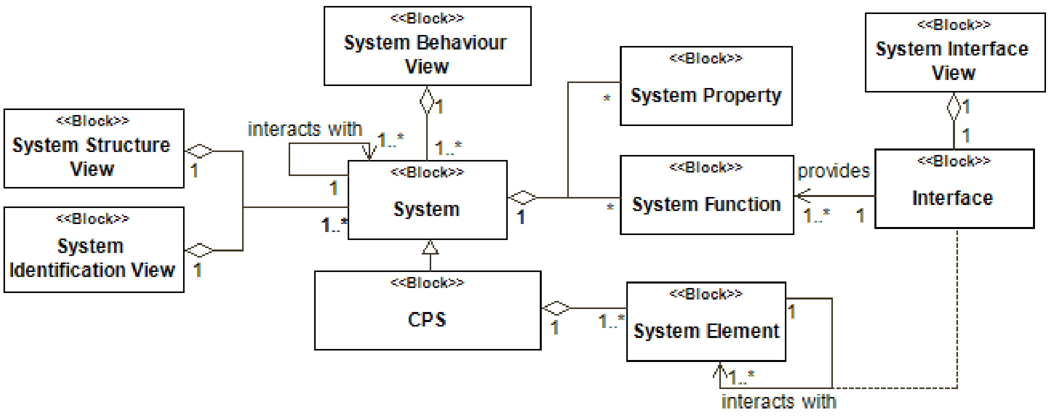

<<Block>> System Behaviour View
<<Block>> System Structure View
<<Block>> System Identification View
<<Block>> System
<<Block>> CPS
<<Block>> System Property
<<Block>> System Function
<<Block>> System Element
<<Block>> System Interface View
<<Block>> Interface
interacts with
provides
interacts with
1
1..*
*

Figure 5.30: Viewpoint Definition View for the System Configuration Viewpoint

View discussion

SysML provides several diagrams to capture the behaviour of a system. These diagrams include: Activity diagram, Sequence diagram, State Machine diagram, and Use Case diagram. The System Behaviour View can be captured with any behaviour diagrams except a Use Case diagram. Use Case diagrams should not be needed here as all the Use Cases should already be captured in the Requirement Context View.

Each of the behaviour diagrams in SysML have its own strengths and weaknesses. It is very common that multiple diagrams using multiple types of behaviour diagrams are used to capture the behaviour of a system [354]. ACPSAF users are expected to be familiar with the SysML behaviour diagrams and should be able to decide what type of diagram is best to define the behaviour of a System, System Elements, and System Functions.

When working on the System Behaviour View, the following questions might be useful in discovering which behaviour needs to be defined:

1. Have the behaviour of all the Systems, System Elements, and System Functions been defined?
2. Have all the behaviour needed to satisfy the Use Cases defined in the Requirement Context View been defined?

On top of the three behavioural diagrams mentioned, System Behaviour View could include a parametric behaviour as well. Parametric behaviour is behaviour that is best expressed using a network of mathematics and logical constraint between System Properties in a system. For these types of cases, Parametric diagrams are used to create systems of equations that can constrain the properties of blocks.

5.9 Viewpoint Definitions - Autonomy Perspective

5.9.1 Autonomy Structure Viewpoint

The Autonomy Structure View is used to organise the System Elements involved and required to satisfy an Autonomy Requirement identified in the Requirement Description View. As System Elements and System Structure have been identified in the CPS Perspective, this view is more for organisational purposes as the System Elements identified here will be carried to the Autonomy Context Viewpoints.

Viewpoint Context View

The Viewpoint Context View for the Autonomy Structure Viewpoint is shown in Figure 5.31. The main Use Case is to 'Show System Elements needed for an Autonomy Requirement' that includes 'Autonomy Requirement as context' and 'System Elements' which are typed into 'actuator', 'sensor', 'part', and 'controller'.

Viewpoint Definition View

The Viewpoint Definition View for the Autonomy Structure Viewpoint is shown in Figure 5.32. The diagram shows that each 'Autonomy Structure View' contains one or more 'ACPSs'; and each 'ACPS' is composed of one or more 'System Elements'. 'System Elements' are typed into 'Actuator', 'Sensor', 'Part', and 'Controller'.

View discussion

The Autonomy Structure View can be produced in SysML using a block definition diagram where each ACPS and System Element is presented as a block. One Autonomy Structure View should be created for each Autonomy Requirement identified in the Requirement Description View. As mentioned earlier, the Autonomy Structure View is more for organisational purposes as the model should already be defined by the

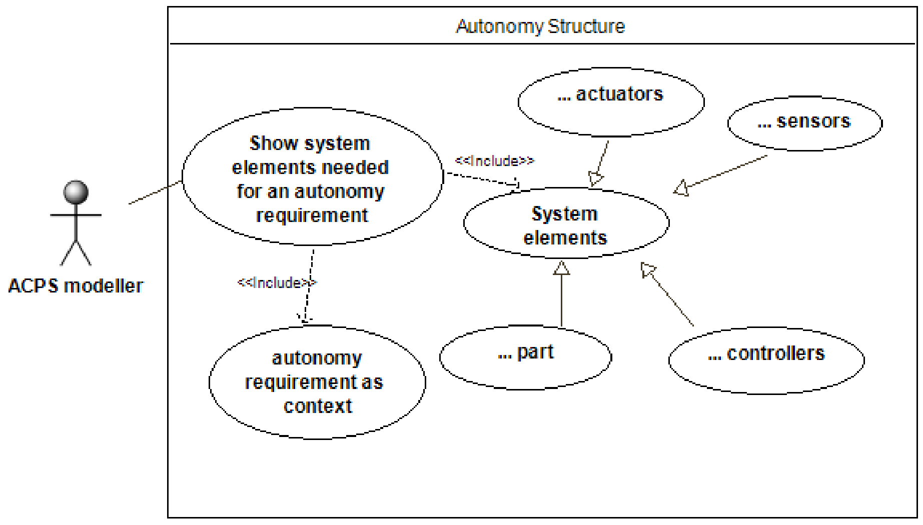


Figure 5.31: Viewpoint Context View for the Autonomy Structure Viewpoint

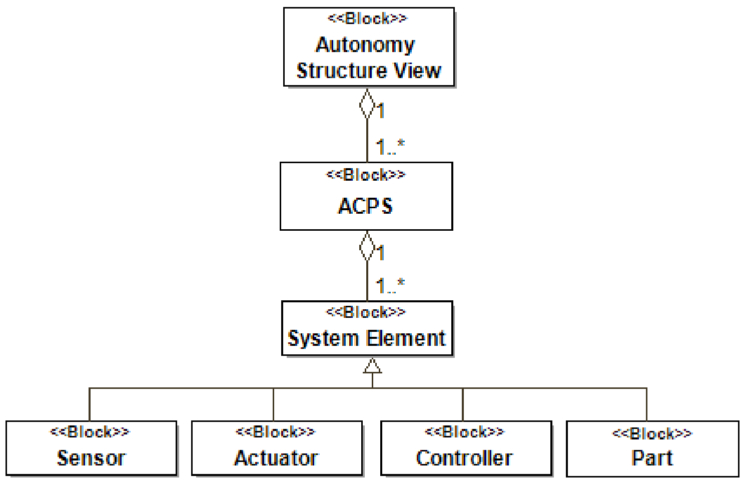


Figure 5.32: Viewpoint Definition View for the Autonomy Structure Viewpoint

CPS Perspective. This is also reflected in the RDV as there is no rule stating that the Autonomy Structure View must be created in the project. This means that this Viewpoint is optional. However, this Viewpoint is beneficial for organising System Elements for the next Viewpoints.

Each Autonomy Structure View must contain at least one controller, sensor, and actuator. This is because the ACPS Ontology defines ACPS as a closed control loop system that interacts and adapts with its environment through the use of sensors and actuators. Hence, having one controller, sensor, and actuator is the bare minimum to fulfilling an Autonomy Requirement. Naturally, it could involve more System Elements.

5.9.2 Autonomy Context Viewpoint

The Autonomy Context View is concerned with ensuring that all Autonomy Requirements in the ACPS can be satisfied. The main activity is to take each Autonomy Requirement into the context of a Controller responsible for that Autonomy Requirement. It generates Use Cases that require the use of Sensors and Actuators to be satisfied.

Viewpoint Context View

The Viewpoint Context View for the Autonomy Context Viewpoint is shown in Figure 5.33. The main Use Case is to 'Identify the context of Autonomy Requirement' that includes 'Define relationship from Use Cases to Sensors and Actuators' and 'Consider Autonomy Requirement in the context of Controller', which also includes 'Identify property to control' and 'Identify data required'.

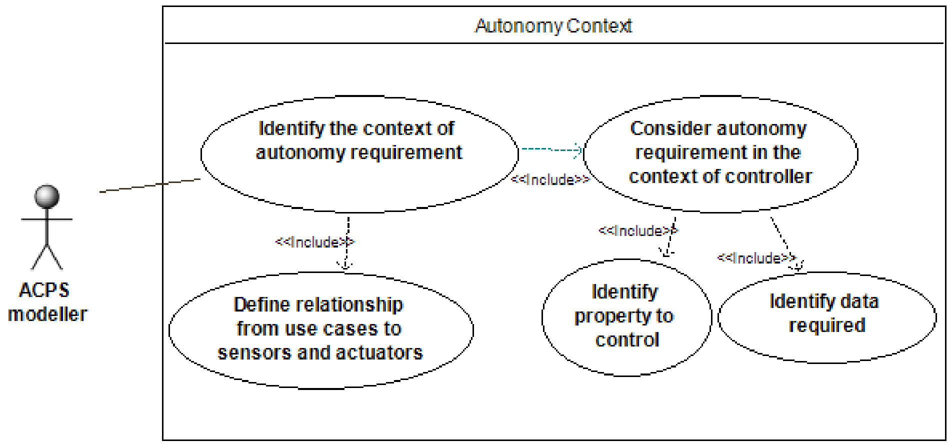

Autonomy Context
Identify the context of autonomy requirement
Consider autonomy requirement in the context of controller
<<Include>>
<<Include>>
<<Include>>
<<Include>>
ACPS modeller
Define relationship from use cases to sensors and actuators
Identify property to control
Identify data required

Figure 5.33: Viewpoint Context View for the Autonomy Context Viewpoint

Viewpoint Definition View

The Viewpoint Definition View for the Autonomy Context Viewpoint is shown in Figure 5.34. The diagram shows that each 'Autonomy Context View' contains one or more 'Use Cases', where each 'Use Case' is considered in the context of a 'Controller' and relates to one or more 'Autonomy Requirements'. One or more 'Actuators' and 'Sensors' are needed to enable one or more 'Use Cases'.

View discussion

The Autonomy Context View can be realised in SysML using a Use Case diagram. When creating a Use Case diagram for Autonomy Context View, the system boundary in the diagram represents the boundary of a Context from a Controller. The Controller is the programmable computer system in the ACPS that is responsible for decision-making in replacement of a human operator. Therefore, it is natural that the Autonomy Requirement should be considered in the context of the Controller.

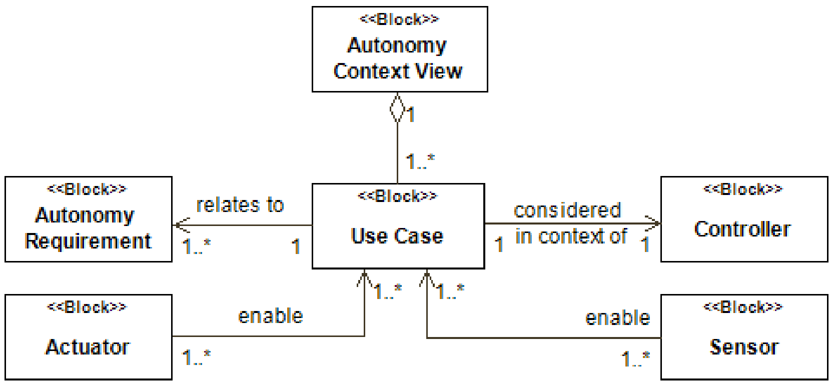


Figure 5.34: Viewpoint Definition View for the Autonomy Context Viewpoint

Each Autonomy Context View has one main Use Case, and this Use Case should be based on the Autonomy Requirement. The main Use Case can be refined into smaller, more manageable Use Cases. Outside the boundary of the Controller, there will be Sensors and Actuators that are needed to satisfy the Use Cases. To be precise, the main Use Case should be refined into two types of branches. One type of Use Case branch would lean towards getting the type of actuation desired. This Use Case branch would eventually be connected to the Actuators that would provide the specified type of Actuation. The other branch leans towards receiving data from Sensors. These Use Cases would eventually connect to Sensors that provide the required type of sensory data.

5.9.3 Autonomy Behaviour Viewpoint

Similar to the System Behaviour Viewpoint, the Autonomy Behaviour Viewpoint is concerned with capturing the behaviour of ACPSs. However, the Autonomy Behaviour View would only describe System behaviours that are related to the Use Cases defined in Autonomy Context Views.

Viewpoint Context View

The Viewpoint Context View for the Autonomy Behaviour Viewpoint is shown in Figure 5.35. The main Use Case is to ‘Define behaviour of Controller to satisfy Autonomy Requirement’ that includes ‘System Functions’ and ‘Interaction with other System Elements’ such as ‘actuator’ and ‘sensors’.

Viewpoint Definition View

The Viewpoint Definition View for the CPS Interaction Viewpoint is shown in Figure 5.36. The diagram shows that each ‘Autonomy Behaviour View’ contains one or more ‘Controllers’, where one or more ‘Controllers’ define the behaviour of the ‘ACPS’ and satisfy one or more ‘Autonomy Requirements’.

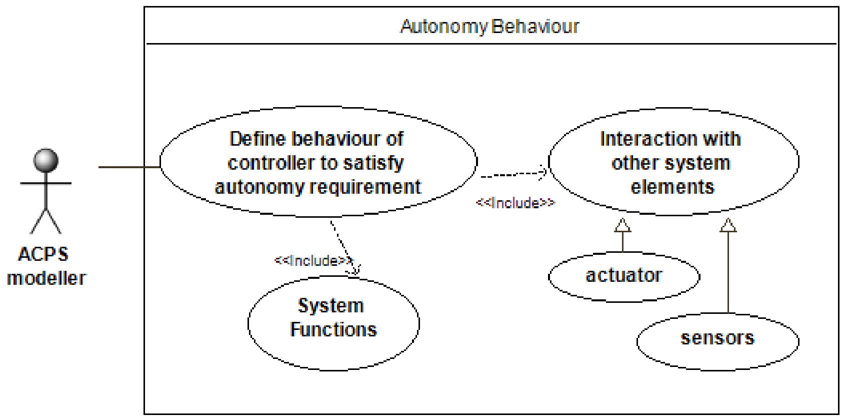

Autonomy Behaviour
ACPS modeller
Define behaviour of controller to satisfy autonomy requirement
<<Include>>
Interaction with other system elements
<<Include>>
System Functions
actuator
sensors

Figure 5.35: Viewpoint Context View for the Autonomy Behaviour Viewpoint

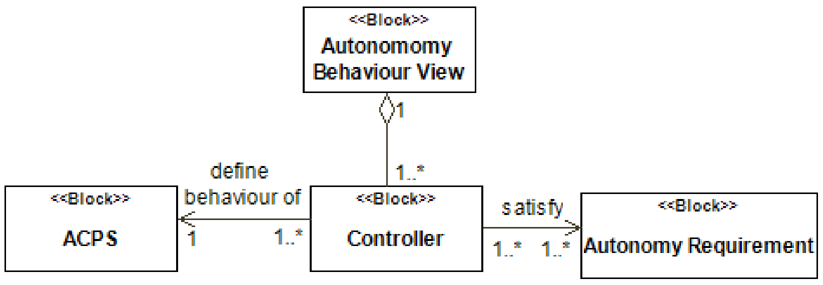


Figure 5.36: Viewpoint Definition View for the Autonomy Behaviour Viewpoint

View discussion

As the Autonomy Behaviour Viewpoint is only concerned with ACPS behaviour that is required to satisfy Autonomy Requirements, only the Controllers are involved in the Autonomy Behaviour View. An exception would be when describing the interaction between Controllers with other System Elements. However, the Controllers would still be the main Lifeline (referring to the Lifeline element in a sequence diagram).

Similar to the System Behaviour Viewpoint, all types of behaviour diagrams can be used to describe the behaviour of the Controller. However, it is unlikely that the Use Case diagram is needed as the Use Cases should be captured in the Autonomy Context View instead. ACPSAF does not define a strategy for the implementation of the Controllers to achieve the Autonomy Requirement. Therefore, users would have to use their expertise to realise the Use Cases into behavioural implementation.

5.10 Viewpoint Definitions - Co-modelling Perspective

5.10.1 Architecture Structure Viewpoint

The Architecture Structure View is used to identify the Models making up the Co-model that represent the ACPS. The relationship on Models making up the Co-model is shown in a hierarchical relationship similar to the System Identification View.

Viewpoint Context View

The Viewpoint Context View for the Architecture Structure Viewpoint is shown in Figure 5.37. The main Use Case is to 'Define co-model' that includes 'Create model to represent ACPS'. The types of models would include 'CT model' and 'DE model'. Although our research would only cover the CT and DE model types, the co-modelling approach here is not limited to the CT and DE model as long as the model can be exported as an FMU according to the FMI standard.

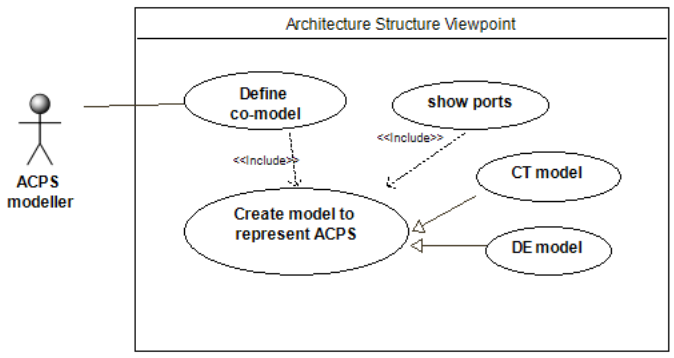


Figure 5.37: Viewpoint Context View for the Architecture Structure Viewpoint

Viewpoint Definition View

The Viewpoint Definition View for the Architecture Structure Viewpoint is shown in Figure 5.38. The diagram shows that the 'Architecture Structure View' contains one 'co-model' that represents one 'ACPS'. It also shows that the 'co-model' is made up of any number of 'CT model(s)' and 'DE model(s)'.

View discussion

The Architecture Structure View is concerned with capturing the structure of a co-model designed to represent the ACPS. The co-model is composed of several constituent models, typically at least one CT model and DE model; whereas a CT model is used to capture the structure and behaviour of physical components in the ACPS and a DE model is used to capture the cyber side of ACPS (e.g., controller). Similar to the System Identification View, the Architecture Structure View can be captured with BDD. Starting with the ACPS as the top level Block within the co-model, we draw the hierarchical relationship of system components in terms of the Model making up the Co-model.

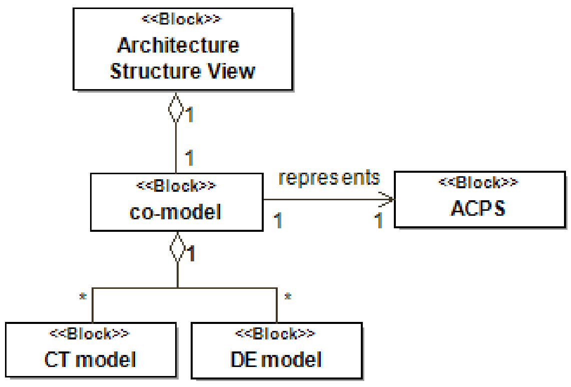


Figure 5.38: Viewpoint Definition View for the Architecture Structure Viewpoint

Each constituent model is represented as a Block and the stereotype «Comodel» is given to the Block for identification. Despite its name, the stereotype does not mean that the stereotyped Block is a co-model, but that the Block is a Model making up a co-model. There are two attributes that will be useful to record for each of the Models. We suggest adding two tags to the «Comodel» stereotype. First is 'Type', which is used to capture whether the Model is CT or DE. The second is 'Platform', which records the modelling tool used to create that Model.

Co-modelling only makes sense when the Models making up the co-model communicate and share their parameters with each other during co-simulation. These shared parameters are categorised into FMU input and FMU output, where an FMU input for a Model would refer to an external parameter shared by another FMU Model and the FMU output would mean the parameter from this Model is being shared with other Models in the co-model. We can define these inputs and outputs as SysML Flow Ports on the Models. The In and Out direction can be set on the Ports to indicate their reference to FMU input or output respectively.

5.10.2 Connection Viewpoint

A Connection View is used to capture the connection between each FMU, showing the parameter shared and received by each FMU in the co-model for a particular co-simulation. Instances of the FMUs should be shown as how they would be executed for a co-simulation. If multiple co-simulations are needed with different FMU configurations, multiple Connection Views would be produced to reflect this.

Viewpoint Context View

The Viewpoint Context View for the Connection Viewpoint is shown in Figure 5.39. The main Use Case is to 'Show connections between FMUs' that includes 'from output ports' and 'to output ports'.

Viewpoint Definition View

The Viewpoint Context View for the Connection Viewpoint is shown in Figure 5.40. The diagram shows that

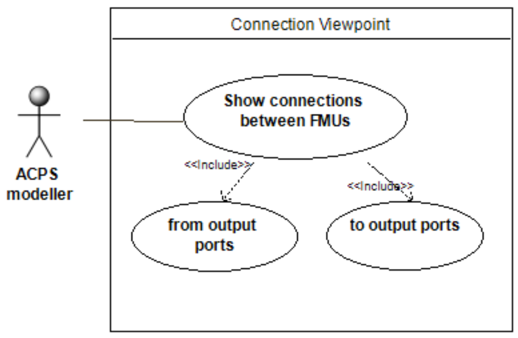


Figure 5.39: Viewpoint Context View for the Connection Viewpoint

'Connection View' contains one or more 'FMUs'. Each 'FMU' represents a 'model', and one or more 'models' represent the 'ACPS'. It also shows that one or more 'FMUs' are used in each 'Co-simulation'.

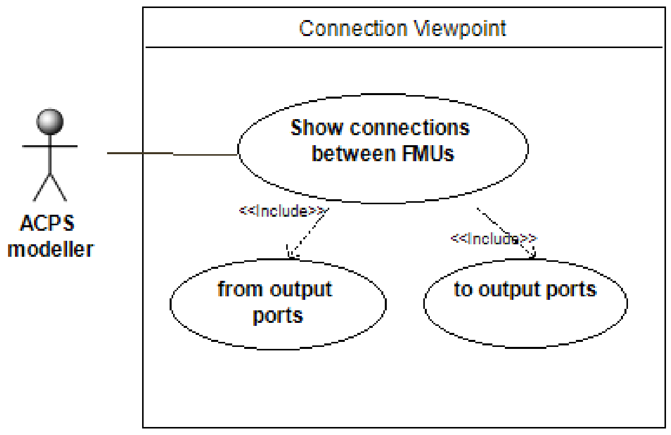


Figure 5.40: Viewpoint Definition View for the Connection Viewpoint

View discussion

The Connection Viewpoint is concerned with capturing how the FMU input and output in the co-model is connected to each other. The Connection View can be realised with the Internal Block Diagram, created from the top-level Block as shown in the Architecture Structure View that represents the co-model. Each of the FMUs should be represented as a SysML Part. The Ports for each Part should reflect its Block type, which is shown in the Architecture Structure View. As several FMU configurations might be used for different co-simulations, several Connection Views might be needed.

5.10.3 System Traceability Viewpoint

Translation from an architecture model in SysML to its respective domain models can result in a complicated set of FMUs. Theoretically, representation of an ACPS model can be made with only two Models, where one CT model describes all of its physical side and one DE model describes the cyber side. However, there are several reasons why this is often not the case in ACPS projects. Different tools might be used to model different parts of the model, and models developed in separate tools must be exported as their own independent FMUs. The project could also be a collaborative effort where several parties are responsible for modelling different parts of the system, where each party would produce its own FMUs. Sometimes, several FMUs could also be created to better represent the ACPS architecture. The System Traceability View ensures that all the FMUs created to represent the ACPS model can be traced back to the SysML architecture model.

Viewpoint Context View

The Viewpoint Context View for the System Traceability Viewpoint is shown in Figure 5.41. The main Use Case is to 'Show traceability between architecture model and co-model' that includes 'System Elements and Models', and 'co-simulation, connection view, and scenario'.

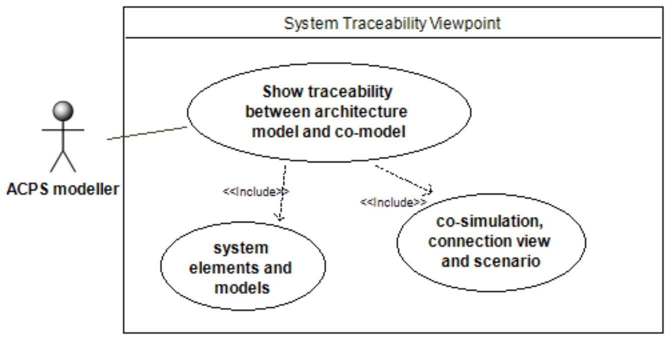

System Traceability Viewpoint
ACPS modeller
Show traceability between architecture model and co-model
<<include>>
<<include>>
system elements and models
co-simulation, connection view and scenario

Figure 5.41: Viewpoint Context View for the System traceability Viewpoint

Viewpoint Definition View

The Viewpoint Context View for the System traceability Viewpoint is shown in Figure 5.42. The diagram shows that 'System Traceability View' contains one or more 'System Elements', 'CT models', 'DE models', 'Scenarios', 'Co-simulations', and 'Connection Views'. One or more 'CT models' and 'DE models' can be traced to one or more 'System Elements', while one 'Connection View' is traceable to one or more 'Co-simulations', and one or more 'Co-simulations' are traceable to one or more 'Scenarios'.

View discussion

There are two main types of traceability shown here. The first is the traceability between the architecture of the Co-model to the architecture of the SysML model in terms of System Elements. Traceability between Models to System Elements can be quite complicated and cluttered compared to traceability between other model elements shown in the Traceability View in Need Perspective. This is because several Models might be created to represent a System Element, but a single Model might also represent several System Elements. As it is a many to many relationship, it could produce many traceability lines when a lot of System Ele-

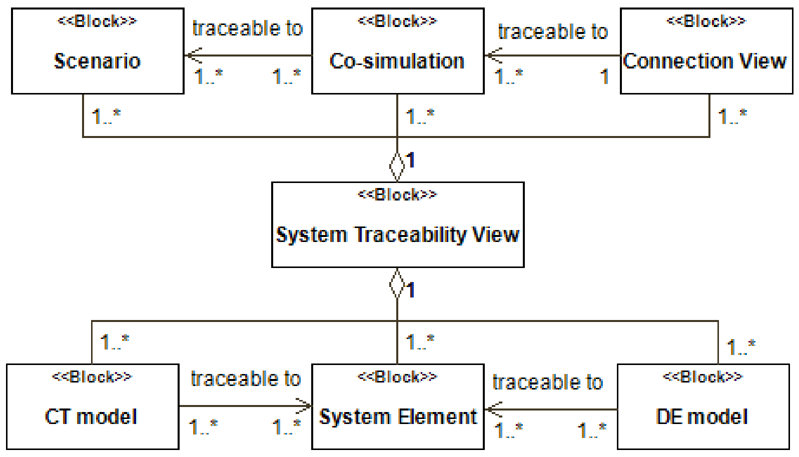


Figure 5.42: Viewpoint Definition View for the System traceability Viewpoint

ments and Models are involved. In that case, multiple System Traceability Views might be created for better visualisation.

The second traceability is between the Co-simulation and validation Scenario defined in the Validation Viewpoint. A Co-simulation might represent and satisfy several Scenarios, as some scenarios could only represent the behaviour of a single element. However, co-simulation could involve all the Elements in the model. Multiple co-simulations will also be needed to show the behaviour from one Scenario that involves branches of alternate behaviour (e.g., an if-else branch). Traceability from Connection View to Co-simulation simply shows which configuration from the Connection View is used for that Co-simulation.

5.11 Evaluation

In this section, we evaluate ACPSAF with reference to the evaluation method indicated in the Support Evaluation stage from DRM. We provide the Documentation of the Actual Support, which includes the evaluation of ACPSAF in comparison to the Intended Support Description, Intended Introduction Plan, and Intended Impact Model shown in Section 3.4.

Actual Impact Model

The Actual Impact Model will be presented as a list, where each element in the list is taken from the Intended Impact Model shown in Table 3.2. For each of the elements in the Impact Model, we will evaluate whether the element has its Need realised in ACPSAF. Note that the evaluation does not indicate whether the framework achieved what it is intended to do (this can only be achieved with a validation method). However, it evaluates whether the design process of ACPSAF has been carried out as planned.

- The Support should use the MBSE ontology for ACPS defined in Chapter 4 - [SATISFIED]
  The ontologies used in ACPSAF are taken strictly from the ACPS Ontology. This is explicitly shown in the Ontology Definition View (Section 5.3).
- The Support should be readily deployed with MBSE modelling language - [SATISFIED]
  ACPSAF is designed to support architecture modelling with SysML as the Viewpoints describe how the model can be created using elements from SysML.
- The Support should be focused on system architecture (as opposed to low-level implementation) - [SATISFIED]
  Each of the Perspectives in the ACPS is focused on the high-level architecture description on ACPS. Users can still describe low-level implementations such as system behaviours. However, specific implementation strategies are not included in ACPSAF.
- The support should be an Architectural Framework - [SATISFIED]
  ACPSAF is an architecture framework that complies with ISO/IEC 42010 and is defined with FAF (which guides the creation of an AF).
- The Support should provide framework description in MBSE modelling language - [SATISFIED]
  ACPSAF provides two descriptions using a combination of natural language and SysML. Both descriptions are meant to be used together.
- The Support should facilitate the MBSE life-cycle, from requirements to V&V - [SATISFIED]
  ACPSAF follows the simplified V-model life cycle where the life cycle is separated into three different stages. The first stage is concerned with the requirement acquisition and is mostly captured in the Need Perspective. The second stage is design engineering, which is captured in the CPS and Autonomy Perspective. The last stage is the V&V, captured in the Co-modelling Perspective and Need Perspective. This separates ACPSAF from other ACPS frameworks such as [71] as they do not have concern over certain life cycle phases such as requirement acquisition.
- The Support should incorporate separation of concerns between autonomy and non-autonomy architecture - [SATISFIED]
  ACPSAF separates these concerns into two Perspectives, where CPS Perspective captures all non-autonomy architecture and Autonomy Perspective captures the autonomy-related architecture.
- The Support should be able to identify autonomy characteristics in ACPS - [SATISFIED]
  ACPSAF allows the Autonomy Requirements to be identified and captured with the Requirement Description View in Section 5.7.2. The context in which these Autonomy Requirements are satisfied in the architecture of ACPS is captured within the Autonomy Context View in Section 5.9.2.
- The Support should be able to identify and capture Autonomy Requirements in ACPS - [SATISFIED]
  This is a subset of the previous requirement and is achieved with the Requirement Description View in Section 5.7.2.
- The Support should be able to express the architecture of ACPS - [SATISFIED]

ACPS captures the structural architecture of ACPS with CPS Perspective and uses both CPS Perspective and Autonomy Perspective to capture the behavioural architecture.

- The Support should provide a method to support the V&V of ACPS - [SATISFIED]
  ACPSAF provides verification and validation support through the use of Scenario-based co-modelling and co-simulation. The Validation View in Section 5.7.6 allows users to define both formal and semi-formal Scenarios that have to be validated against the ACPS model. Some SysML modelling tools allow test cases to be defined and tested in the tools for verification and validation. However, we focus on verifying and validating Scenarios against the simulation of the co-model. The verification has to be done manually as the user has to match the Scenario designed with the output of the co-simulation. A potential work for the future is to provide support for automation of verification.
- The Support should not be restricted to a limited type of autonomy - [SATISFIED]
  ACPSAF is not designed for any specific types of autonomy architecture. This design choice presents its own set of benefits and disadvantages. The benefit of this approach is that the framework is suitable as a starting point for any ACPS if required, and it can be extended to accommodate Needs that it currently does not support. One such example is self-organising behaviour where a collection of self-adaptive ACPSs interact with each other to achieve emergent behaviour. The Autonomy Requirements are now considered individually for each Controller. However, there might be a need to capture the Autonomy Requirement of self-organising globally as well.
- The Support should not be restricted to certain CPS domains - [SATISFIED]
  One example of domain-specific support is the work proposed by Rahatulain and Onori [70] targeted at manufacture CPSs. CPSAF uses ACPS Ontology that is not specific to any CPS domain.
- The Support should be usable by experienced, individual systems engineers - [SATISFIED]
  ACPSAF is designed following methodology that is common in MBSE, making it familiar to systems engineers.
- The Support should be defined with a framework - [SATISFIED]
  ACPSAF is defined with FAF.
- The Support should reuse components from existing framework where applicable - [SATISFIED]
  Components and concepts from ACRE, MBSEAF, and INTO-SysML are reused in ACPSAF. Further details are mentioned in Section 5.1.1.

Actual Introduction Plan The Actual Introduction Plan has not changed from the Intended Introduction Plan shown in Section 3.4. However, the Actual Introduction Plan is still defined and shown here for completion.

1. Target user: ACPS designer. *ACPS designer* is used loosely here to include anyone that has an interest to design, model, and engineer an ACPS. The term is not limited to a single individual and no level of qualification is required. However, understanding of modelling language is required (see below).

2. Required knowledge: SysML. ACPSAF guides the framework user to design an ACPS model with SysML as the modelling language of choice. ACPSAF assumes the user has knowledge of modelling with SysML as a prerequisite. The Viewpoints created in ACPSAF follow the convention of SysML 1.3. The latest version of SysML at the time of writing is SysML 1.5. Changes from SysML v1.3 to v1.5 are minor and do not affect the compatibility of ACPSAF. However, this is not guaranteed for future versions of SysML.
3. Required tools: Modelling tools for SysML. ACPSAF is designed to support the architecting of ACPS models in SysML. To use ACPSAF, users would need to have a modelling tool that allows SysML models to be created. In our work, SysML models and diagrams are created with Modelio [1] (open source version). The exact version is Modelio 3.7 with SysML Architect 3.7.02 module installed.
4. Additional tools: Co-modelling and co-simulation tools. ACPSAF supports the use of co-modelling and co-simulation as a method to support verification and validation. If a user is interested in this feature, co-modelling and co-simulation tools will be needed. A list of tools that support co-simulation through FMUs is recorded in the fmi-standard website[2]. The tools of choice used in our case studies are overture, 20-sim, and INTO-CPS.
5. Disclaimer: ACPSAF is the result of an effort to advance the state of the art of MBSE for ACPS. It has not been accredited for industrial use and should not be treated as one without further formal evaluation.

Actual Support Description

- **Problem addressed**

  The lack of an MBSE framework that supports the architecting of ACPS.
- **Goal and objectives**

  The goal is to create an AF that supports the architecting of ACPS through the MBSE life cycle, from requirements to V&V. This would include: Identification of Autonomy Requirement, architecture description to achieve Autonomy Requirement, traceability to co-model, and V&V scenario for co-simulation.
- **Elements**

  ACPSAF is made up of five Perspectives as listed below:

  1. AF Perspective - contain Viewpoints that describe ACPSAF itself
     - AF Context Viewpoint - define the Context for ACPSAF
     - Ontology Definition Viewpoint - define Ontology for ACPSAF
     - Viewpoint Relationships Viewpoint - show relationship between Viewpoints in ACPSAF
     - Rules Definition Viewpoint - define the Rules that constrain ACPSAF
  2. Need Perspective - contain Viewpoints for requirements acquisition
     - Source Element Viewpoint - show all the information needed to capture the Needs
     - Requirement Description Viewpoint - describe Needs using Need Description

[1]Modelio, https://www.modelio.org/, Last accessed: Nov 6, 2019
[2]Fmi-standard, https://fmi-standard.org/tools/, last accessed: Nov 6 , 2019

  - Definition Rule Set Viewpoint - capture Rules to constrain the Need Description
  - Context Definition Viewpoint - capture Contexts or point of views to be considered in the project
  - Requirement Context Viewpoint - capture Use Cases by considering Need Description into Context
  - Validation Viewpoint - capture Scenarios as a basis of validation
  - Traceability Viewpoint - capture traceability between Needs and other traceable elements

3. CPS Perspective - capture structure and non-autonomous behaviour of ACPS
   - System Identification Viewpoint - identify ACPS and the System Elements making up the ACPS from a hierarchical relationship
   - System Structure Viewpoint - show the structure and relationship between System Elements
   - System Interface Viewpoint - define Interfaces that will be used in the ACPS
   - System Configuration Viewpoint - show the structural configuration of the overall system
   - System Behaviour Viewpoint - define the behaviour of System Elements and System Functions
4. Autonomy - capture autonomy characteristics of ACPS
   - Autonomy Structure Viewpoint - show System Elements required to satisfy an Autonomy Requirement
   - Autonomy Context Viewpoint - show that Autonomy Requirements in the ACPS can be satisfied
   - Autonomy Behaviour Viewpoint - define the ACPS behaviour to fulfil the Autonomy Requirements
5. Co-modelling - for V&V with Co-modelling and Co-simulation approach
   - Architecture Structure Viewpoint - identify the Models making up the co-modelling in a hierarchical structure
   - Connection Viewpoint - shows the connection between each FMU for co-simulation
   - System Traceability Viewpoint- capture traceability between models and System Element, as well as co-simulation to Scenario

- **How it works**

  ACPSAF provides principles and practices for creating and using the architecture description of ACPSs. However, the process for using the framework is not provided. An ACPSAF user may decide on their chosen MBSE life cycle model, which could result in a different process when using the framework. For the V-model life cycle, the recommended process (in the intended order) is to complete the Need Perspective for the requirement stage, the CPS and Autonomy Perspective for the design stage, and the Co-modelling Perspective for the implementation of V&V. ACPSAF is not meant to be a rigid framework as it is intended to be adaptable. When required, users are encouraged to modify framework elements such as the ontology, stereotype, and tags in the stereotype. However, it is important to be cautious to not compromise the consistency and correctness of the model when making such changes.

6 Validation for ACPSAF

Evaluation in DRM is categorised into Application Evaluation and Success Evaluation. This chapter is concerned with the Application Evaluation of ACPSAF, which assesses the usability and applicability of ACPSAF through the use of case studies. Section 6.1 outlines a set of criteria that the case study must satisfy in order to evaluate ACPSAF. Section 6.2 presents a case study of a self-healing smart grid, and Section 6.3 presents a case study of Search and Rescue UAV, both designed with ACPSAF. Finally, Section 6.4 evaluates the extent to which the case study can be used to validate ACPSAF and provide the results of the Application Evaluation.

6.1 Criteria for Case Study

Application Evaluation and Success Evaluation rely on the case studies to be undertaken in this chapter. As a result, the case studies need to ensure they demonstrate aspects of ACPSAF sufficiently for these evaluations. As mentioned in Chapter 3, our research conducts the initial stage of Descriptive Stage-II, where findings from case studies should be treated as an initial evaluation and not as proof. However, the evaluation should provide useful insights into the problems and issues related to the proposed Support. Evaluation are done by comparing the result from the ACPSAF case study against its Actual Impact Model (Section 5.11) and Success Criteria (Section 3.3.2).

Based on the Actual Impact Model and Success Criteria, we define requirements that the case study must satisfy so that it can be evaluated. Section 3.3.2 shows that the criteria for success is 'Improved MBSE support for holistic ACPSs'. This is a high-level criterion that is difficult to measure, so a measurable criterion is defined as 'Integration of ACPS techniques with MBSE approach'. This is a fairly straightforward measurement as the case study simply needs to include external ACPS techniques from existing literature. The exact ACPS technique used for each case study will not be decided here but in the case study itself.

REQ1 Each case study should demonstrate the application of external ACPS techniques in conjunction with ACPSAF.

Actual Impact Model in Section 5.11 contains a few factors to be considered. To ensure that ACPSAF can be evaluated against these factors, the following requirements are proposed.

REQ2 The case study should strictly follow the ontology provided in ACPSAF without modification.

REQ3 The case study should be designed with SysML.

REQ4 The case study should be carried out through the MBSE life cycle, from requirements, design, verification, and validation.

REQ5 The case study should use all the provided Perspectives and Viewpoints.

REQ6 The case study should produce at least one View for each Viewpoint.

REQ7 The case study should show how Autonomy Requirement can be identified with ACPSAF.

REQ8 The case study should use the Co-modelling and Co-simulation approach proposed in ACPSAF to verify and validate ACPS.

REQ9 Two case studies should be carried out, demonstrating different types of autonomy.

REQ10 Two case studies should be carried out, demonstrating different domains of ACPS.

REQ11 The case study should be carried out by a systems engineer.

6.2 Case study: Self-Healing Smart Grid

This section is concerned with a case study based on the self-healing smart grid as a way to demonstrate and validate ACPSAF. The smart grid model for this case study will be built from the ground up to demonstrate the role of ACPSAF in the design and architecting of ACPS. The background of the relevant smart grid technology is covered in Section 6.2.1 to serve as a reference and starting point. Section 6.2.2 considers the requirements that the case study has to satisfy and clarifies assumptions to simplify the model. We will then present the case study following the V-model process, starting from requirement acquisition in Section 6.2.3, System design in Section 6.2.4, and V&V in Section 6.2.5. The case study requirements state that the case study should incorporate the external ACPS technique. This case study integrates a form of run-time monitoring framework for ACPS called the Dependability Cage in Section 6.2.6. A brief evaluation is carried out in 6.2.7 to compare the case study in relation to its requirements.

6.2.1 Background

This section describes the background to the self-healing smart grid system. As the case study will be modelled from the ground up, we cover the relevant technology regarding power grids, smart grids, and eventually the self-healing mechanism for smart grids.

6.2.1.1 Power Grid

In general, the electric system is an interconnection of generation, transportation, and consumption of electricity. The large electric system in the real-life scale is called a power grid. Also known as an electrical grid, the power grid is a network of high tension cables delivering electricity from the power plant to customers. The network of electric power distribution can be abstracted into power generation at power stations, transmission lines, and distribution lines. Figure 6.1 illustrates the basic structure making up a typical power grid system.

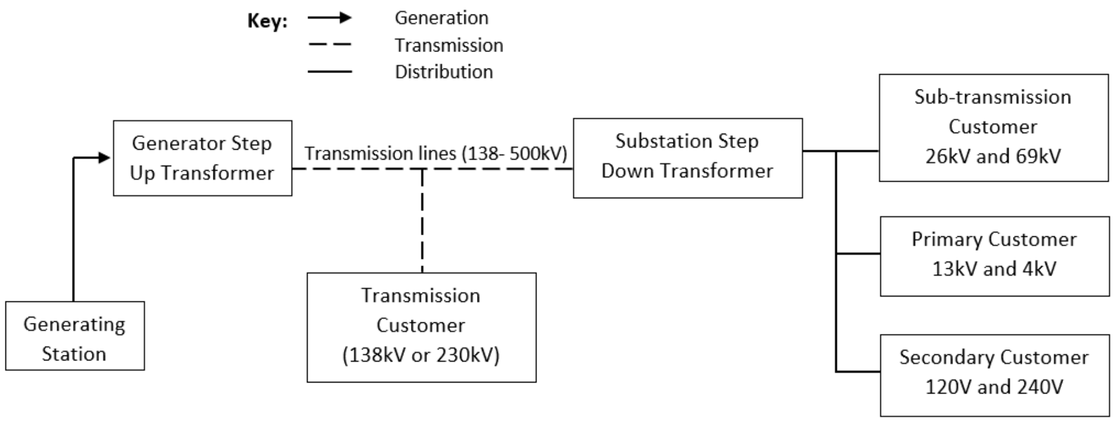

Key:
Generation
Transmission
Distribution
Generator Step Up Transformer
Transmission lines (138- 500kV)
Substation Step Down Transformer
Sub-transmission Customer 26kV and 69kV
Primary Customer 13kV and 4kV
Secondary Customer 120V and 240V
Transmission Customer (138kV or 230kV)
Generating Station

Figure 6.1: Basic structure of the Power Grid system

Power Station and Generation

A power station is an industry service that generates electric power for the purpose of electric power distribution. This is the place where electricity is generated with specific types of generators and fuels. Although there is more than one way to generate electricity, the most common method is electromagnetic induction based on Faraday's law; transforming kinetic energy into electricity by rotating a magnet within closed loops of a conductor. There are many types of power stations, and each can be classified differently according to its fuel type, functionality, and type of current.

The fuel type used by power stations can be classified as non-thermal or thermal. Examples of non-thermal are hydroelectric, windmills, and tidal power. Examples of thermal include steam power, gas turbine, and nuclear power. A non-thermal power source (also known as renewable energy), such as a windmill, is usually used to rotate the turbine of the generator directly. In a thermal type, fuel is usually used to generate heat, which heats water to create steam that will be used to turn the turbine.

Energy demands fluctuate greatly throughout the day, putting the grid under varying loads that it has to satisfy. An efficient energy generation system should produce sufficient energy to meet high energy demands without overproducing during low demand. This is an issue as a single power plant typically does not have the flexibility to meet such a requirement reliably. To address this challenge, a power plant can be classified as a base load power plant, a load following power plant, or a peaking power plant. A base load power plant operates continuously and provides the minimum electricity demand, while the peaking power plant operates when there is a peak demand for electricity. A following power plant operates in between the extremes of the base and peaking power plant (the so-called intermediate load).

The majority of the existing power stations mostly produce AC instead of DC, except for HVDC (High Voltage DC). The main benefit that AC has over DC in a power grid system is that AC voltages can be readily transformed to higher or lower voltage levels, which is difficult to achieve with DC voltages. Although AC is by far the most popular and common type chosen by power industries, the nature of AC imposes some drawbacks for industrial use. As the name suggests, AC changes its polarity over time, producing the shape of a sine wave. This property means that AC power cannot provide constant, peak power delivery. While this might not be an issue for residential use, commercial and industrial customers require a constant power delivery rate. For this reason, power stations generate three phase AC instead of the normal single phase AC. The three phase AC is basically a product of three concurrent, distinct AC phases with an offset of 120 degree from each phase. It is of course possible to have two phase or four phase. However, having three phase is the most economical and efficient solution compared to the others. A properly arranged offset ensures that there is always a phase that is reaching its peak.

Since the power grid transmission uses a three phase AC system, it is understandable to see a transmission tower supporting 3 wires for each phase; but in reality, there are more than 3 cables. This is because the transmission tower can also carry a neutral wire, ground wire, and phone wire. A neutral wire is used to balance the electrical circuit by carrying any excess current, while a ground wire is used to allow a path for the electricity to flow from the tower to the earth. The latter is extremely useful when the tower is electrically charged from a power leakage or lightning strike.

Special attention is required when measuring the voltage level generated by an AC power system. Unlike DC where the flow of the electric charge is constantly in one direction, the flow of AC's periodically reversing electric charge means that the average voltage output over time needs to be considered within the voltage level calculation. Assuming the source is a pure sine wave, this can be achieved by calculating the Root Mean Square (RMS) of the voltage waveform, which is defined as

$$V_{RMS} = \frac{V_p}{\sqrt{2}}$$

where V_{RMS} is the RMS voltage and V_p represents the peak voltage. This would mean that for AC graphs where the peak voltage is 120V, the RMS voltage would be around 85V (see Figure 6.2). Note that the voltage value stated in any electronic device or generator typically means the RMS value and not the peak value unless specifically stated. Putting the voltage aside, the frequency of the power needs to be regulated as well. As explained earlier, generators generate electricity by rotating a copper wire around magnets to produce a current by means of electromagnetic induction. When the copper wire finishes rotating all the way through its axis (3600) until it reaches its original position again, that is called a cycle. The frequency in terms of a generator is the number of times the copper wire within that generator can complete a full cycle in a second. Therefore, 60Hz would mean that the copper wire completed 60 cycles of rotation in a second. The standard for frequency level in a power grid can vary between countries but should be respected to ensure that the electronic devices connected to the grid match with the frequency generated. Typical power line frequency is 50Hz or 60Hz, with the UK currently being standardised at 50Hz and 230V transmission for residential houses.

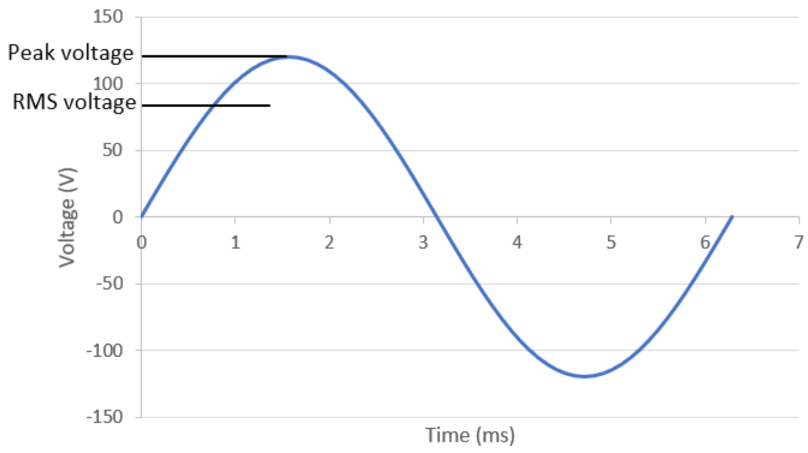


Figure 6.2: RMS and peak voltage in AC system

Transmission Lines

The transmission line is used to transmit the electrical power generated from the generating substation to the various distributing units. Electricity is almost exclusively transmitted using overhead lines and via underground cables only in special cases. The overhead lines are made of pylons that carry the circuit conductors and ground conductor at the top of the structure. The voltage levels during transmission are typically maintained at a very high level for several reasons. Transmitting at high voltages means that power is transmitted at a lower current that ultimately leads to lower resistance losses in the conductors. On top of that, lower current means that thin, light-weight wires can be used in long-distance transmission. As a result, transmission towers do not need to be engineered to support the weight of heavier wires that would be associated with a high current.

Transmission lines can be considered the most complex part of the power grid because transmission networks are often interconnected with other transmission networks to create better redundancy. Because of the length of wiring in the transmission scheme, this is where the efficiencies and faults can affect the power system the most. However, our case study would consider a relatively simple transmission line architecture, so more complex transmission networks are beyond the scope of our research.

Distribution lines

Distribution lines are the final phase in delivering the electricity to customers. Customers can be categorised depending on their power demands (see Figure 6.1). In general, power grid customers are categorised into transmission customers, sub-transmission customers, primary customers, and secondary customers. A transmission customer is any customer that is qualified to carry out a transmission service or is on the receiver end of the transmission itself. This customer is relatively different from other customers because of their demand for high voltage transmission power. There is no voltage cut off from the transmission line (after the step-up) to the transmission customer, so the voltage level is around 138kV or 230kV. The most common transmission customer is the Purchasing-Selling Entity.

Except for the transmission customer (who can be categorised as part of the transmission lines), the power will go to a substation where a step-down transformer will decrease the voltage level to a suitable voltage level for the customer. The sub-transmission customer is not a power consumer that stands at the end of power transmission itself, but rather, a part of the transmission system too. Sub-transmission takes a relatively high voltage level to be transmitted into one or more substations. This is useful when it is practically uneconomical to have a direct connection from the main transmission into all the distribution substations.

Primary customers are customers that require a power transmission high enough to power an industry with lots of heavy machinery. The voltage lever for this customer can vary depending on the need, but it is generally at 4kv or 13kv. The secondary customer is a customer that requires power transmission to power up general purpose appliances. Residential houses and buildings are the major customers in this category, with the transmission voltage of 120v or 240v. However, most households have single-phase loads while the transmission is in a three-phase. Therefore, only 1 of the 3 power cables is connected to each house.

Control centre

The control centre is an integral part of a maintainable electric grid system. It checks the status of the power system, adjusts its condition, and provides a defence against unwanted events. The primary task of the control centre is to provide a reliable power system from generation until consumption, doing so with utmost efficiency in terms of energy utilisation. Electricity moves at nearly the speed of light, which means that the generated energy must be delivered and consumed within that instant. Although there are many ways to detect the balance between the amount of power generated and consumed, such as using sensors or smart meters which will be covered in detail in the next section, a physical effect caused by this difference of generation and load can be observed by the frequency of the power plant generators.

In a stable power system environment where the generation is equal to the load and loss, the generator will be operating under an optimum condition at normal frequency. However, when the load is greater than the generation, it will put a stress onto the generator and consequently slow down the frequency of the generator and vice versa. One way to illustrate this is to imagine that the frequency of a generator with a certain level of power is like the speed of the wheel of a car with a certain level of force applied to it. When there is more load applied to the generator, the frequency slows down, similar to if a car were to climb up a steep hill. The car would slow down because it needs more force. If the reverse were to happen, where the load is much smaller than the power of the generator, the frequency will increase as if the car was now driving downhill.

Depending on the type of the generator, different inertia constants and power-factor ratings will produce different amounts of relationship between frequency and the load reduction; however, the frequency ratio between the frequency drop and the load reduction is 1:2. This means that for every drop percentage in the frequency, the load loss is double the percentage of that. For example, a generator rotating at 50Hz and producing 100KW is at a state where the generated power is equal to the sum of load and energy loss. If the frequency has dropped to 45Hz, we can calculate that the generator has a 5Hz frequency drop, which is a 10% drop. This means that the load reduction is 20% of 100KW, which is 20KW. Therefore, the generator is essentially rotating at 45Hz and producing 80KW to supply an original load of 100KW in this situation. However, such a big percentage of frequency deviation will most likely cause the generator to fail. The regulation for the frequency control usually limits the nominal of up to $\pm 1\%$ deviation to be considered within normal situations.

Having the frequency response in check does not equate to the absence of fault in the power system. For example, consider a power grid that provides electricity from the power plant to two cities, city A and city B, and both cities are consuming the same exact amount of power. If city A doubles its power consumption and city B stops consuming power simultaneously, there would not be any anomaly in the frequency analysis because the total load is still the same. However, this situation will most likely lead to the overloading of other elements in the power system.

Each element in the power grid has its own capacity of how much current it can carry. When it is forced to carry a big amount of current beyond its capacity (usually the transmission cable), it could become overloaded. Overloaded wires could trip the breaker in the transmission tower, or they might sag until they hit a tree and cause a short circuit that will eventually damage the electrical equipment, causing a power failure. This broken circuit will then have its load shifted to another nearby interconnected circuit. Such a big load would usually cause the second circuit to be overloaded as well, and this could continue tripping another circuit. If not mitigated immediately in an appropriate manner, this would then affect the whole power system.

This is called a cascading failure.

Most of the time, the control centre has to take appropriate actions when a fault occurs in the power system. Some faults can be mitigated quickly into smaller failures before it spreads into a major power failure. This action of mitigating power failure can be divided into different categories of power outages, such as Dropout, Brownout, Blackout, and Load shedding. The goal of the self-healing grid is to detect and contain blackouts in a smaller area and redirect the grid such that other parts of the grid would only experience a dropout.

Category	Description
Dropout	A power outage that occurs only for a matter of seconds. This is a type of transient fault, in which the fault occurring can be fixed by disconnecting the power momentarily and then reconnect it again; such faults involve lightning strikes and momentary tree contact.
Brownout	Instead of a complete power outage, brownout is experienced as a drop of voltage in the electrical power supply or temporary reduction in electrical power. This type of fault is usually caused by a power plant's inability to generate enough power to supply the load. This can also be caused by a fault in one of the phase wires in the three phase wire. Brownout can be observed from the dimming of light and may cause poor performance or even damage to electrical appliances.
Blackout	This fault refers to the complete absence of any electrical power for a duration of time that could last from hours to days or even weeks. Blackouts are the result of power stations tripping and being unable to recover promptly.
Load shedding	Also called a rolling blackout, load shedding is when the electric company purposely shuts down a power transmission to a specific area on the grid. This is usually done because the power plant is unable to generate electricity to supply the whole load. Therefore, some areas have to be shut down to prevent a total blackout.

Table 6.1: Types of power outage

6.2.1.2 Smart Grid

The power grid is a complex infrastructure of wires, towers, transformers, and more. Although the system is designed to deliver power to every customer reliably, disturbances such as natural disasters, vandalism, or even unexpected load demands could lead to power failure. The power grid system also requires utility companies to send their employees into the field to gather data needed to run the power system. Workers have to visit houses and buildings to read meters, patrol for impaired or broken equipment, and take other measurements. Combined with the increasing demand for energy, the current power grid system is unable to catch up with the energy demand.

Since around the 1960s, power industries have started the use of computers to observe and extend the activity of the power system. Such a centralised control needs multiple, high-speed two-way communication. However, even at its best, the current system still requires a 20-second delay before the central system is updated of the power system's condition [355]. It might seem like a short delay, but considering that the electricity is moving at almost the speed of light, some failures leading to cascading failures could trip the power system beyond control within that delay. The current power grid is also inefficient in terms of managing transmission efficiency and load management.

The effort to modernise the power grid with the application of sensors, communications, computational ability, and control to improve the old electric system is the smart grid. With such a general definition of smart grid, it is not surprising that there is more than one system which defines a smart grid [356]. In general, there are three main activities towards working on the future smart grids that address current power grid challenges:

- **Data gathering**. Sensors will be placed throughout the grid to measure the activities and conditions of the grid. This will be used to provide remote, real time monitoring of the grid condition. Smart grid sensors can detect power flow, such as voltage and current, together with weather elements like temperature and wind flow. Data can also be gathered from smart meters that will be installed in each house to monitor their power usage.
- **Data Analysis/forecasting**: The gathered data will be analysed to provide useful insight to benefit the customers and the utility company. The customer can benefit from smart meters, ensuring a more accurate billing and being a participant of a demand response, while the utility company can benefit from the sensors helping them to avoid failures, allowing them to take better care of their electrical equipment and reduce costs by reducing transmission power loss.
- **Monitoring and response**: The data analysed will trigger a process from the utilities side to take action that will improve overall performance, reliability, and efficiency. Current utility companies use a type of industrial control system to process the gathered information and inform the operator at the control centre of any anomaly that needs attention, which is usually SCADA [357]. However, the response of a smart grid is not limited to only a supervised response. The smart grid can be structured such that when a sensor picks up a damaged element, the grid itself can automatically react to heal or prevent further failure. This introduces a self-healing smart grid.

As explained above, there are many possible advancements offered by smart grids, such as the utilisation of renewable energy. The majority of improvements offered by smart grids are usually focused on their economic value, environmental impact, and reliability. The key element for a smart grid is the deployment of smart grid sensors and smart meters.

As the name implies, the basic functionality of a smart grid sensor is to ‘sense’ the property and condition of an element in the power grid and the environment surrounding it, sending the monitored data to a control centre. A typical application of a smart grid sensor is one that detects the current flowing through the grid. Not limited to measuring electricity flowing in the grid, sensors can be extended to detect ambient weather conditions such as temperature, wind speed, and solar radiation.

A smart meter can be installed for each customer, typically residential customers. This smart meter is responsible for gathering an accurate reading of the customer's power usage to be transmitted to the control centre, enabling better power flow control and accurate billing. Combined with a demand response facility, it opens the opportunity for bidirectional energy flow and effective utilisation of renewable energy. Real time energy consumption monitoring also enables energy suppliers to offer time-variable pricing [358]. For example, time-of-use rates, where prices move at set times and amounts throughout the day, or real time pricing where customers will be charged at or close to wholesale price.

6.2.1.3 Self-Healing Grid

The recovery process from an electrical grid failure involves an elaborate set of activities. Once the control centre has been notified of the failure, workers have to be dispatched to investigate the exact location of the fault. Depending on the architecture and location of the fault, this can be a time-consuming process of scouring through the conductors in the faulty area. Once the location of the fault is identified, the faulty circuit has to be isolated to ensure that there is no electricity transmitted to the conductor during the repair. After the fault is fixed, the power grid can continue its regular service.

Despite its name, self-healing grid does not refer to a power grid that is able to fully 'heal' from its faults without external (i.e., human) intervention. However, it does use smart grid technology to aid the recovery process of a faulty grid. A self-healing smart grid is characterised by its ability to:

1. Detect anomalies in the power flow of the grid and pinpoint the fault location to a reasonably small area by calibrating the data from the sensors distributed in the grid.
2. Isolate the faulty grid by controlling the switches connected around the faulty area.
3. Reconfigure its grid network to maximise energy delivery to customers outside the isolated areas.

To illustrate the self-healing grid mechanism, we consider a smart grid as shown in Figure 6.3. The simple smart grid above shows that eleven buildings are being powered by two generators. *Generator1* powers up *house1* to *house6* plus a *hospital*, and the power flow is marked by the yellow line; meanwhile, *Generator2* powers *house7* to *house10* and is indicated by the green line. Between those power flows are black dots that represent a transmission pole along the grid and three switches that can be controlled to connect and disconnect the electric flow. The switch also serves as a smart grid sensor that is able to take an electricity reading of the power transmitted to it.

We can now consider a fault where conductors on the transmission pole directly in front of *House2* are disconnected (see Figure 6.4). The two closest sensors (switches) will notice an anomaly in the power flow. *Switch1* would detect some fluctuation in the electricity flowing through it, and *Switch2* would detect that there is no electricity flowing through it anymore. This information is then transmitted to the control centre, indicating that there is a fault located in the area between *Switch1* and *Switch2*.

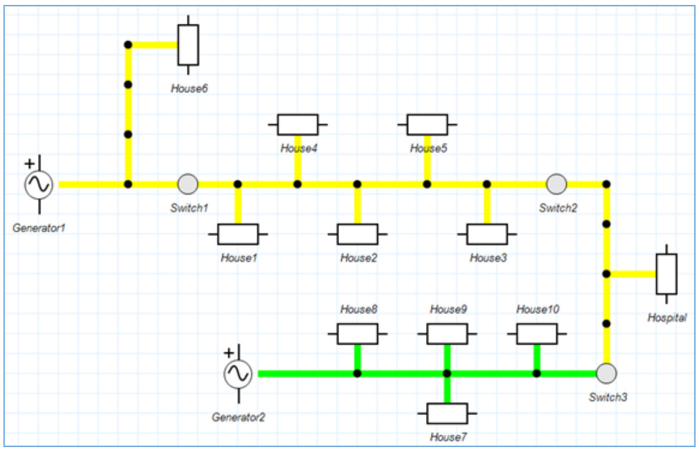


Figure 6.3: Smart grid sample with loads, generators, and switches.

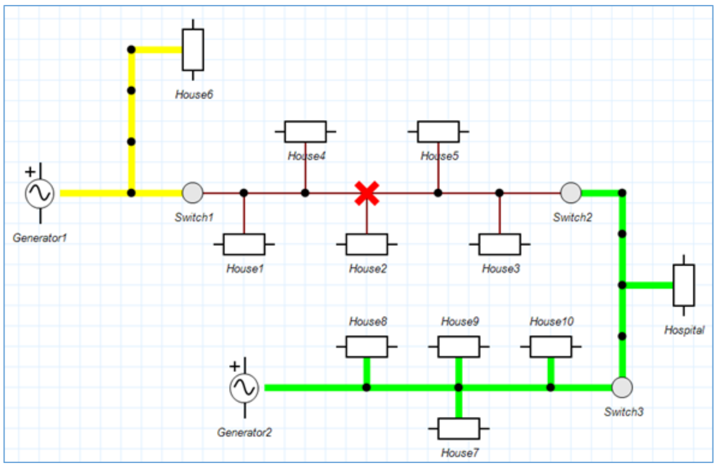


Figure 6.4: Smart grid sample with faulty grid.

With the information from the sensors, the control centre could disconnect *Switch1* and *Switch2* to isolate the faulty area. Furthermore, *Switch3* would be controlled to allow electricity to pass through it to the hospital. The end result is that *house6* is still powered up by *generator1* while *house1* to *house5* are now isolated and an engineer will be sent to fix the fault. *House7* to *house10* are still powered by *generator2*, and the *hospital* has now switched to consume power from *generator2*. With a smart grid, all of these processes can be carried out autonomously, and there would only be a power cut for a split of a second. This technology is especially beneficial for facilities in which the reliability of power delivery is critical, such as in hospitals.

6.2.2 Requirement and Assumption

In this section, we present the requirements for the self-healing smart grid case study. The scenario for the case study should be simple enough to minimise the resources needed while demonstrating key aspects of ACPSAF. For this reason, the case study demonstrated will be much more simplified compared to its real-life application. Assumptions are made to simplify the model and are elaborated in this section as well.

6.2.2.1 Project Requirement

Project requirement refers to the requirements that are related to the quality of the case study and not directly to the property of the ACPS in the case study itself.

- The case study should be engineered using the MBSE technology mentioned in Chapter 3. This includes the ACPSAF for Architecture Framework, SysML for modelling language, and INTO-CPS tool stack (i.e., Modelio, 20-sim, Overture) for modelling tools. Note that we will be using the open source version of Modelio, which does not support the use of requirements diagrams. This means that although some views are best captured with a requirements diagram (e.g., traceability view, requirement description view), we could only use alternative diagrams such as BDD.
- The case study should produce at least one view (SysML diagram) for each viewpoint in ACPSAF.
- The case study should demonstrate how the smart grid model developed with ACPSAF can be integrated with verification and validation techniques for ACPS. For this case study, we consider the integration of a run-time monitoring framework for dependable autonomy behaviour.
- The case study should evaluate the extent to which the autonomy-explicit approach in ACPSAF aids the design and architecting of the smart grid system.

6.2.2.2 System Requirement

System requirements refer to the requirements that are directly related to the property of the ACPS that is intentionally designed for the case study. Note that this is only one source of the requirements for the case study as the final requirements are derived through the Requirement Engineering process with ACPSAF.

1. The system shall be a smart grid that consists of power generation, transmission line(s), and distribution line(s).
2. Smart grid must contain loads that are connected to the transmission line and distribution line.
3. Smart grid shall contain a high-priority customer (e.g., hospital) located in the distribution line.
4. Smart grid must contain at least two sensors and switches distributed in the grid.
5. Smart grid must have a centralised control centre that monitors and detects faults in the grid in real time through the sensors.
6. Control centre shall have control over the switches to direct the electricity flow through the grid.
7. Smart grid shall control relevant switches to isolate the faulty grid.
8. Smart grid shall autonomously reconfigure its network of grid connections to maximise energy transmission to customers even in the presence of faults.
9. High-priority customers shall be given higher priority compared to other customers when a trade-off has to be made.

6.2.2.3 Simplification

There are certain complexities in the design of the system that will not make a substantial difference (or at all) to the autonomy in the smart grid. Therefore, several decisions were made on design choices that may deviate slightly from the typical real world design in favour of reduction in design complexity as it is safe to do so for the purpose of this case study.

1. Single-phase system. The power grid typically uses AC with a three-phase electric power. However, our model shall use AC with single-phase power instead. The three-phase system provides advantages that are not relevant in the demonstration of self-healing capabilities in a smart grid. Therefore, a single-phase system is chosen as it is simpler to model.
2. Single-circuit transmission line. Transmission lines are typically configured as either single or double circuits. Double circuits are used where greater reliability is needed, and it also enables the transfer of more power over a particular distance. Similar to the previous point, this advantage is not relevant for our case study, so a single-circuit is chosen for simplification.
3. Overhead transmission lines. Although it will not be modelled explicitly, we will assume that the transmission system is the overhead transmission lines. This will affect the types of faults for the self-healing scenario as we will consider faults that are common to overhead transmission lines.
4. Static load and generation. In real-life applications, the amount of load in the grid changes dynamically over time based on customer consumption. To simplify the model, we assume that each customer is drawing power from the grid at a static rate. This would also mean that the power will be generated at a static rate as well.
5. No energy loss. We simplify the model of the conductors such that energy can be transferred from one point to the other without any energy loss.
6. Naive power rate. Although we will try to match the voltage level as closely to those in the real-life application, the power generated and consumed is rather arbitrary. There will be some logic behind the power rate, such as the hospital consuming more electricity than a residential house, but the rate of the power consumption will not reflect those in a real application.
7. No ground wire. A grounded wire is sometimes strung along the tops of the towers to provide lightning protection. This wire will be omitted from the model.
8. Simplified generator and transformer. A power plant goes through a complex process to generate and transmit electricity. Although we call it a generator, there will not be any power generation process (e.g., burning fuels) in the model. We are not interested in this complexity, so it will be reduced to a static power source resembling a battery instead.
9. Sensor communication. We assume that the sensors can communicate to the control centre without explicitly specifying the communication architecture. Communication can be assumed to be made through a wireless connection over the internet which would happen almost instantaneously. This is a common

assumption in simulation models. However, this means that we would not be able to evaluate the real time response time of the self-healing mechanism. Technique [46] to bridge this gap will be considered for future work.

6.2.3 Requirement Acquisition

The case study will be presented based on the V-model process, albeit quite loosely. The main process would still be the requirement, design, implementation, and verification. However, there is no explicit method to assess when the project is ready to move from one phase to the other. This section deals with the first phase of the V-model, capturing the system requirements. This activity is different from the one presented in Section 6.2.2 as requirements in MBSE are captured into the model as well. The viewpoints concerned with the requirements engineering in ACPSAF are encapsulated into the Need Perspective. This section will utilise the Need Perspective to capture the self-healing smart grid requirements as an MBSE model.

Source Element View

The first step to model requirements with the ACPSAF is to capture all the information sources that are relevant to the system. Any type of documents, recordings, and even emails should be gathered if it is relevant to the system of interest. We model each of the sources as SysML blocks with the <<Source Element>> stereotype. The Source Element View for this case study is shown in Figure 6.5.

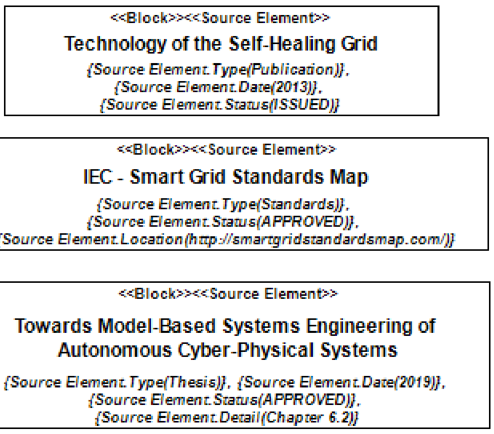

<<Block>><<Source Element>>
Technology of the Self-Healing Grid
{Source Element.Type(Publication)},
{Source Element.Date(2013)},
{Source Element.Status(ISSUED)}
<<Block>><<Source Element>>
IEC - Smart Grid Standards Map
{Source Element.Type(Standards)},
{Source Element.Status(APPROVED)},
{Source Element.Location(http://smartgridstandardsmap.com/)}
<<Block>><<Source Element>>
Towards Model-Based Systems Engineering of Autonomous Cyber-Physical Systems
{Source Element.Type(Thesis)}, {Source Element.Date(2019)},
{Source Element.Status(APPROVED)},
{Source Element.Detail(Chapter 6.2)}

Figure 6.5: Smart grid case study: Source Element View

Several tags are applied to the Source Elements. The *Type* captures the document types of the Source Elements. *Status* represents the acceptance of the proposed Source Elements, where *ISSUED* would mean that it is under consideration and *APPROVED* means that the Source Element is approved and should be considered into the requirement. The tag *Location* indicates where this document can be accessed, and *detail* allows further abstract information to be recorded.

Figure 6.5 shows that we have three Source Elements in our Source Element View, with different *Type* and *Status*. One of the Source Elements refers to this thesis, specifically at Chapter 6.2 that refers to this case study. The background literature, requirements, and assumptions covered in the earlier sections contain most of the relevant information for this case study.

Definition Rule Set View

The Definition Rule Set View defines Rules that must be followed by each Need Description. Rules for each project will differ depending on the organisation, team composition, culture, and preferences. There is no standard for a set of Rules that will work for everyone. ACPSAF users have to establish Rules by creating new Rules or reuse Rules from previous projects. When the project involves more people (i.e., stakeholders, engineers), the Rules tend to be stricter as they are a form of quality control. Good Rules help ensure that the Need Descriptions are not ambiguous and a common understanding can be achieved. For the purpose of this case study, the Rules we need are very lenient. Figure 6.6 shows that there are only two Rules for this case study. It states that each Requirement must have a unique identifier and must be described within a sentence.

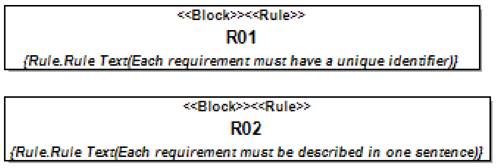

<<Block>><<Rule>>
R01
{Rule.Rule Text(Each requirement must have a unique identifier)}
<<Block>><<Rule>>
R02
{Rule.Rule Text(Each requirement must be described in one sentence)}

Figure 6.6: Smart grid case study: Definition Rule Set View

Requirement Description View

Based on the Source Elements and Rules, we can now model the requirements in the form of Need Descriptions. The Need Descriptions are realised here as a SysML Block, identified with the <<requirement>> stereotype (see Figure 6.7). As specified by the Rules, each requirement is assigned with an id, and the description of each rule is restricted to a sentence. Typically, an ACPS project would have more Need Descriptions than the one shown here. However, this set of requirements is sufficient for this case study.

Figure 6.7 shows that the *Self-Healing* requirement can be decomposed into several smaller requirements. To fully achieve self-healing, the smart grid has to be able to identify faults, isolate them, optimise power distribution through reconfiguration, and eventually reset to its initial state when the faults are fixed. Note that the requirement *Optimisation* is stereotyped with <<autonomous>>. This indicates that the requirement is in fact an Autonomy Requirement.

An Autonomy Requirement in ACPS can be identified through the following characteristics:

- The requirement involves the use of sensor(s) to acquire information regarding the environment.

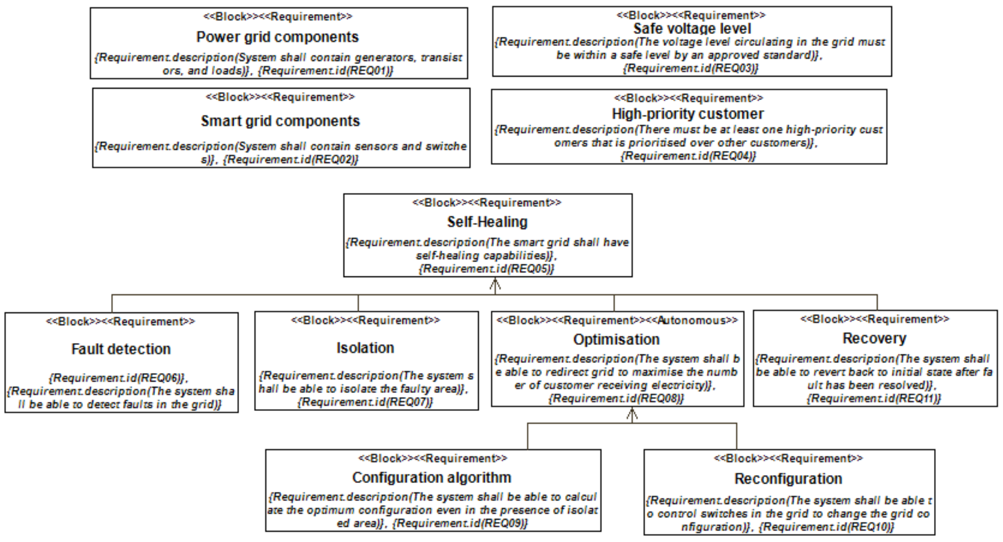

<<Block>><<Requirement>>
Power grid components
{Requirement.description(System shall contain generators, transist ors, and loads)}, {Requirement.id(REQ01)}
<<Block>><<Requirement>>
Safe voltage level
{Requirement.description(The voltage level circulating in the grid must be within a safe level by an approved standard)}, {Requirement.id(REQ03)}
<<Block>><<Requirement>>
Smart grid components
{Requirement.description(System shall contain sensors and switche s)}, {Requirement.id(REQ02)}
<<Block>><<Requirement>>
High-priority customer
{Requirement.description(There must be at least one high-priority cust omers that is prioritised over other customers)}, {Requirement.id(REQ04)}
<<Block>><<Requirement>>
Self-Healing
{Requirement.description(The smart grid shall have self-healing capabilities)}, {Requirement.id(REQ05)}
<<Block>><<Requirement>>
Fault detection
{Requirement.id(REQ06)}, {Requirement.description(The system sha ll be able to detect faults in the grid)}
<<Block>><<Requirement>>
Isolation
{Requirement.description(The system s hall be able to isolate the faulty area)}, {Requirement.id(REQ07)}
<<Block>><<Requirement>><<Autonomous>>
Optimisation
{Requirement.description(The system shall b e able to redirect grid to maximise the numb er of customer receiving electricity)}, {Requirement.id(REQ08)}
<<Block>><<Requirement>>
Recovery
{Requirement.description(The system shall be able to revert back to initial state after fa ult has been resolved)}, {Requirement.id(REQ11)}
<<Block>><<Requirement>>
Configuration algorithm
{Requirement.description(The system shall be able to calcul ate the optimum configuration even in the presence of isolat ed area)}, {Requirement.id(REQ09)}
<<Block>><<Requirement>>
Reconfiguration
{Requirement.description(The system shall be able t o control switches in the grid to change the grid co nfiguration)}, {Requirement.id(REQ10)}

Figure 6.7: Smart grid case study: RDV for high-level Need Descriptions

- The requirement involves the use of actuator(s) to influence the environment.
- The requirement is a *high-level objective* (see Chapter 4 for definition).

The *Isolation* requirement is not considered as an Autonomy Requirement as it does not fulfil all three characteristics of an Autonomy Requirement. When compared against the characteristics for Autonomy Requirements, it only matches with the first two points: It requires the sensors to detect faults in the grid, and switches as the actuator to influence the environment. Note that Environment here is mentioned in the context of the control centre. Anything that is outside the control centre is considered as Environment, and this would include other System Elements in the smart grid as well, such as the conductor.

However, the isolation mechanism is not a “high-level” objective as it is not an “indirect” objective. This is because there is a direct mapping between sensor reading to the state of the actuator switches, which can be easily satisfied with an IF-ELSE approach (i.e., as a teleo-reactive program). Therefore, there is no reason to consider the *Isolation* as autonomy as it would encourage a design that is more complex than is needed. For this case study, the isolation mechanism would be reactive rather than adaptive. An adaptive isolation mechanism would monitor the state of the energy flowing through the grid and change the state of the switches to stop the electricity from reaching the isolated area. Our approach would be a reactive mechanism where the switches automatically disconnect the grid when it detects disturbances in the power flow. When multiple switches disconnect the grid, it effectively isolates an area without being aware of it.

The *Optimisation* requirement is similar to the *Isolation* requirement in terms of its interaction with the environment. The requirement can only be satisfied by detecting a fault in the grid and controlling the switches to influence the environment. However, the goal is different as it tries to influence the environment in such a way that maximises the number of customers receiving electricity. This is a high-level requirement as the control centre has choices over multiple different grid configurations and has to adapt to the changes in the grid due to faults to maintain maximum energy distribution.

For the size of the smart grid that we will be modelling in this case study, it would be more practical to develop a teleo-reactive program with a simple IF-ELSE approach to cover all possible scenarios due to the low number of components involved. However, we decided to design the *Optimisation* requirement as an Autonomy Requirement (for the sake of ACPSAF validation). This means that the developed intelligent controller should be able to deal with any type of topology, even dynamically changing topology at run time, as long as it has a method to register the topology and its list of components. This is opposed to the teleo-reactive approach where changes in the topology or components would require logic re-programming and grows exponentially in difficulty as the number of components grows.

Context Definition View

The Context Definition View identifies the Contexts that will be explored in the Requirement Context View. These Contexts may take many forms, such as the Stakeholder Roles and System Elements as suggested in ACPSAF. For this case study, the two mentioned contexts are sufficient to capture all the Requirements into Use Cases in the Requirement Context View. The Context Definition View for Stakeholder Roles is presented in Figure 6.8. It shows that there are three types of *Customers*; the *Residential*, *Commercial*, and *Hospital* customer. There are also two types of *Suppliers*, the *Control Centre* and *Generator*.

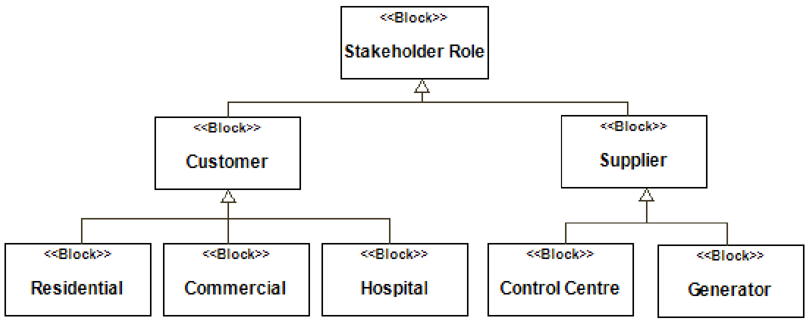


Figure 6.8: Smart grid case study: CDV for Stakeholder Context

The Context Definition View for the System Element Context is presented in Figure 6.9. At the top, all the System Elements are under the *Self-Healing Grid* block, which is an ACPS. At this point, it is not necessary to decompose the system into the lowest level of System Elements composition. The System Elements presented here are typically at the sub-system level. For example, the *Grid System* would be composed of the conductors, transformer, and the load. However, the Context Definition View only needs to show blocks that would be explored further in the Requirement Context View.

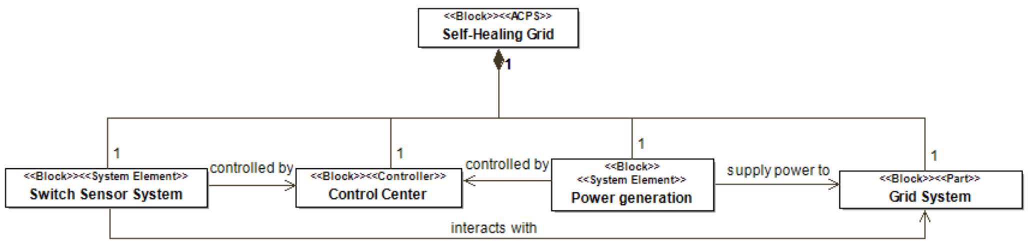


Figure 6.9: Smart grid case study: CDV for System Element Context

Requirement Context View

The Requirement Context View is used to generate Use Cases by considering Needs from a particular Context. One way to approach the design of the Requirement Context View is to start from each Context described in the Context Definition View. We will start from the Context of *Self-Healing Grid* and attempt to correspond a Use Case from each of the Need Descriptions (from *REQ01* to *REQ11*) listed in the Requirement Description View.

Since *Self-Healing Grid* is a top level system, all Need Descriptions can be traced to it. However, we should only include Use Cases that are directly related to the Context. If the Use Case can be delegated to the System Element of this Context, it is better to move it to the lower level Context. For example, we can consider the requirements *REQ01* in the context of *Self-Healing Grid*. Since the *Self-Healing Grid* is directly responsible to ensure that *REQ01* is satisfied, we create a Use Case based on *REQ01* for the Requirement Context View in the Context of *Self-Healing Grid* (see Figure 6.10).

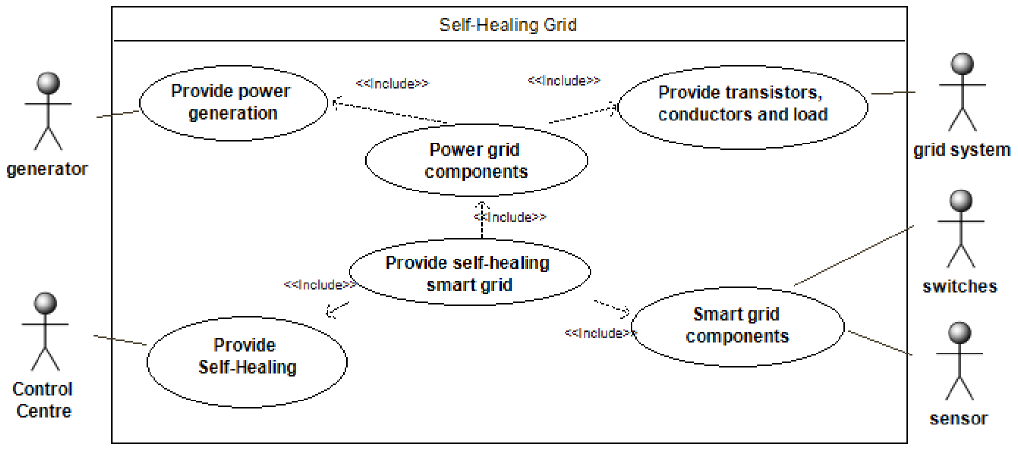


Figure 6.10: Smart grid case study: RCV for Self-Healing Grid Context

Not all Need Descriptions are considered in Use Cases in Figure 6.10. The Use Case *Provide Self-Healing* is not expanded here as the refinement for *REQ06* to *REQ11* is best left to be captured in the Context of *Control Centre*. Some other Need Descriptions such as *Save voltage level* from *REQ03* are not even mentioned here as they are not directly responsible for it. This is because the *Control centre* only controls the configuration of the grid through the *switch*.

Theoretically, the Requirement Context View can be created for each of the Contexts listed in the Context Definition View. However, the aim is to create as little Requirement Context Views as possible while ensuring that all the Need Descriptions are captured into Use Cases at the right Context. While there are plenty of Contexts that can be covered, we will not go into all the Contexts, but instead focus on the *Control Centre* Context as the *Provide Self-Healing* Use Case is handled by this Context and shown in Figure 6.11.

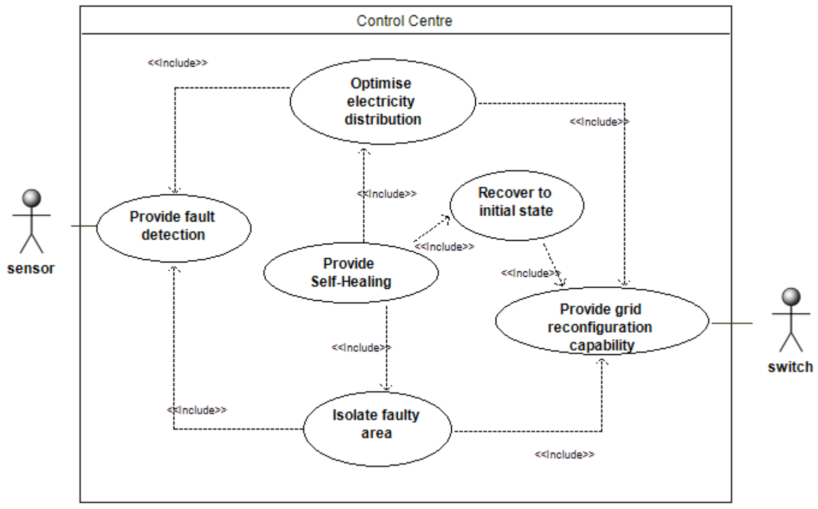


Figure 6.11: Smart grid case study: RCV for Control Centre Context

Traceability View

The Traceability View can be used to show the traceability between elements in the model such as the Source Elements, Need Descriptions, Use Cases, and System Elements. In the Requirement Context View, we mention that each Use Case should be based on a Need Description. However, it does not show which Use Cases are traced from which Need Description. This View can be used to show this traceability, as shown in Figure 6.12.

6.2.4 System Design

The next step in the V-model process is system design and development. The system design process can be split into multiple stages, working from higher level specification down into lower level implementation. However, instead of presenting the model following the chronological process, it is split into sections that focus on the structural or behavioural aspect of the system. This structure allows us to present the case study better as the autonomy behaviour is the focus of the research.

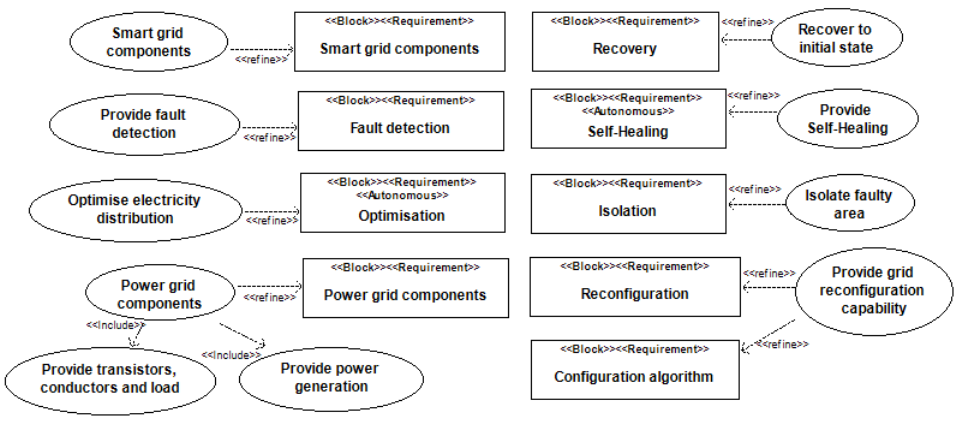


Figure 6.12: Smart grid case study: Traceability between Use Cases and Need Descriptions

6.2.4.1 System Structure

This section covers the architecture of the smart grid case study, focusing on the structural design. The Views related to this section are CPS Identification View, CPS Structure View, CPS Definition View, and CPS Configuration View.

CPS Identification View

The CPS Identification View identifies System Elements within the ACPS and draws their relationships. If a Context Definition View is created for the System Element Context, it will closely resemble the CPS Identification View as they both serve a very similar purpose. For the purpose of this case study, the CPS Identification View is identical to the Context Definition View for the System Element Context shown in Figure 6.9. Therefore, it will not be duplicated here.

CPS Structure View

While the CPS Identification View shows the high level system composition, the CPS Structure View shows the relationships between System Elements in greater detail along with their Properties and Functions. We present four instances of the CPS Structure View, with one for each System Element shown in Figure 6.9. The resulting views are the CPS Structure Views for *Control Centre* (Figure 6.13), *Grid System* (Figure 6.14), *Power Generation* (Figure 6.15), and *Switch Sensor System* (Figure 6.16).

CPS Interface View

Several views from the CPS Structure View show that the System Element could implement an Interface. Here, we defined the Interfaces between System Elements in the ACPS as the CPS Interface View. The CPS Interface View is realised using a block definition diagram, with each Interface represented by interface blocks. The Interfaces used in the ‘Self-Healing Grid’ are shown in Figure 6.17.

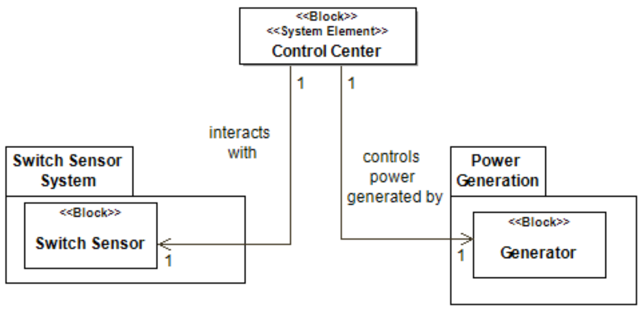


Figure 6.13: Smart grid case study: CPS Structure View for Control Centre

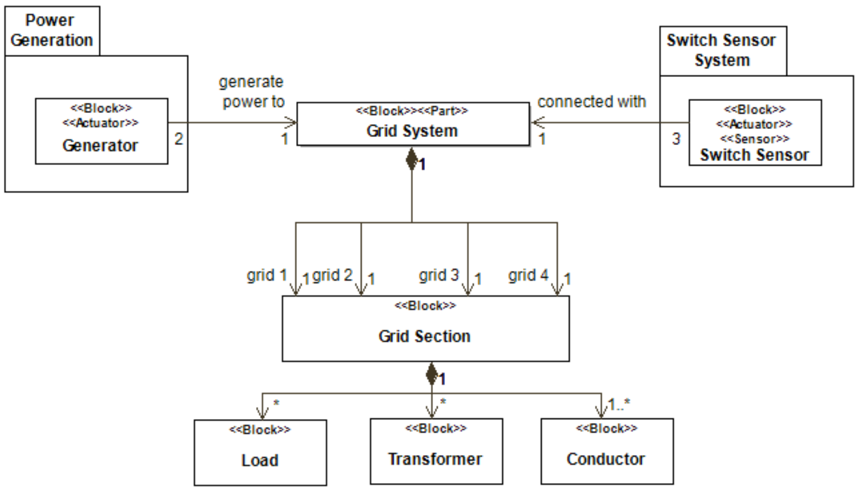


Figure 6.14: Smart grid case study: CPS Structure View for Grid System

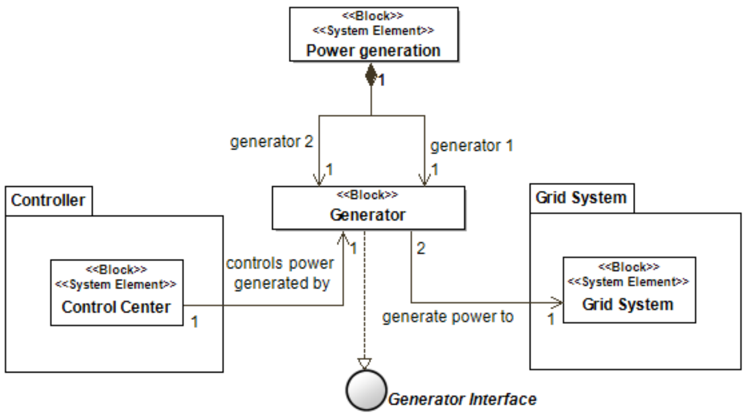


Figure 6.15: Smart grid case study: CPS Structure View for Power Generation

CPS Configuration View

The CPS Configuration View shows how the CPS is configured based on the connections between each System Elements, including its Interfaces. The CPS Configuration View is realised using an internal block

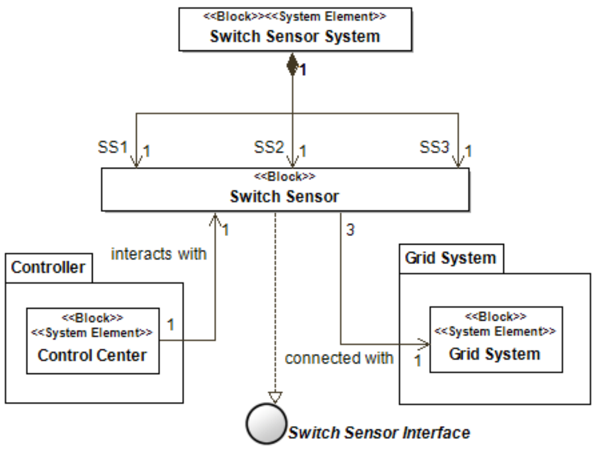


Figure 6.16: Smart grid case study: CPS Structure View for Switch Sensor System

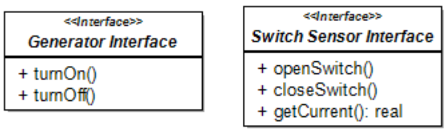


Figure 6.17: Smart grid case study: CPS Interface View

diagram, with parts to represent the System Elements. Parts can be connected to each other via item flows, and Interfaces through ports. Here, we present the CPS Configuration Views that capture the configuration of 'Self-Healing Grid' from three different levels of abstraction.

Figure 6.18 shows how there are three instances of *Switch Sensors* in the *Switch Sensor System*, and that they are all connected to the *Control Centre*. It also shows that there are two *Generators* that are similarly connected to the *Control Centre*. The connection between the *Generator* and *Control Centre* is actually not specified in the requirements, but it was included in the design for testing purposes.

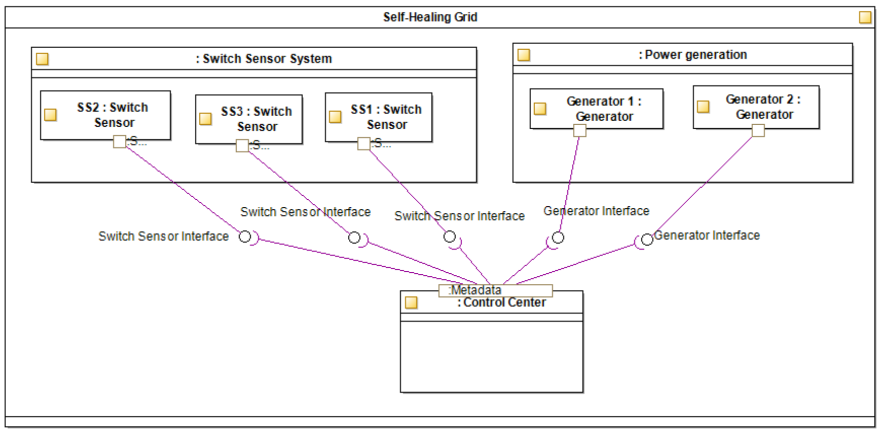


Figure 6.18: Smart grid case study: CPS Configuration View on Control Centre

Figure 6.19 shows the overall configuration of the *Grid System* with *Power generation* and *Switch Sensor System*. A *Grid Section* consists of *Loads*, *Transformers*, and *Conductors*. To see the full configuration of the system, the CPS Configuration View for each *Grid Section* is needed.

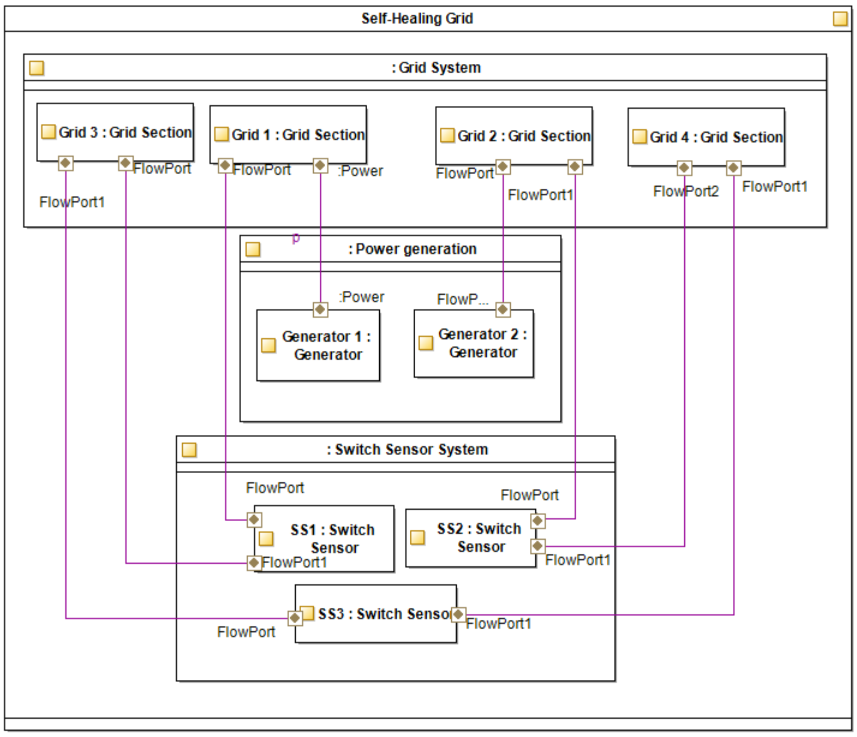


Figure 6.19: Smart grid case study: CPS Configuration View on Grid System

Figure 6.20 until 6.23 shows all the *Grid Section* configurations within the *Grid System* in the smart grid case study.

Based on the structure model shown in the CPS Structure View and CPS Configuration View, a 20-sim model can be created to reflect this model. Figure 6.24 shows the structure of a self-healing smart grid case study modelled in 20-sim.

6.2.4.2 Non-autonomous behaviour

As ACPSAF focuses on the autonomous behaviour of ACPS, the framework support for the description of non-autonomous behaviour is rather generic. The CPS Behaviour Viewpoint uses a SysML behaviour diagram and a parametric diagram to capture the behaviour of a System, System Element, Use Case, and System Function.

For example, we can consider the parametric behaviour of a *Transformer*. Parametric behaviour refers to behaviour that is best expressed by a network of mathematical and logical constraints. Transformers take in electrical power and change its potential. The power output should be equivalent to the input, so the current

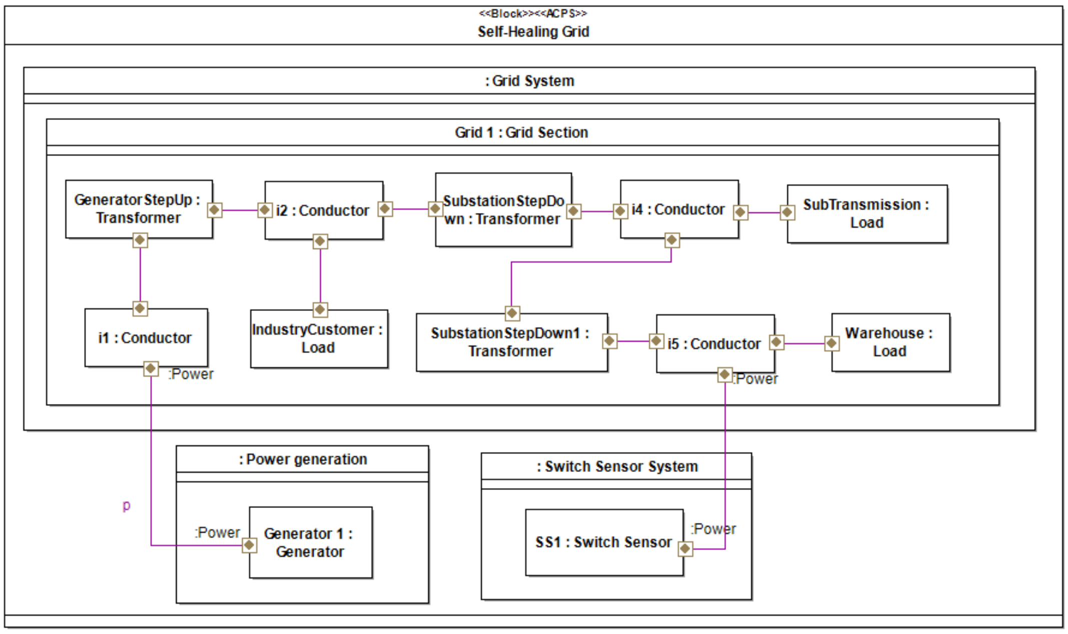


Figure 6.20: Smart grid case study: CPS Configuration View on Grid1

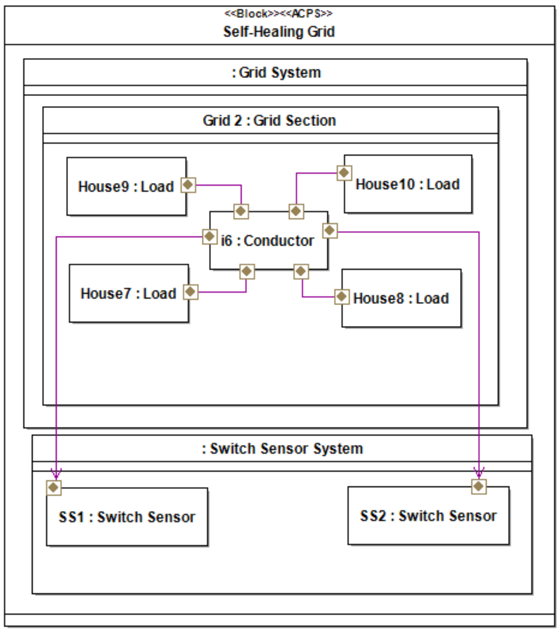


Figure 6.21: Smart grid case study: CPS Configuration View on Grid2

has to be scaled equally. This can be expressed in the parametric diagram as shown in Figure 6.25. Following this diagram, the equation for the *GeneratorStepUp* model in the 20-sim can be defined as shown in Figure 6.26.

The behaviour of the conductors, generators, and loads is fairly simple as it follows the default behaviour of

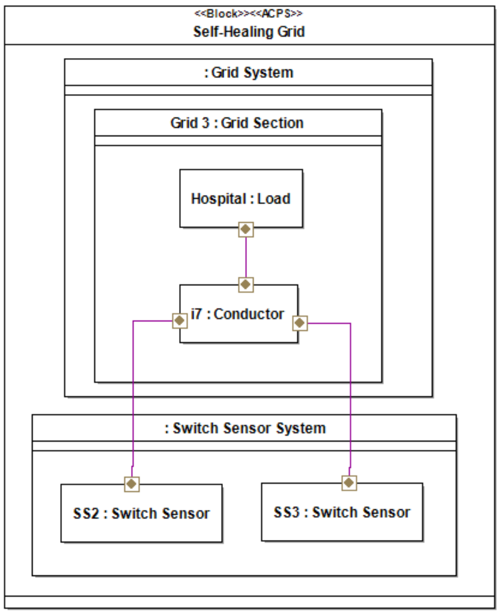


Figure 6.22: Smart grid case study: CPS Configuration View on Grid3

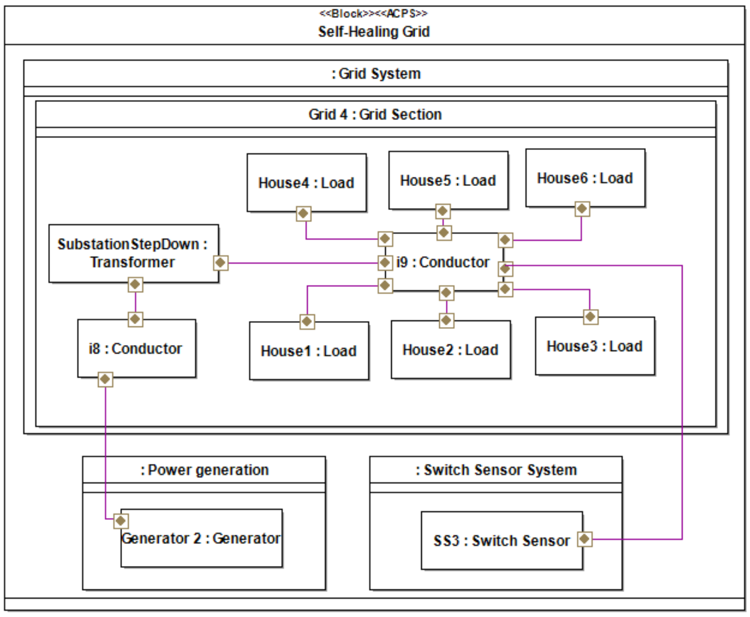


Figure 6.23: Smart grid case study: CPS Configuration View on Grid4

the basic electrical component it represents. There is no need to design the behaviour of basic components that are well-understood in the domain. The System Element *Switch Sensor* differs slightly to the other electrical components in the grid. As the name implies, the *Switch Sensor* is a component that provides the functionality of an electrical sensor and a switch. The *Switch Sensor* has both an internal and interface behaviour.

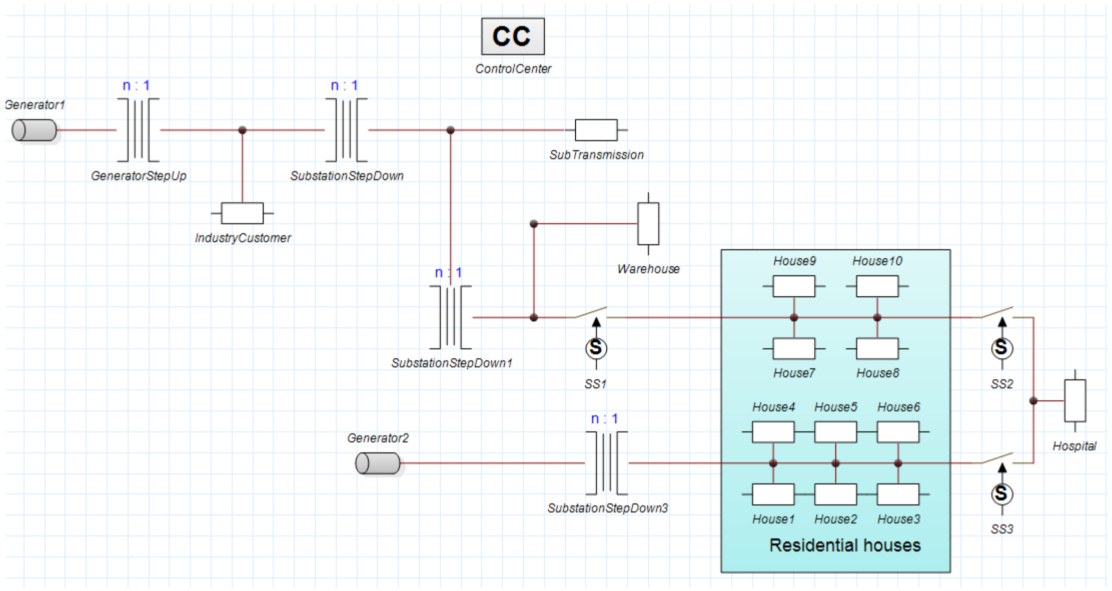


Figure 6.24: Smart grid case study: 20-sim model

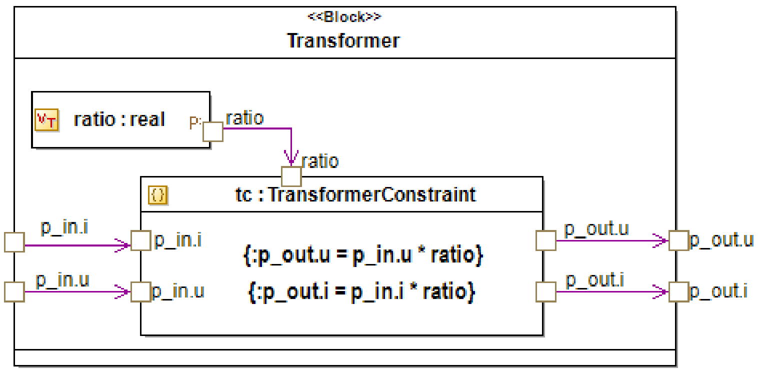
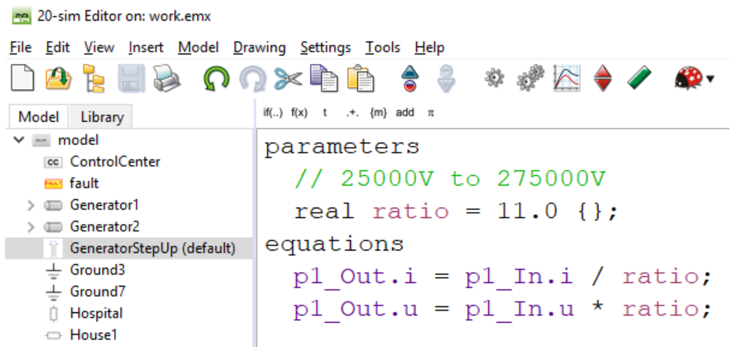


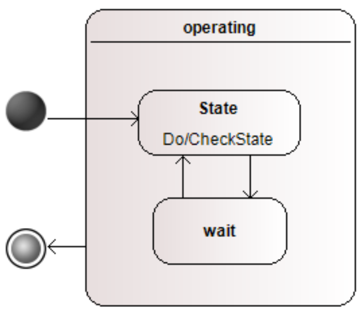
Figure 6.25: Smart grid case study: Transformer Constraint


20-sim Editor on: work.emx
File Edit View Insert Model Drawing Settings Tools Help
Model Library
model
ControlCenter
fault
Generator1
Generator2
GeneratorStepUp (default)
Ground3
Ground7
Hospital
House1

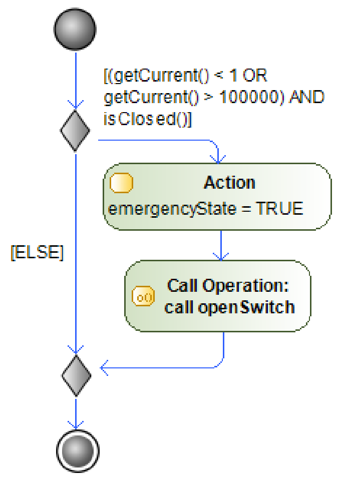
```
parameters
  // 25000V to 275000V
  real ratio = 11.0 {};
equations
  p1_Out.i = p1_In.i / ratio;
  p1_Out.u = p1_In.u * ratio;
```

Figure 6.26: Smart grid case study: GeneratorStepUp equation in 20-sim

Figure 6.16 shows that *Switch Sensor* implements an interface that allows the *Control Centre* to read the electric current passing through it and to open or close the switch. Note that the term ‘open’ here means to



(a) State behaviour of Switch Sensor



(b) CheckState behaviour in Switch Sensor

Figure 6.27: Smart grid case study: State and Activity behaviour of Switch Sensor System Element

break the circuit, and ‘close’ is to complete the circuit. The operators in this interface are very straightforward (e.g., getCurrent() simply return the p.i of the Switch Sensor), so the architecture model for these behaviours can be omitted as well. Although we are leaving out a few of the architectural models on the low-level implementation, these behaviours need to be implemented in the modelling tools to run the simulation. The full implementation code of the smart grid model for this case study is covered in Appendix B.

On top of the Interface, we would like the *Switch Sensor* to also implement a reactive mechanism that breaks the circuit when there is an anomaly in the power flow. One way to achieve this is to treat the *Switch Sensor* system using the embedded system approach. An embedded computer could be integrated in the *Switch Sensor* to periodically check the electricity flowing through. A simple state machine for this behaviour is captured in Figure 6.27(a). The *Switch Sensor* would periodically call the *CheckState* function that contains a simple logic check that will open the switch when certain conditions are met (Figure 6.27(b)).

This functionality can be implemented in VDM-RT as this behaviour is best captured with a discrete model. A snippet of this functionality is shown in Figure 6.28.

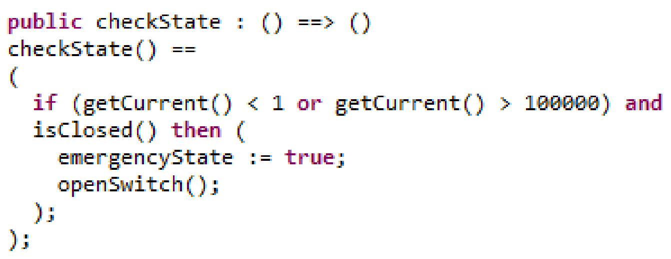

```
public checkState : () ==> ()
checkState() ==
(
  if (getCurrent() < 1 or getCurrent() > 100000) and
  isClosed() then (
    emergencyState := true;
    openSwitch();
  );
);
```

Figure 6.28: Smart grid case study: VDM snippet of CheckState function

6.2.4.3 Autonomous behaviour

Methods to architecting autonomy-related behaviour in ACPSAF are presented in the Autonomy Perspective. During requirement acquisition, we have identified an Autonomy Requirement (refer to Figure 6.7). The requirement describes that 'the system shall be able to maximise the number of customers receiving electricity'. Although not explicitly stated in the requirements, this behaviour needs to be achieved autonomously.

Autonomy Structure View

The Autonomy Perspective starts with focusing on the System Elements involved or required to achieve this autonomy. This is achieved by creating an Autonomy Structure View for each Autonomy Requirement. The Autonomy Structure View should list all the System Elements involved in fulfilling an Autonomy Requirement, and must include at least one Sensor, one Controller, and one Actuator. As we only identified one Autonomy Requirement for this case study, only one Autonomy Structure View is needed, as shown in Figure 6.29.

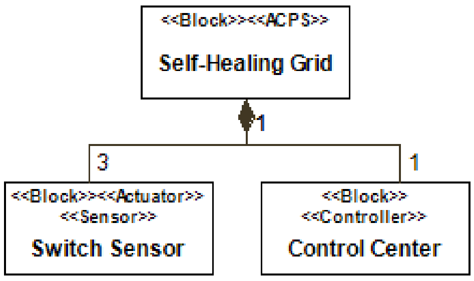


Figure 6.29: Smart grid case study: Autonomy Structure View

The Autonomy Structure View for this case study is very simple as it is limited to one Autonomy Requirement. When there are multiple Autonomy Requirements, there will be multiple Autonomy Structure Views needed, and each one might involve different types of sensors and actuators available in the ACPS.

Autonomy Context View

The Autonomy Context View defines the Use Cases that the *Controller* has to satisfy in order to fulfil the Autonomy Requirement. The Autonomy Context View for this case study is shown in Figure 6.30. The Autonomy Context View is defined with the Use Case Diagram, with the controller as the system boundary. The *Sensors* and *Actuators* identified in the Autonomy Structure View will be included as an external actor. The first Use Case could be directly derived from the Autonomy Requirement. In this case, it is the 'Optimisation'. The Use Case *Optimisation* can be refined into two separate Use Cases, which are to *Calculate available configuration* and *Select best configuration*.

As Autonomy Requirements are dependent on the state of the environment that is sensed by the sensors, one or more of the Use Cases must be traced to a sensor outside the system boundary. In this case, the Use Case

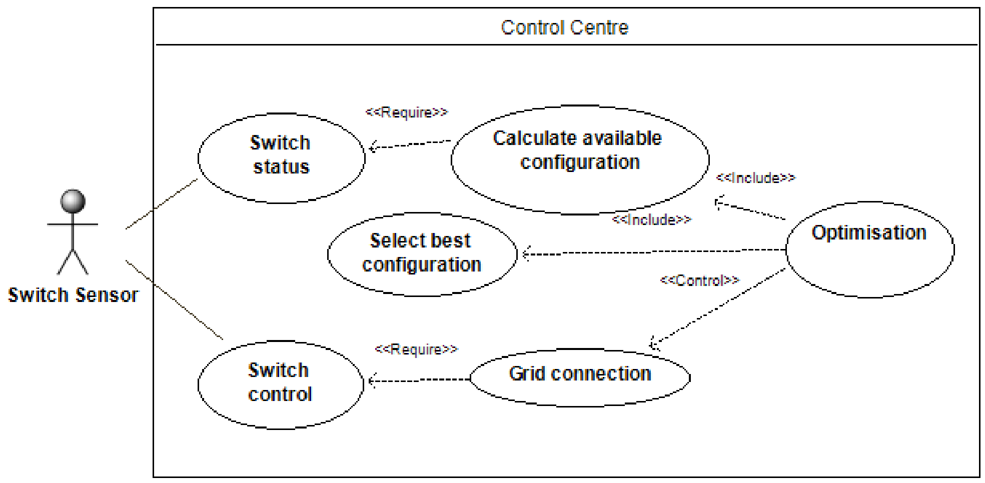


Figure 6.30: Smart grid case study: Autonomy Context View

‘Calculate available configuration’ requires the status of the grid in order to determine which configurations are possible. This knowledge can only be retrieved from the sensors distributed in the grid.

Typically, there will be many more sensors distributed in the grid, some of which (e.g., smart meters) are directly installed in the customer’s location. Using this information, the control centre would be able to deliver a more reliable service as it can confirm the effectiveness of its reconfiguration. Our case study uses a more naive approach where it assumes that a fault does not exist in the grid if a fault is not detected in the grid.

The *Switch Sensor* is designed in such a way that it would automatically isolate the faulty area (see Figure 6.27). The *Switch Sensor* affected by this fault will enter an ‘emergency’ mode, where it is unsafe for it to be connected to the grid. Since the *Switch Sensor* has to break the circuit, the number of available grid configuration changes. Therefore, the *Control Centre* needs to know the emergency status of the *Switch Sensor* in the grid to be able to calculate the available grid configurations.

The next step is to complete the connection between the main Use Case to the actuator. There is always a minimum for two Use Cases in between the main Use Case and the actuator. The first Use Case is the environment that it intends to influence and second is the type of actuator manipulation it requires to achieve it. In this case, the *Control Centre* needs to control the *Grid connection* to achieve *optimisation*. For the *Control Centre* to control the *Grid connection*, it requires a *Switch control*, which is provided by the *Switch Sensor*.

Note that the relationship between the main Use Case and the *Grid connection* is stereotyped with <<Control>>, which is a special stereotype from ACPSAF. Although it does not apply to this case study, there are examples of ACPS that could achieve an Autonomy Requirement in several ways. For example, a self-driving car could achieve an Autonomy Requirement such as ‘Avoid Collision’ by either changing its wheel direction or activating its brake system. In this self-driving car example, the controller has a choice to control direction, velocity, or both. Each type of control requires different types of actuator control. A similar scenario to this is included in the next case study focusing on UAVs. When the Autonomy Context View has been defined, the next step is to describe the behaviour that satisfies those Use Cases.

Autonomy Behaviour View

The Autonomy Behaviour Views are treated the same way as the CPS Behaviour View. All types of SysML behaviour diagrams and parametric diagrams can be used to capture the behaviour of the Use Cases described in the Autonomy Context View. Figure 6.31 shows the state behaviour of the *Control Centre*, while Figure 6.32 and 6.33 show the behaviour of each action call in the state.

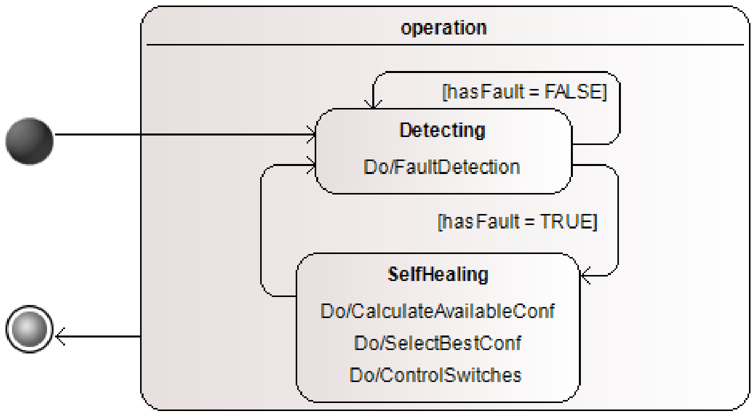


Figure 6.31: Smart grid case study: State Behaviour of Control Centre

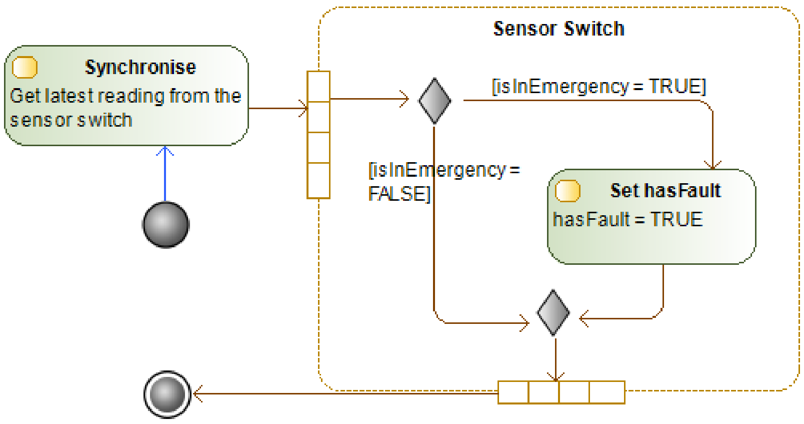


Figure 6.32: Smart grid case study: FaultDetection Behaviour in Control Centre

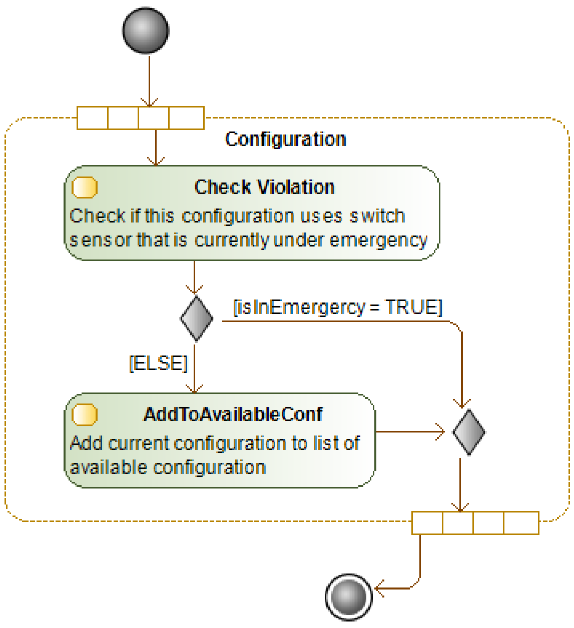

Configuration
Check Violation
Check if this configuration uses switch sensor that is currently under emergency
[isInEmergercy = TRUE]
[ELSE]
AddToAvailableConf
Add current configuration to list of available configuration

Figure 6.33: Smart grid case study: CalculateAvailableConf Behaviour in Control Centre

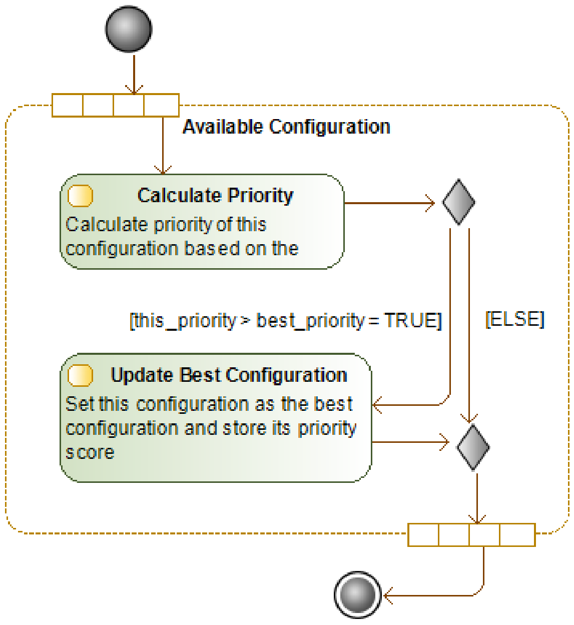

Available Configuration
Calculate Priority
Calculate priority of this configuration based on the
[this_priority > best_priority = TRUE]
[ELSE]
Update Best Configuration
Set this configuration as the best configuration and store its priority score

Figure 6.34: Smart grid case study: SelectBestConf Behaviour in Control Centre

6.2.5 Verification and Validation

ACPSAF focuses on verification and validation using co-modelling and simulation techniques. However, ACPSAF does not have a process to systematically translate SysML models into the domain model. ACPS engineers have to use their expertise to create the domain model from the SysML model. At this point,

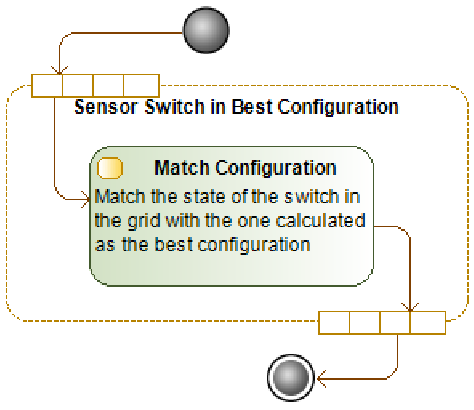

Sensor Switch in Best Configuration
Match Configuration
Match the state of the switch in the grid with the one calculated as the best configuration

Figure 6.35: Smart grid case study: ControlSwitches Behaviour in Control Centre

we assume that the system model from the previous section has been translated into a domain model. This model will not be included here, but can be found in Appendix B. For the domain model, we use the default modelling tools directly supported in the INTO-CPS tool-chain, which are the Overture for describing DE model in VDM-RT and 20-sim for CT models.

We will not go through the process of translating each SysML model into a domain model. ACPSAF provides Viewpoints to describe the architecture of the domain model and states that there must be traceability to the SysML model. The Architecture Structure View shows the structure of the domain model, with tags specifying whether it is a DE or CT model, and the platform in which it is developed. The Architecture Structure View also shows the connection port of each model, where data will be exchanged during co-simulation. Figure 6.36 shows the Architecture Structure View for this case study.

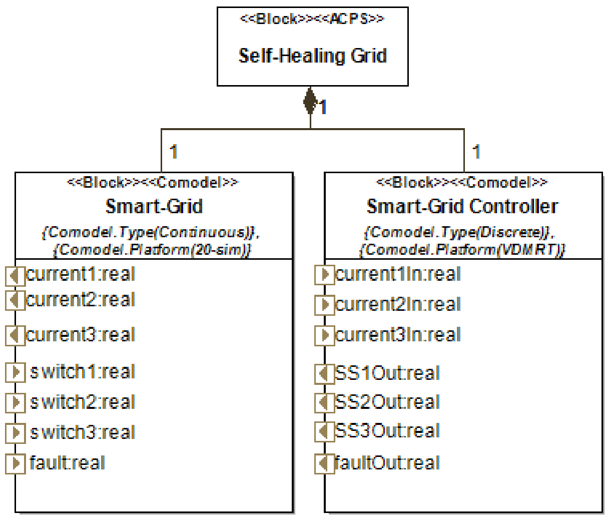


Figure 6.36: Smart grid case study: Architecture Structure View

The exact instances created for each of the models shown in the Architecture Structure View and the connections between them are shown in Figure 6.37 as a Connection View. To ensure that the domain model reflects those of the SysML model, each of the Comodel blocks should be traceable to one or more System Elements making up the ACPS. For complete traceability, all System Elements should be covered in the System Traceability View and should have a Comodel block traced into it. Figure 6.38 shows the System Traceability View for this case study.

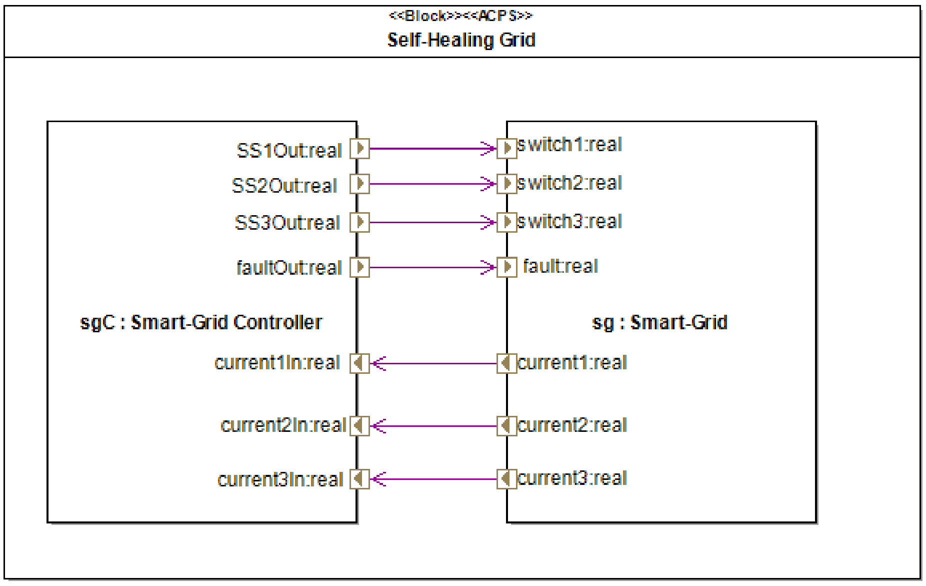


Figure 6.37: Smart grid case study: Connection View

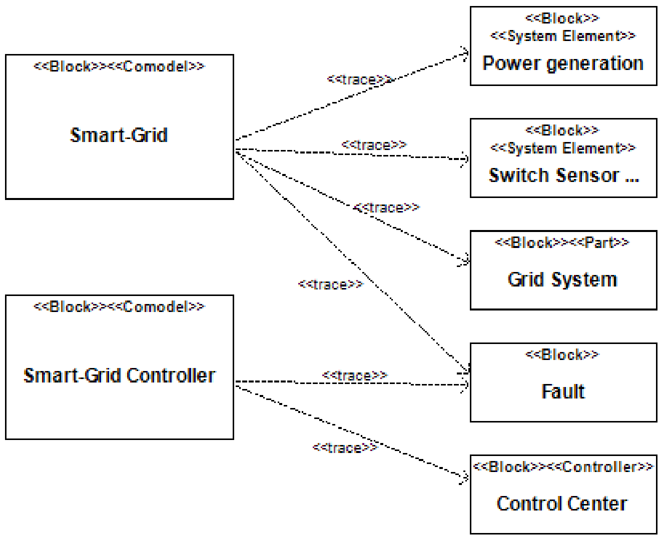


Figure 6.38: Smart grid case study: System Traceability View

However, test cases are needed to validate the model and have not been covered in the previous sections yet.

Self-healing behaviour is triggered under the presence of faults in the grid. To do this, we will be doing fault modelling. A fault model is an engineering model of something that could go wrong in the construction or operation of a piece of equipment, which can be used to predict the consequences of a given fault [145, 146]. In our case, we want to introduce fault in the model to induce the self-healing mechanism of the smart grid.

There are multiple ways to simulate the presence of faults in the grid. The first is to override the sensor reading from the *Switch Sensor* at a specific simulation time to simulate the sensor behaviour in the presence of fault. The advantage of using this method is that it is easily adjustable for multiple different fault scenarios as everything is done programmatically. However, this is not the preferable method here as this means that we will not be able to test the reliability of the sensor reading in the presence of fault, and we would also need to manipulate other components in the grid as a single fault in the grid is likely to affect multiple loads, transformers, and generators as well.

Another method that is more realistic is to introduce a fault model in the physical plant of the grid. As opposed to injecting the fault directly into the DE model (which essentially models a cyber attack to give a false reading to the sensor), physical faults introduced on the CT model must be detected by the sensor, simulating what would happen in real-life scenarios.

We consider the type of fault where the conductor wire is disconnected. We assume that both ends of the disconnected wire are suspended in mid air, so there is no ground fault. There is no easy way to implement this exact type of fault in 20-sim, but we can simulate similar effects using a resistor.

We introduce a resistor into the grid called the *'fault'*. During normal operation, the *fault* resistor would inflict a negligible amount of resistance. However, it can also simulate the presence of fault by applying a high level of resistance. The resistance should be high enough such that the amount of electricity passing through it is negligible. In turn, this would simulate the absence of a power supply to the other end of the faulty area. The disadvantage of using this approach is that the CT model has to be modified if the fault location were to be moved to a different location for a different scenario.

The controller uses the reading from the sensor to detect anomalies in the grid. Since the grid is based on an AC system, the controller will be storing several readings and calculating the RMS from those readings. In our case, we will be storing 5 time-step worth of sensor readings. One advantage of this approach is that we can easily adjust our controller to be more tolerant to transient faults, such as electrocuted birds which briefly short a line. We can extend the number of readings being stored in the history and implement a form of safeguard that only triggers the self-healing mechanism when the fault has been detected for several seconds.

For our test case, we locate the fault in *Grid 2*, located in between *SS1* and *SS2*. The updated 20-sim model with the *fault* resistor included is shown in Figure 6.39.

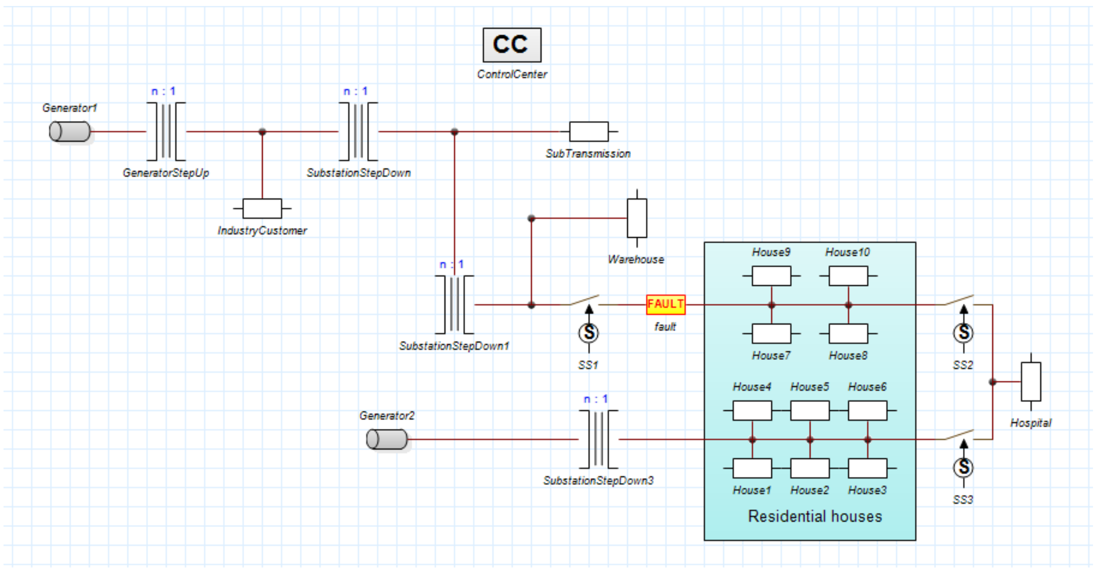


Figure 6.39: Smart grid case study: 20-sim model with fault

Autonomy Scenario View

To verify and validate that the model works as stated in the requirement, we need to define a Scenario for the test case. We need to produce Scenarios that can test the system against the *Self-Healing* requirement. Depending on the project, multiple Scenarios might be needed to fully test the system against all the requirements. However, the *Self-Healing* requirement and its sub-requirement can be tested with a single Scenario involving faults.

The scenario can be captured as an Autonomy Scenario View using a SysML sequence diagram, demonstrating a sequence of interactions between System Elements in the smart grid. The scenario involves a relatively large number of interactions to fit into a single diagram. Instead of one cluttered sequence diagram, we will split the scenario into multiple sequence diagrams before assembling the references into a single sequence diagram.

One interaction that we expect from the Scenario is for the *Control Centre* to set an initial state to the configuration of the *Switch Sensors* in the grid. This interaction can be expected at the beginning of the Scenario, as well as after fault recovery. After the initial setup, the *Control Centre* should check the *Switch Sensor* for the emergency state periodically. Eventually, a fault would be introduced to the grid and the *Control Centre* would detect the *Switch Sensor* in the emergency state. It will then go through the optimisation behaviour and enforce a different configuration to the grid.

The *fault* for this Scenario is placed strategically such that the *hospital* would experience a short dropout. The fault would disconnect the power supply from reaching the hospital, but there is an alternative grid configuration that allows power to be supplied to the hospital from a different generator. Eventually, the fault would be fixed, and the *Control Centre* would be notified. The *Control Centre* would then revert the grid back to the initial configuration. All the described sequences are captured by the sequence diagrams from Figure 6.40 to 6.44. The full Scenario combining all the mentioned sequences is shown in Figure 6.45.

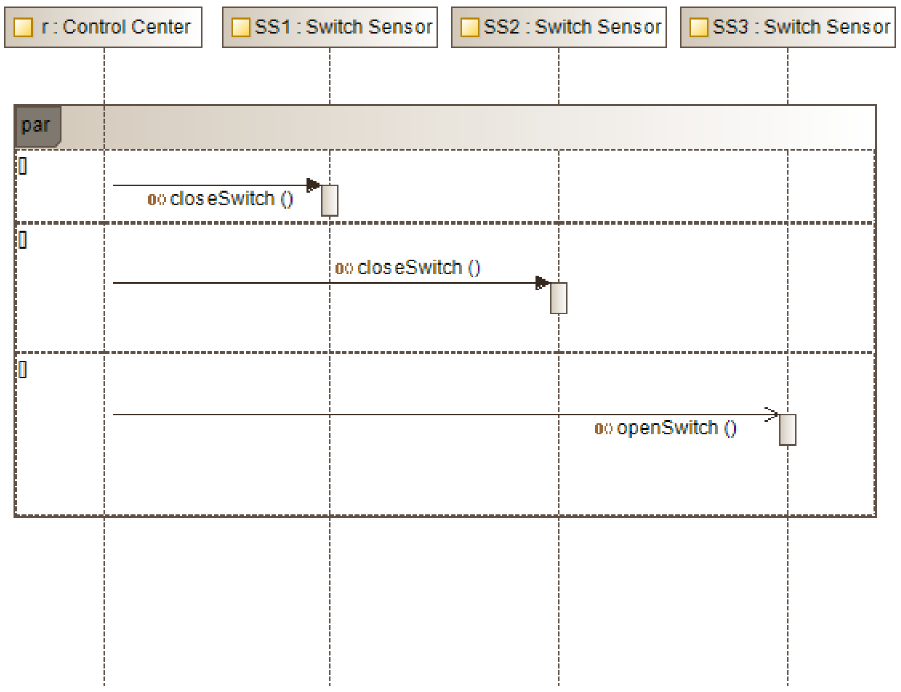


Figure 6.40: Smart grid case study: Initial Setup sequence Scenario

With this scenario, we can produce a Traceability View that shows the full traceability between all the components in the model created. Figure 6.46 shows the traceability between Source Elements, Need Descriptions, Use Cases, System Elements, and Scenarios, focusing on *Isolation* and *Optimisation*. More diagrams can be created to show the complete traceability to and from other Need Descriptions.

Simulation results

The simulation results shown here are taken from 20-sim tools as they have a built-in feature that produces relevant graphs for Scenario validation. The sequence shown in Figure 6.45 does not show the timestamp between each event. For this scenario, we set the fault to manifest at 5s into the simulation time and the recovery at 15s simulation time. There are several properties in the model that we can observe to validate the scenario.

To help us compare the test case Scenario against the Use Case it is trying to validate, we create some form of a test table listing out the Use Case, the element in the model being tested, the expected behaviour, and the test result.

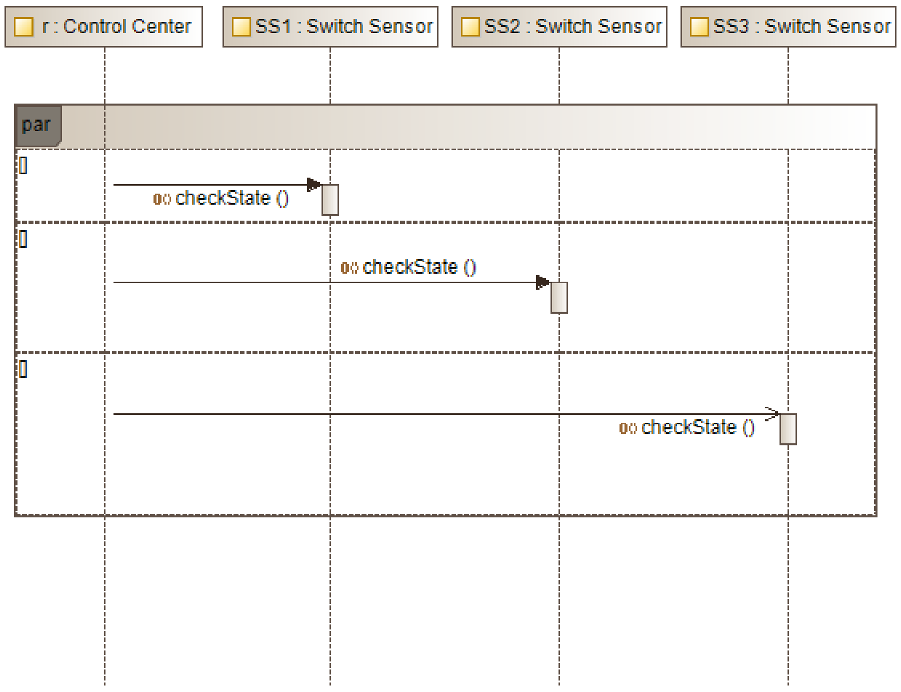


Figure 6.41: Smart grid case study: check State sequence Scenario

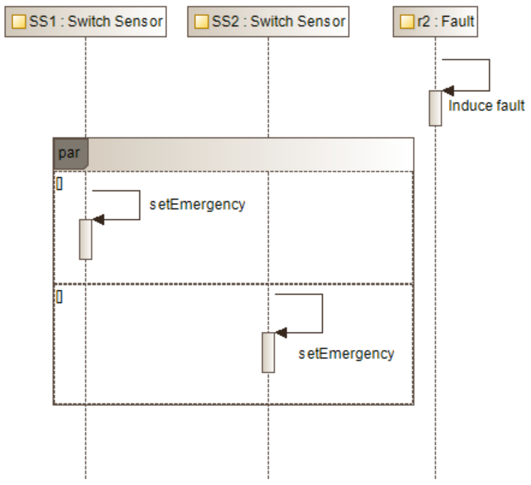


Figure 6.42: Smart grid case study: Fault Scenario sequence Scenario

6.2.6 Dependability Cage

One of the project requirement sets for this case study is to demonstrate how the smart grid model developed with ACPSAF can be integrated with separate, existing verification and validation techniques for ACPS. For this case study, we consider the development of a run-time monitoring technique for ACPS. To be precise, we consider a variant of the run-time monitoring method called the 'Dependability Cages' [17]. Figure 6.50 provides an overview to the simplified version of the Dependability Cages approach.

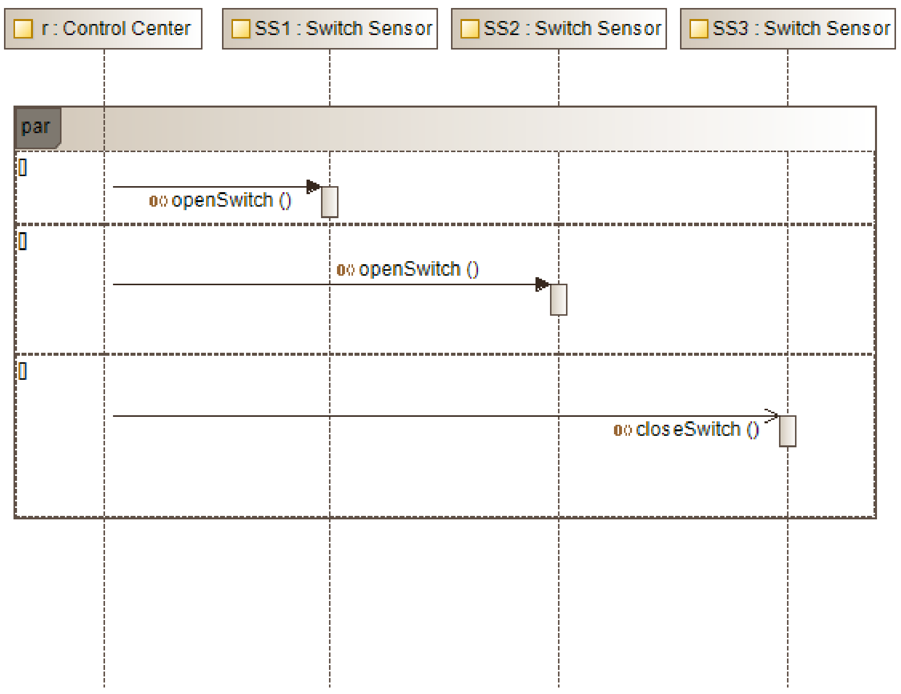


Figure 6.43: Smart grid case study: Optimise sequence Scenario

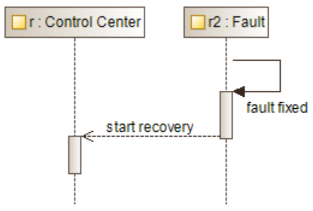


Figure 6.44: Smart grid case study: Recovery sequence Scenario

The motivation behind a run-time monitoring technique for ACPS is the acknowledgement that as ACPS adapts to the changing environment, it is possible that the ACPS produces behaviour that has not been tested during the design phase. As the behaviour is not tested, it is possible that the resulting behaviour is undesirable. The behaviour might be undesirable because it violates the system requirement or a stakeholder's Need that was not fully captured in the Requirement. Dependability Cages are used at development time and during operation time to check the correctness of the system behaviour with respect to its requirements.

For a given ACPS and its environment, the Dependability Approach makes a distinction between *engineered behaviour* and its *tested behaviour*. The *engineered behaviour* refers to the behaviour specified, modelled, and documented by the existing development artefacts. Meanwhile, the behaviour of the *engineered behaviour* that has been tested during development is denoted as *tested behaviour*.

At operation time (run-time), the Dependability Approach makes a distinction between *real behaviour* and

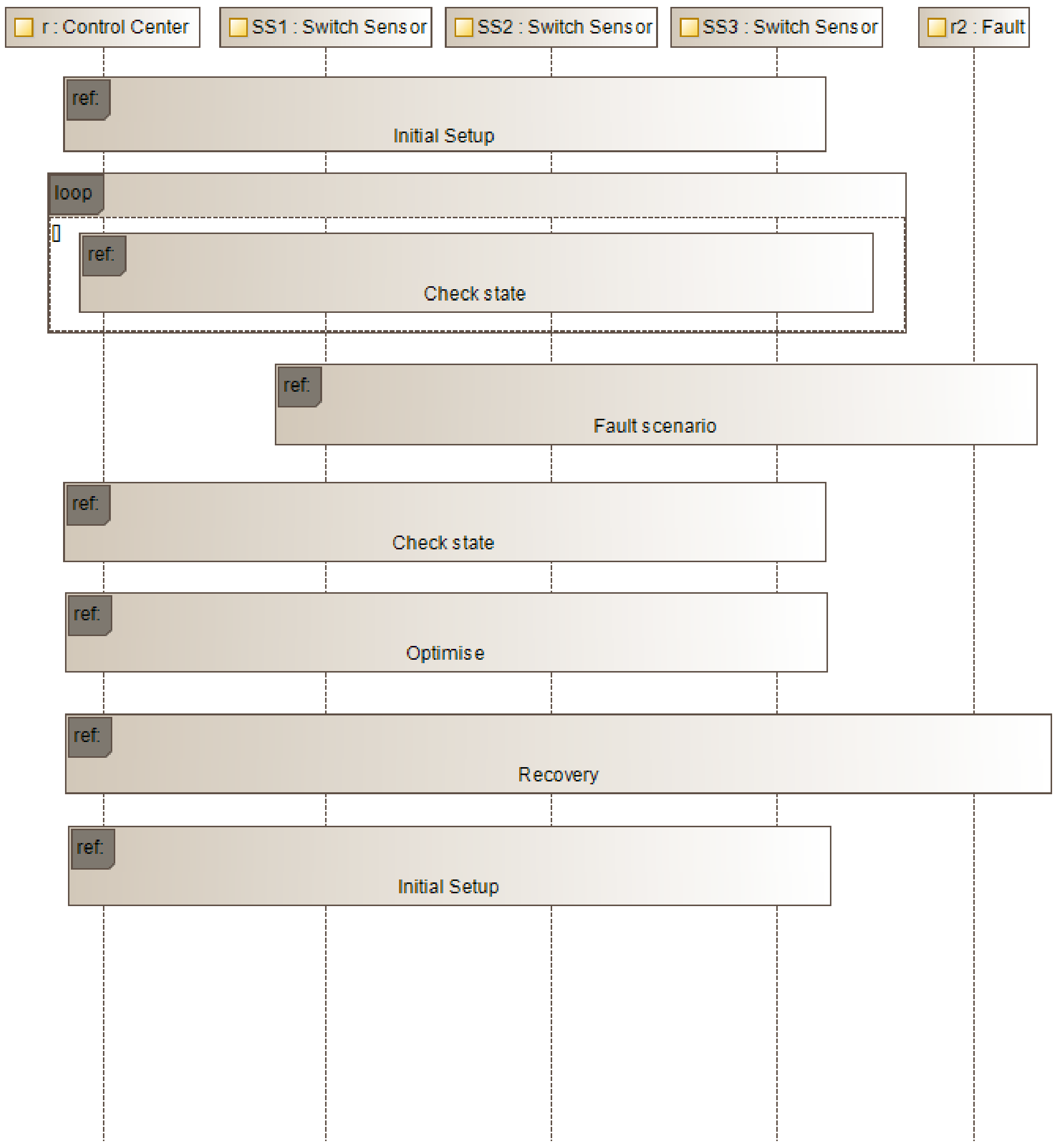


Figure 6.45: Smart grid case study: 'Main_test' Scenario

observed behaviour. *Real behaviour* is the behaviour at operation time, which may differ from the engineered behaviour, and the *observed behaviour* is a subset of *real behaviour* that represents the behaviour the ACPS monitors at operation time through the Dependability Cages.

When the test cases are completed during the development phase, the *tested behaviour* is included into the ACPS to be used by the ACPS during operation time. Eventually, the *observed behaviour* is transferred back to development time for the continuous evolution and development of ACPS. For this case study, we will not implement the whole concept presented by the Dependability Cage.

Our interest is to develop the *System Dependability Cage* for the development phase, obtain and integrate the

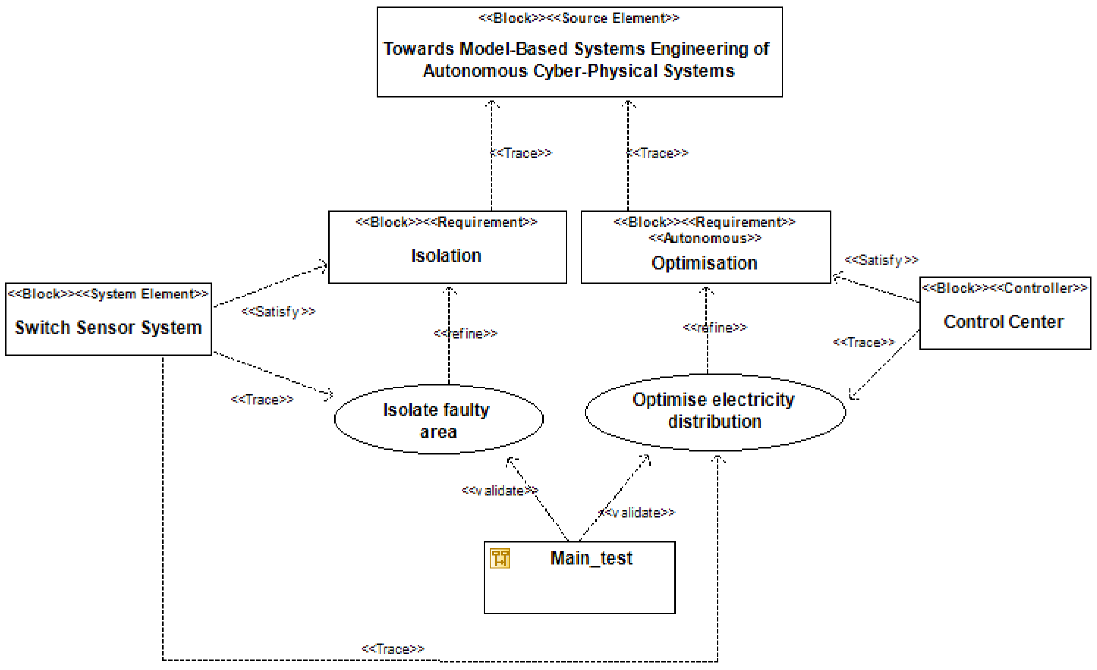


Figure 6.46: Smart grid case study: Traceability from Source Element to test case Scenario

Use Case	**System Element**	**Expected behaviour**	**Ref**	**Result**
Fault Detection	SS1 (Switch Sensor)	Switch in SS1 should transition from closed to open when fault occurs at 5s.	Fig 6.47	PASS
Fault Detection	SS2 (Switch Sensor)	Switch in SS2 should transition from closed to open when fault occurs at 5s.	Fig 6.47	PASS
Isolation	House 10 (Load)	House10 should not receive any power from the time fault started and recovered (5s to 15s).	Fig 6.48	PASS
Optimisation	Hospital (Load)	Hospital should receive continuous power supply throughout the simulation. Up to a few seconds of dropout due to reconfiguration delay is acceptable).	Fig 6.48	PASS
Optimisation	Generator 2 (Generator)	Generator 2 should be producing more power to facilitate the hospital after fault was induced.	Fig 6.49	PASS
Recovery	House 10 (Load)	House 10 should receive power again after 15s.	Fig 6.48	PASS
Recovery	SS1, SS2, SS3 (Switch Sensor)	All the switches pass 15s should revert back to their original state at the start of the simulation.	Fig 6.47	PASS

Table 6.2: Smart grid case study test table

Tested System Behaviour for operation purposes, and design the *System Dependability Cage* to influence the ACPS behaviour when it enters untested behaviour.

As our ACPS has only one Autonomy Requirement, the Dependability Cage will be developed to target the

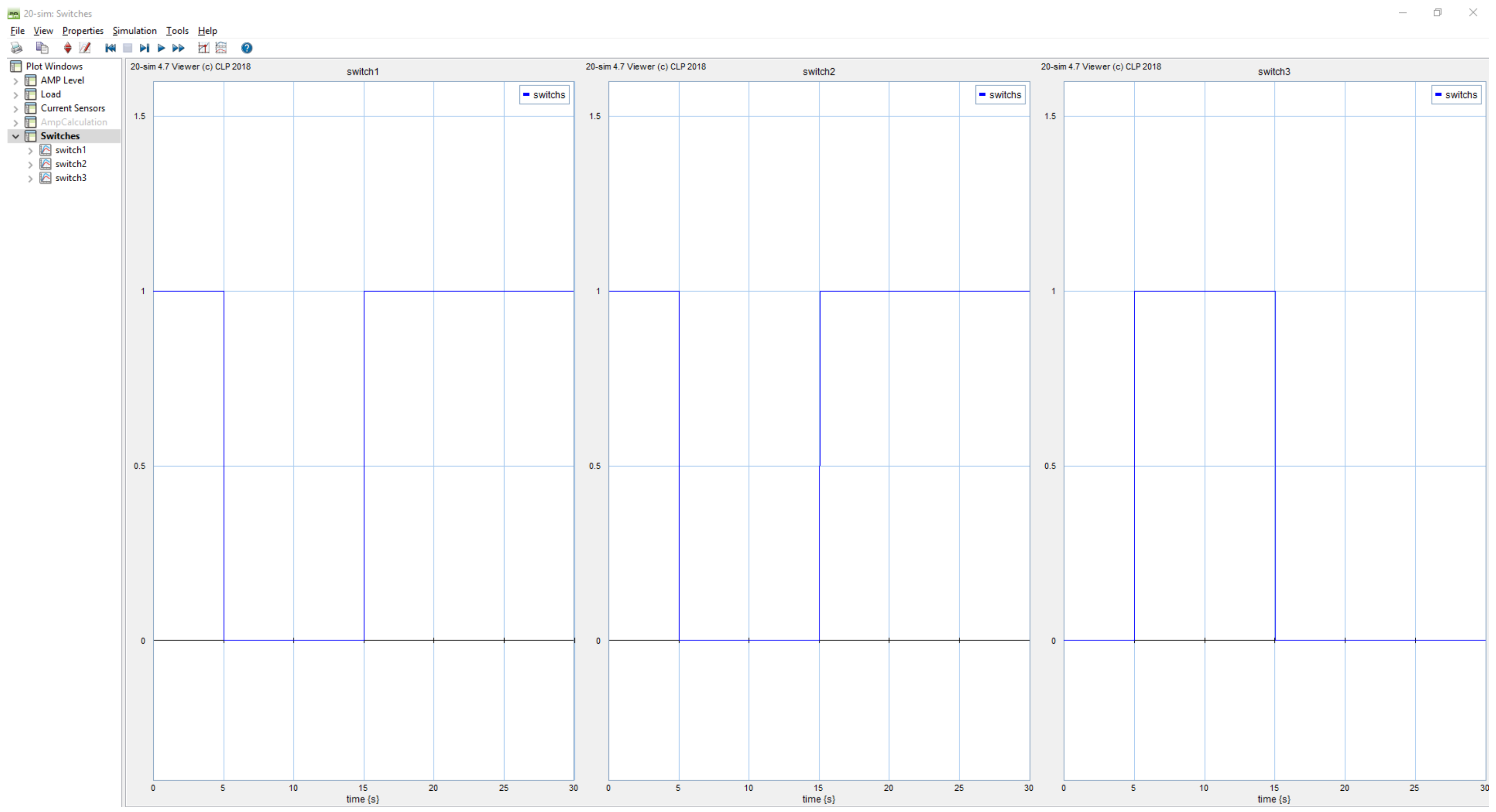


Figure 6.47: Smart grid case study: Switch status from Main_test Scenario

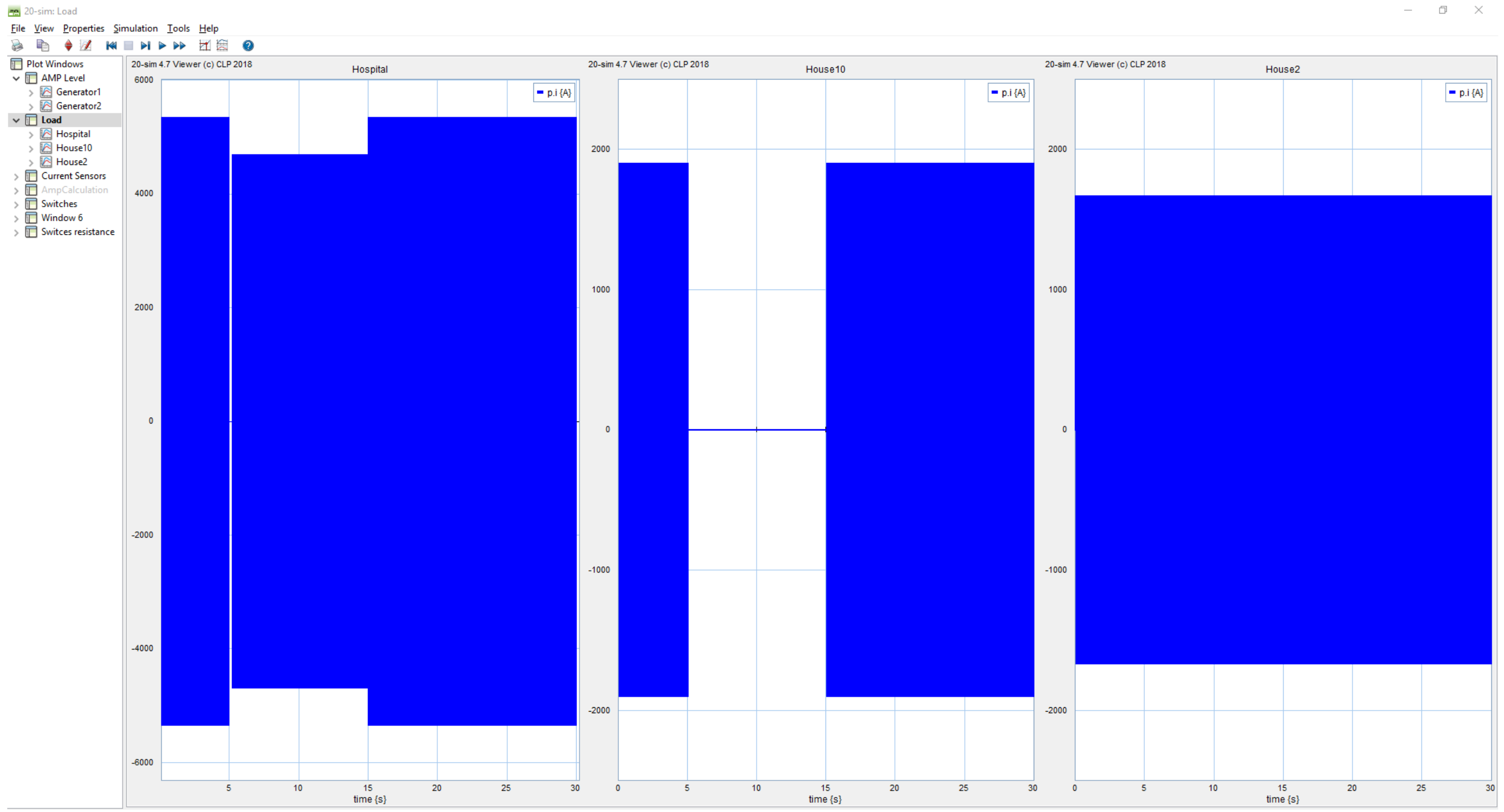


Figure 6.48: Smart grid case study: Load values from Main_test Scenario

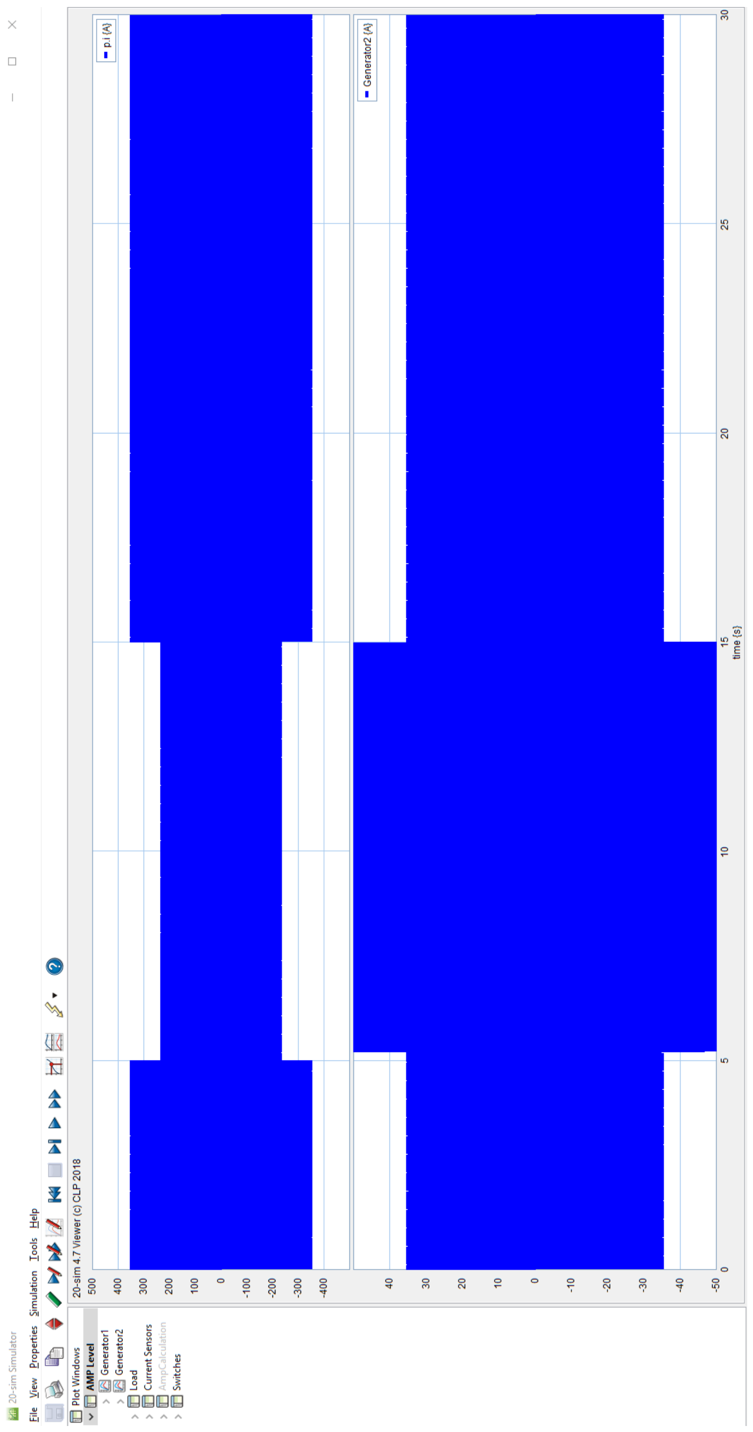


Figure 6.49: Smart grid case study: Generator values from Main_test Scenario

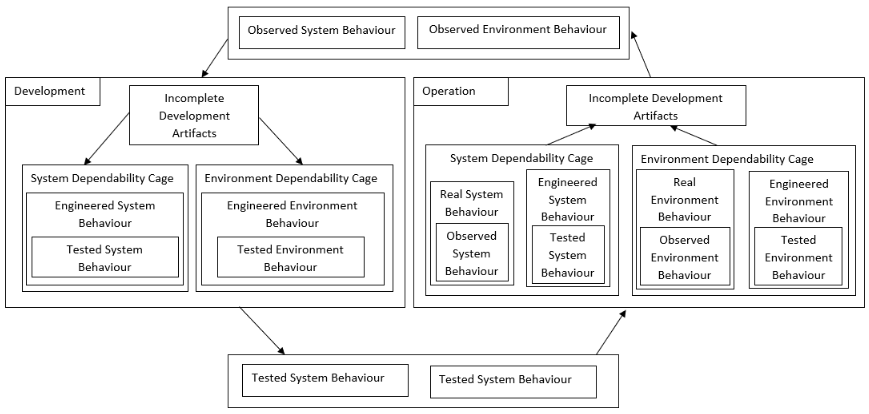

Observed System Behaviour
Observed Environment Behaviour
Development
Incomplete Development Artifacts
System Dependability Cage
Engineered System Behaviour
Tested System Behaviour
Environment Dependability Cage
Engineered Environment Behaviour
Tested Environment Behaviour
Operation
Incomplete Development Artifacts
System Dependability Cage
Real System Behaviour
Observed System Behaviour
Engineered System Behaviour
Tested System Behaviour
Environment Dependability Cage
Real Environment Behaviour
Observed Environment Behaviour
Engineered Environment Behaviour
Tested Environment Behaviour
Tested System Behaviour
Tested System Behaviour

Figure 6.50: Simplified Version of the Dependability Cages Approach

behaviour of the *Optimise* requirement. To be specific, the *Optimise* function in our VDM model finds the best configuration available at that moment and enforces that configuration to the switches in the grid (see Figure 6.51). However, the selected configuration might be undesirable or unsafe if the configuration has not been tested. This scenario does not make sense for our case study as we can definitely test all possible configurations before deployment.

```
public optimise : () ==> ()
optimise() ==
(
  dcl bestConf : seq of bool :=  priorityMap.getBestConfig().getConfValue();
  if(bestConf(1)) then (switch1.closeSwitch()) else (switch1.openSwitch());
  if(bestConf(2)) then (switch2.closeSwitch()) else (switch2.openSwitch());
  if(bestConf(3)) then (switch3.closeSwitch()) else (switch3.openSwitch());
);
```

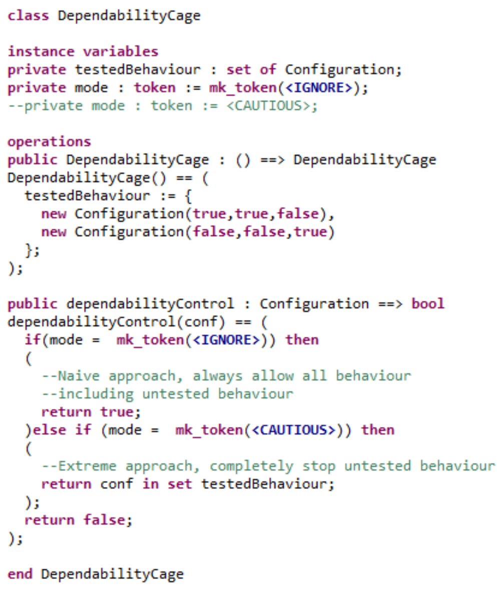
Figure 6.51: VDM snippet of optimise function

However, real-life application for the smart grid would be a very large, multi-connected network that might make testing all configurations unfeasible. This is especially the case if the smart grid were to include on-demand distributed energy sources such as wind turbines. The combination of possible configurations would change dynamically depending on the availability of the turbines. In those cases, the Dependability Cage would add an extra layer of protection heading towards an untested grid configuration.

There are multiple ways the Dependability Cage can influence the decision-making behaviour of the controller. An optimal Dependability Cage would ideally be able to determine the severity of the untested behaviour and decide whether to allow the controller to enforce the untested behaviour or go for a less optimal configuration that has been tested.

We implement a relatively simple Dependability Cage behaviour. We introduce two different types of modes in the Dependability Cage. The first mode is *IGNORE*, which essentially behaves as if the Dependability Cage is turned off. It simply allows both tested and untested behaviour to be enforced. The second mode is the *CAUTIOUS* mode, where it allows tested behaviour to be enforced and rejects all untested behaviour. The Dependability Cage does not suggest an alternative configuration. So in this case, it would behave as if the self-healing mechanism has been completely disabled.

The set of tested behaviours for our Dependability Cage are set to be the two configurations that we have simulated in the previous section. The first configuration is the initial setup (see Figure 6.40) and the second configuration is the optimised configuration that isolates grid section 2 (see Figure 6.43). The VDM-RT code snippet for the DependabilityCage class is shown in Figure 6.52.



```
class DependabilityCage

instance variables
private testedBehaviour : set of Configuration;
private mode : token := mk_token(<IGNORE>);
--private mode : token := <CAUTIOUS>;

operations
public DependabilityCage : () ==> DependabilityCage
DependabilityCage() == (
  testedBehaviour := {
    new Configuration(true,true,false),
    new Configuration(false,false,true)
  };
);

public dependabilityControl : Configuration ==> bool
dependabilityControl(conf) == (
  if(mode =  mk_token(<IGNORE>)) then
  (
    --Naive approach, always allow all behaviour
    --including untested behaviour
    return true;
  )else if (mode =  mk_token(<CAUTIOUS>)) then
  (
    --Extreme approach, completely stop untested behaviour
    return conf in set testedBehaviour;
  );
  return false;
);

end DependabilityCage
```

Figure 6.52: VDM-RT snippet of DependabilityCage class

When we simulate the model that is now updated with the Dependability Cage with the same fault scenario, it will produce the same kind of behaviour as the configuration is already saved as tested behaviour. To properly demonstrate the effects of the Dependability Cage, a new scenario needs to be carried out where the fault is moved to a different location. Figure 6.53 shows the new fault scenario, where *SS1* and the *fault* component have switched places. With this scenario, grid section 2 should not be isolated anymore as the fault is now located at grid section 1.

We test the Dependability Cage under both types of the available modes. When the simulation is performed with the Dependability Cage on the *IGNORE* mode, the smart grid should react to the fault and reconfigure with an optimised configuration. Figure 6.55 shows how the switch is configured as a reaction to the new fault scenario. Note that in this scenario, all loads continue to receive power supply (Figure 6.55) as it is not possible to isolate Grid Section 1 where the fault resides. This is because the fault is not located in between two switches.

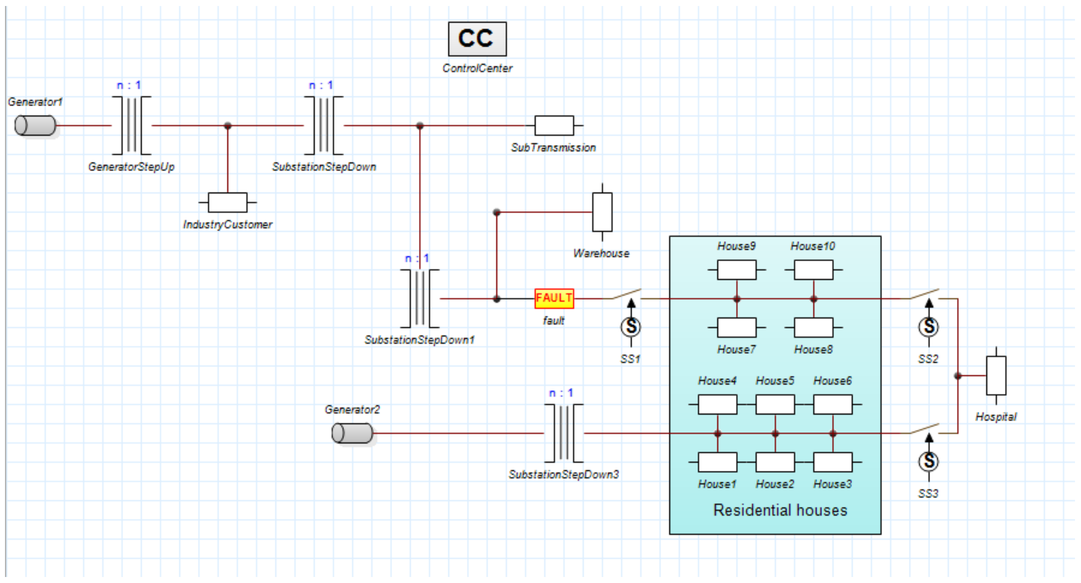


Figure 6.53: 20-sim model for Dependability Cage test

When the simulation is performed with the Dependability Cage on the *IGNORE* mode, the smart grid should not react to the fault as the smart grid is attempting to use a configuration that has not been tested. As a result, grid section 2 and 3 would not receive any power supply throughout the duration of the fault (Figure 6.56). It would only receive power back when the control centre initiates the recovery process after the fault is fixed. Figure 6.57 shows the state of the switch during the CAUTIOUS mode Scenario.

6.2.7 Evaluation

This case study demonstrates how ACPSAF can be used to support the design and architecting of a self-healing smart grid ACPS. The case study was engineered using MBSE methodology with ACPSAF, SysML, and INTO-CPS tool chain. The case study also produces at least one View for each of the Viewpoints in ACPSAF. Here, we evaluate the case study against each of the case study requirements defined in Section 6.1.

REQ1 Each case study should demonstrate the application of external ACPS techniques in conjunction with ACPSAF.

FULFILLED - the self healing smart grid case study was initially carried out with ACPSAF and then combined with an ACPS technique for run-time system monitoring. The ACPS technique taken is termed the Dependability Cage, which is meant to provide ACPS with greater dependability by deploying a run-time monitoring framework. Not all features from the Dependability Cage are demonstrated here as we only demonstrate the use of the System Dependability Cage. However, the case study does demonstrate that a Dependability Cage can be used with ACPSAF. This is carried out in Section 6.2.6.

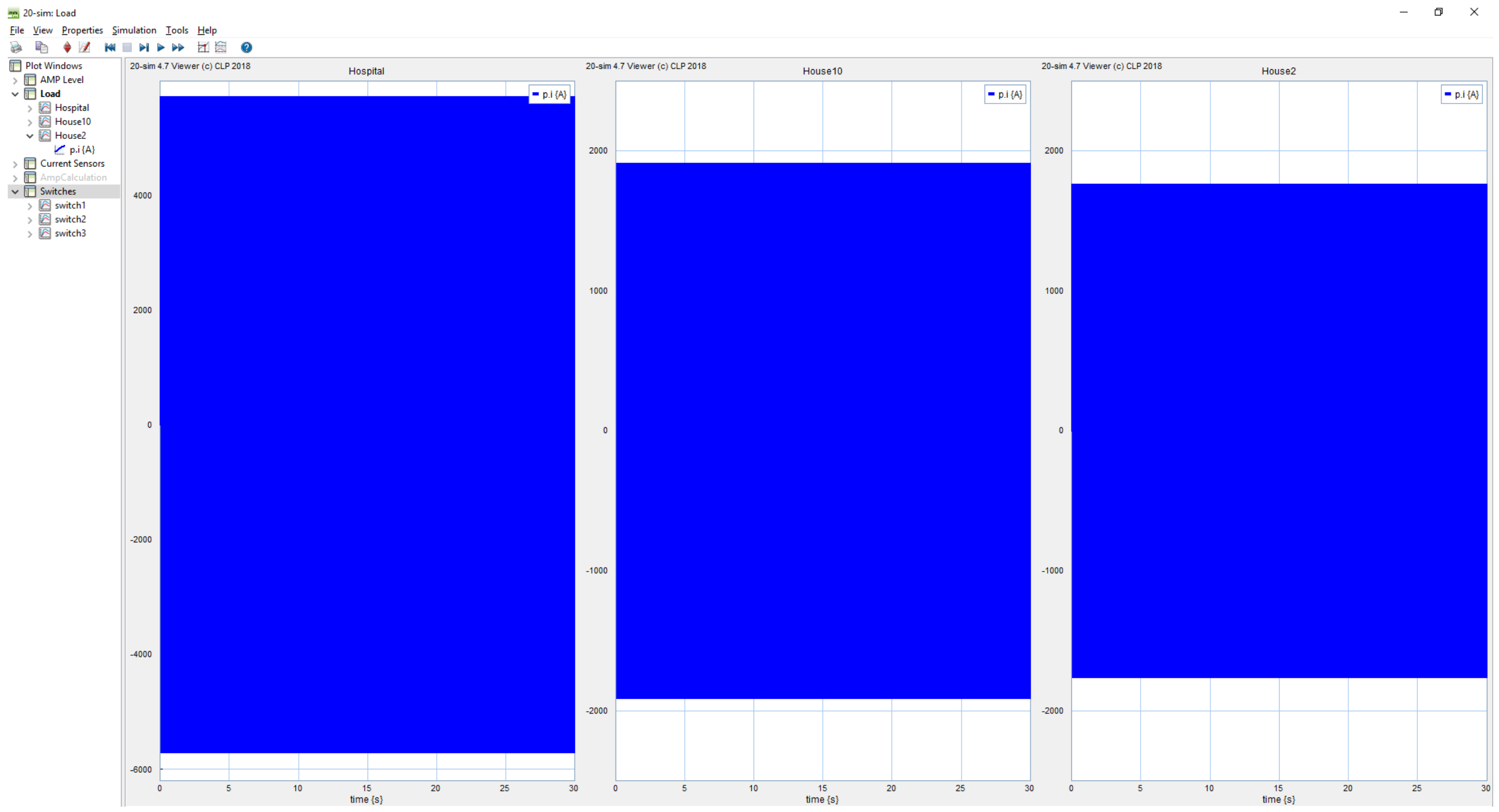


Figure 6.54: Smart grid case study: Load values in IGNORE mode Scenario

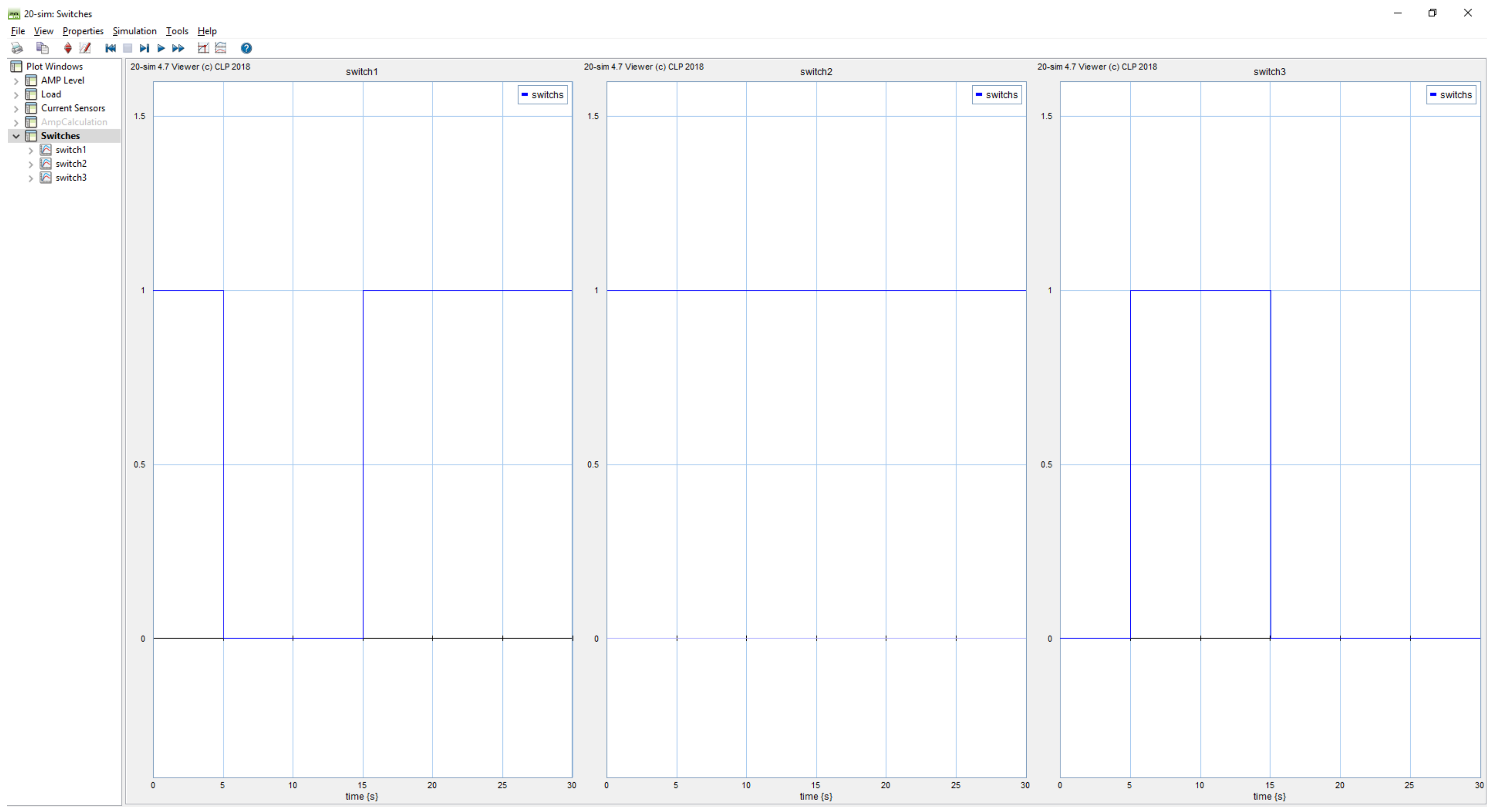


Figure 6.55: Smart grid case study: Switch values in IGNORE mode Scenario

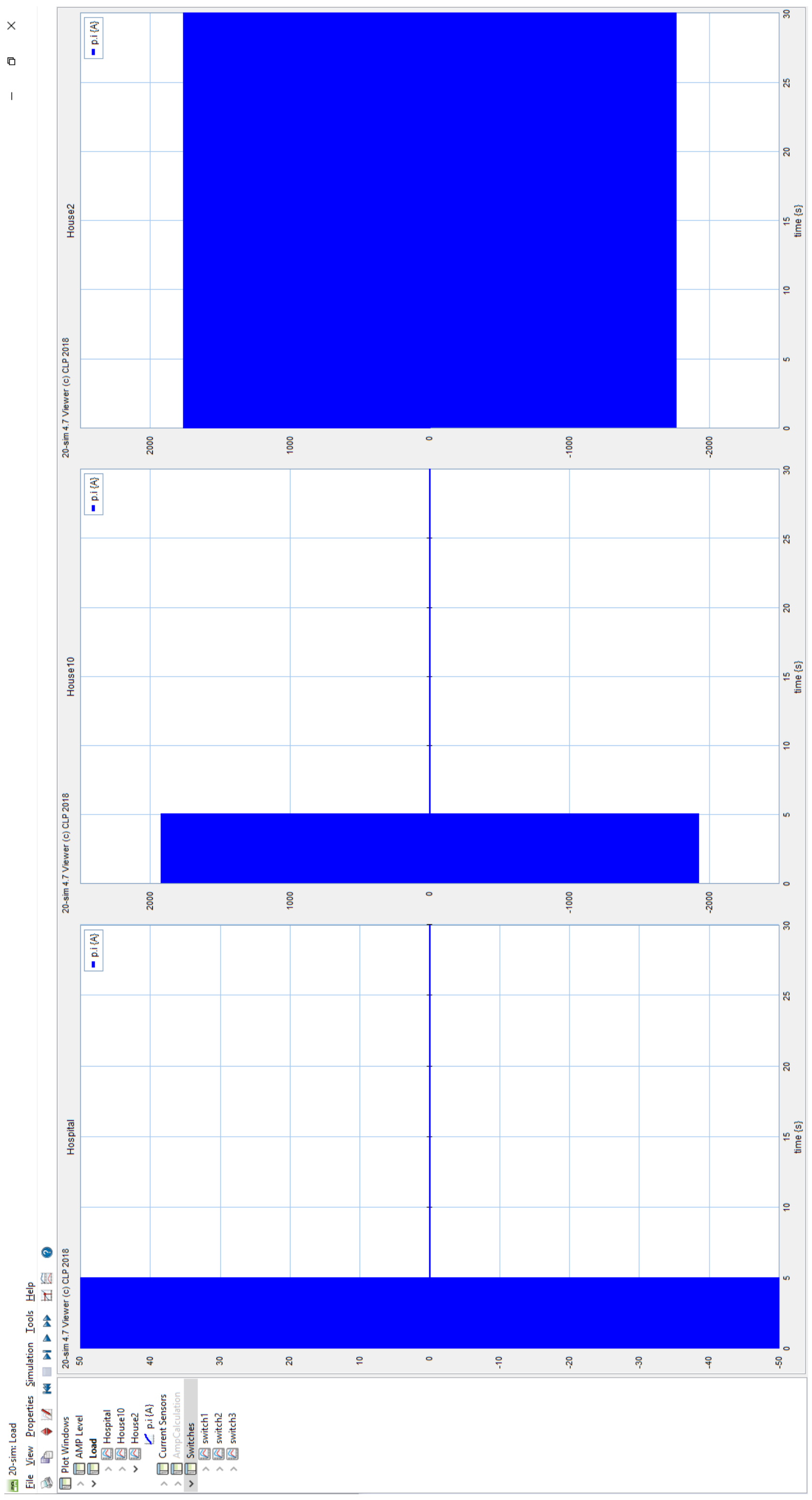


Figure 6.56: Smart grid case study: Load values in CAUTIOUS mode Scenario

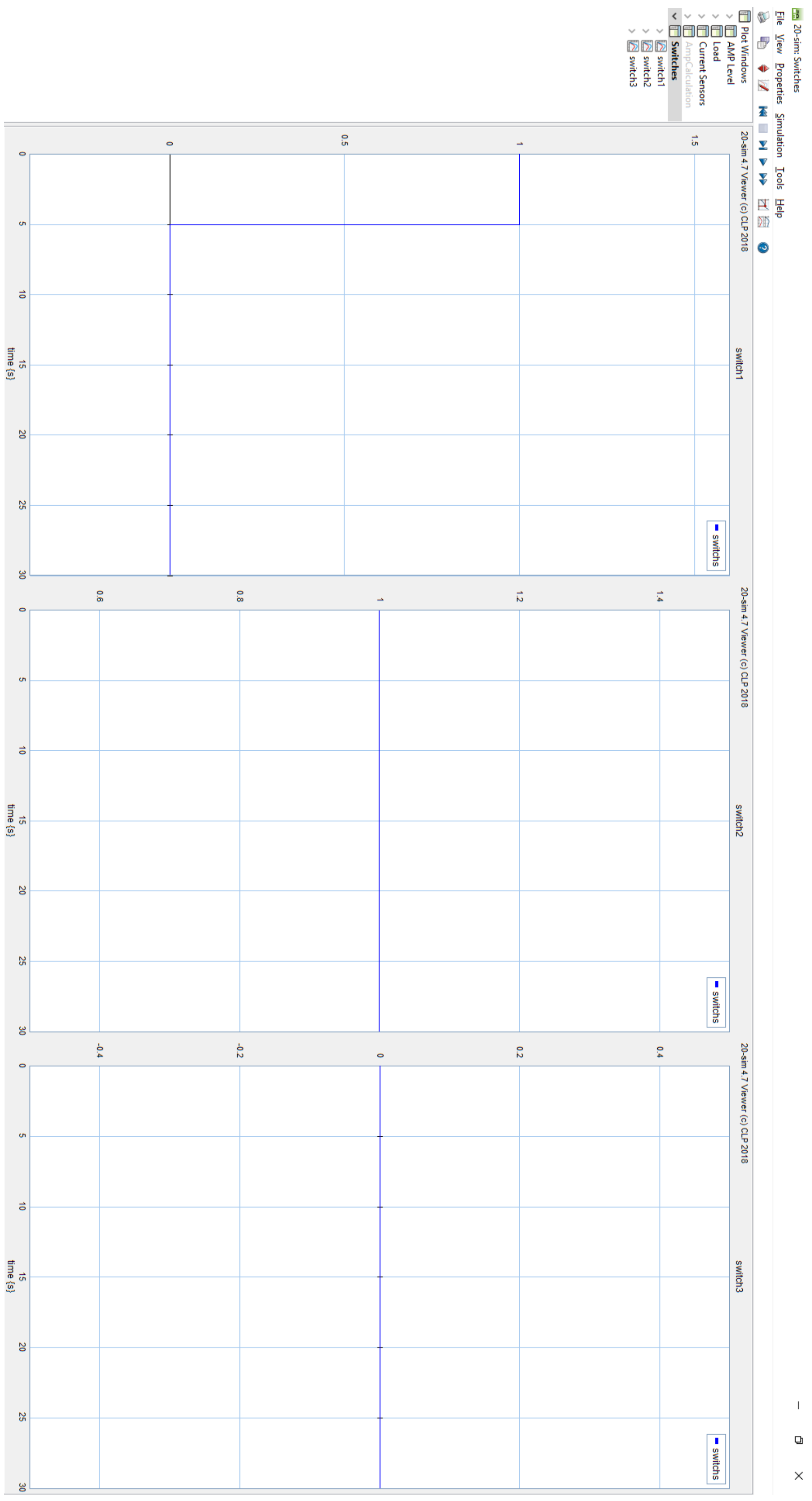


Figure 6.57: Smart grid case study: Switch values in CAUTIOUS mode Scenario

REQ2 The case study should strictly follow the ontology provided in ACPSAF without modification.
FULFILLED - the case study does not add nor remove the ACPS Ontology used in ACPSAF. This has allowed us to notice an area for improvement that will be discussed at the end of this section.

REQ3 The case study should be designed with SysML.
FULFILLED - the case study is designed with SysML 1.2 using Modelio.

REQ4 The case study should be carried out through the MBSE life cycle, from requirements, design, verification, and validation.
PARTIAL - the case study hinted that it has been carried out following the V-model life cycle. It shows that there is a distinction between different phases from the requirement acquisition in Section 6.2.3 to system design in Section 6.2.4 and V&V in Section 6.2.5. However, the process in which the system evolves through the different phases in the life cycle is not well-defined and documented.

REQ5 The case study should use all the provided Perspectives and Viewpoints.
FULFILLED - all ACPSAF Perspectives and Viewpoints are engaged in the case study.

REQ6 The case study should produce at least one View for each Viewpoint.
FULFILLED - at least one View was created for each Viewpoint in ACPSAF.

REQ7 The case study should show how Autonomy Requirements can be identified with ACPSAF.
FULFILLED - this is completed in the Requirement Description View is Section 6.2.3.

REQ8 The case study should use the Co-modelling and Co-simulation approach proposed in ACPSAF to verify and validate ACPS.
FULFILLED - this is carried out in Section 6.2.5.

REQ9 Two case studies should be carried out, demonstrating different types of autonomy.
PARTIAL - the case study demonstrated ACPS with self-adaptive architecture to achieve self-healing. A second case study is needed to fulfil the requirement.

REQ10 Two case studies should be carried out, demonstrating different domains of ACPS.
PARTIAL - the case study demonstrated smart grid ACPS from the energy infrastructure domain. A second case study is needed to fulfil the requirement.

REQ11 Case study should be carried out by a systems engineer.
NOT FULFILLED - this case study was carried out by the author, which invalidates the requirements due to familiarity with the framework.

The evaluation list shows that most of the requirements are fulfilled with some exceptions. The full evaluation of the research will be carried out in Chapter 7. Here, we provide a personal summary on the strengths and weaknesses of ACPSAF that we identify from conducting the case study:

- Identification of Autonomy Requirement. ACPSAF provides a process to capture system Requirements and identify Autonomy Requirements among the list of system Requirements. By identifying Autonomy Requirements, the model can be easily integrated with other ACPS techniques such as the Dependability Cage.
- Assessment of Autonomy Capability. When creating the Autonomy Context View (Figure 6.30), we were not able to fully satisfy the ACPSAF design pattern. ACPSAF states that the sensor in the Autonomy Context View must be connected to the Use Case that is directly related to the state of the environment. In our case, it should have been the power supply of the grid customers (e.g., smart meter). However, our design only allows us to measure the state of the switch sensor. This design is susceptible to false positives and negatives as the system does not have actual data regarding the customer. Although ACPSAF does not have an explicit method to diagnose the problem that could be caused by not following its design pattern, it is able to indicate that there is an issue in the design of the ACPS.
- Need for specialised Use Cases. The Autonomy Context View is relatively complex as it looks at defining three types of Use Cases. The Use Case that is based on the Autonomy Requirements (called the main Use Case), the Use Case that connects the main Use Case to Sensors, and the Use Case that connects the main Use Case to Actuators. They are now all typed as Use Case, but it might be beneficial to refine the generic Use Case into several specialised Use Cases. A special Use Cases type will improve the clarity of the model and help enforce the design pattern.
- Distinction between the architecture of SysML model and domain model. The architecture of the co-model is based on the CT and DE notation, whereas the SysML model is based on a Systems Engineering approach (e.g., ISO/IEC 15288). The ACPSAF recognises the distinction between the two architectures by separating them into two separate architectures, and the co-model needs to be traced back to the SysML model. However, the traceability between these models is relatively weak. It lacks the traceability for behaviour, and the framework does not provide support for a systematic translation process.
- Manual V&V. Currently, ACPS engineers that use ACPSAF have to manually validate the co-simulation results against the test case scenarios. A process or method to automate this validation could be considered in future work.

6.3 Case study: Search and Rescue UAV

This case study is concerned with the application of ACPSAF to create an ACPS model based on a Search and Rescue UAV. Unlike the first case study, this case study focuses on the application of ACPSAF to introduce autonomy to an existing CPS to demonstrate that the ACPSAF can be used to design a new system or upgrade an existing one. This would be relevant, for example, for on-road autonomous vehicles where the system design may start with standardised requirements for on-road vehicles and autonomy is to be added around those constraints. The case study starts by presenting the scope of the case study, which includes the background of the ACPS, requirements, and assumptions in Section 6.3.1. The case study is then presented following the V-model process starting from the requirement acquisition in Section 6.3.2, System design in Section 6.3.3, and V&V in Section 6.3.4. The case study requirement states that the case study should incorporate the external ACPS technique. This case study integrates a technique for automated test case generation in Section 6.3.5. A brief evaluation is then carried out in 6.3.6 to compare the case study in relation to its requirements.

6.3.1 Scope

This section presents the background, requirements, and assumptions of the UAV case study. Unlike the first case study, this case study is built on top of an existing ACPS work. This is to demonstrate that ACPSAF can be used for both new ACPS projects and on top of the existing CPS project. In particular, we look for CPS without an autonomy feature and introduce autonomy with ACPSAF.

6.3.1.1 Background

The background of this case study is relatively short as we are building on an existing CPS project. The case study will be based on an INTO-CPS pilot study called the Swarm of UAVs. The Swarm of UAVs pilot study is documented in the INTO-CPS public report [272]. The co-model of the pilot study is included as part of the INTO-CPS public project examples and can be downloaded through the INTO-CPS application.

The Unmanned Aerial Vehicle (UAV) Swarm pilot study is concerned with a collection of UAVs that communicate in order to achieve some global behaviour. This pilot study reuses the UAV model from another UAV study documented in the same report [272]. In the pilot, each UAV is able to adjust its pitch, yaw, and roll to move in 3D space using rotors. The pilot is used to demonstrate the use of a central controller to dictate the desired movements of the UAVs comprising the swarm.

The UAV swarm pilot study does not have any autonomy features. Communication between UAVs and the central controller are made of values representing coordinates in a 3D Cartesian coordinate system. The central controller should be capable of controlling the movement of the UAVs by sending the coordinates that the UAV should go to. Each UAV has awareness of its current location and will attempt to reach the location given by the central controller. In a real-life scenario, this could represent a control centre sending GPS location to the UAVs. Each UAVs will determine its own current location and fly in a straight-line trajectory towards the target GPS location.

Our interest is to introduce autonomy into the UAV swarm pilot study. There are several types of autonomy features that are commonly found in UAVs, such as collision avoidance and coordinated formations. However, it is best that we start with the application domain instead of autonomy features. For this case study, we consider the use of UAVs as a support to Search and Rescue (SAR) missions.

SAR operations using traditional aerial systems (e.g., aircraft and helicopters) are typically very costly. UAVs can contribute to reducing the resources needed in support of more efficient SAR operations. There are also plenty of resources in the existing literature for SAR UAVs [359, 360, 361, 362]. While there are several literature on novel approaches to supporting the SAR mission with UAVs, the typical scenario of autonomous UAVs in SAR missions is:

1. Path planning process. Rescue team define the search area at the control centre.
2. Calculate optimal flight path to cover the area. This can be achieved by a computer in the control centre or directly in each UAVs depending on level of autonomy.
3. Search process. UAVs go through the flight path while avoiding obstacles.
4. If a survivor is found, notify the control centre. Otherwise, redo search through the area again or return to base depending on implementation.

Most of the steps mentioned earlier can be achieved autonomously, with some exceptions. The first step is not autonomous as it requires a human operator to set the search perimeter. The fourth step involves the UAV identifying a survivor, which is still a challenging task to achieve autonomously. While the advancement of machine learning has enabled computers to identify humans better than ever, most UAV applications for SAR still include the aid of a human operator due to complex scenarios in SAR missions [363, 364]. Implementing a realistic human detection feature would add an unnecessary layer of complexity for the demonstration of ACPSAF. In our case study, we make an assumption that UAVs are able to accurately identify survivors without actually implementing it. With this information, we can now propose requirements for our SAR UAV case study.

6.3.1.2 Requirement

Similar to the first case study, we organise two types of requirements. The first is project requirements that are concerned with the quality of the case study for the purpose of evaluation in this research. The second type of requirement is the system requirement that directly relates to the properties that the SAR UAV must satisfy. Note that the given requirements take inspiration from the characteristics of the SAR scenario without having the intention to develop UAVs for real-life SAR missions. Requirements for realistic SAR missions would include complexities that are not fully considered here.

Project Requirement

- The case study should be engineered using the MBSE technology mentioned in Chapter 3. This means

ACPSAF for architecture framework, SysML for modelling language, and INTO-CPS tool stack (i.e., Modelio, 20-sim, Overture) for modelling tools. Note that we will be using the open source version of Modelio that does not support the use of requirements diagrams. This means that although some views are best captured with requirements diagrams (e.g., traceability view, requirement description view), we could only use alternative diagrams such as BDD.

- The case study should produce at least one view (SysML diagram) for each viewpoint in ACPSAF.
- The case study should demonstrate how ACPSAF can be integrated without of the box verification and validation technique for ACPS. For this case study, we consider the integration of automated test case generation for ACPS.

System Requirement

1. Swarm of UAVs as a base model. The case study should use the Swarm of UAVs model from INTO-CPS pilot study as a starting point and modify it as required.
2. SAR mission. The ACPS scenario shall be based on the deployment of UAVs to look for human survivors within a search area.
3. Multiple UAVs. The case study should use multiple UAVs working together for the SAR mission.
4. Unknown environment. The UAVs should be deployed to an unknown environment. Environment in this context consists of a search area, human survivors, and obstacles. An unknown environment means that the UAVs do not have prior knowledge regarding the search area, location of the survivors, and location of obstacles.
5. Path planning. Each UAV shall be able to plan its own flight path for any given search area. The path planning should attempt to cover everything inside the search area.
6. Search area. UAVs shall be able to navigate through the flight path that it produces from the path planning.
7. Identify survivor. The UAVs shall have the ability to identify human survivors.
8. Collision avoidance. UAVs shall be able to avoid obstacles autonomously. Here, obstacles can be any physical entity that is located within its path of flight.

6.3.1.3 Assumption

1. 2D plane. Although the model is developed for a 3D plane, we will disregard the vertical axis (altitude) for the case study. This would mean that obstacles are always at the same altitude with the UAVs, and it is not possible to achieve collision avoidance by flying over the obstacles.
2. Human detection. We assume that there is a component that we can install in the UAVs to reliably identify human survivors. The component would be a form of camera/radar with a built-in computer to identify human(s) and send a signal to the UAV on-board controller when any human survivor is detected.
3. Simplified obstacle. We make a lot of simplifications to the state of the obstacles. Obstacles in real life

can come in a variety of shapes and sizes, and possibly be mobile. For the case study, we only deal with stationary obstacles, and each obstacle has uniform shapes and sizes. However, it is still possible for multiple obstacles to be located near enough to each other to block the UAVs from passing through between the obstacles.

4. Immobile survivor. We assume that survivors are not mobile and will stay at the same location throughout the mission. Depending on the SAR scenario, this can be a realistic assumption. For example, SAR missions in residential areas after natural disasters are usually more concerned with finding immobile survivors.
5. Battery life. The Swarm of UAVs model comes with a battery model that depletes its energy reserve over the time on flight. We will not be considering the battery level for this case study and assume that the UAVs have constant access to energy.
6. Communication network. We assume that communication can always be established reliably between the control centre and UAVs.
7. Naive control strategy. The control strategy implemented in the case study will be very simple, potentially with strategy issues that will cause the UAVs to fail the requirement, such as collision avoidance under certain special circumstances. This is done so that the case study can demonstrate how automated test case generation can be used to catch these issues. Readers interested in a more realistic UAV control strategy should refer to other literature [63] instead.

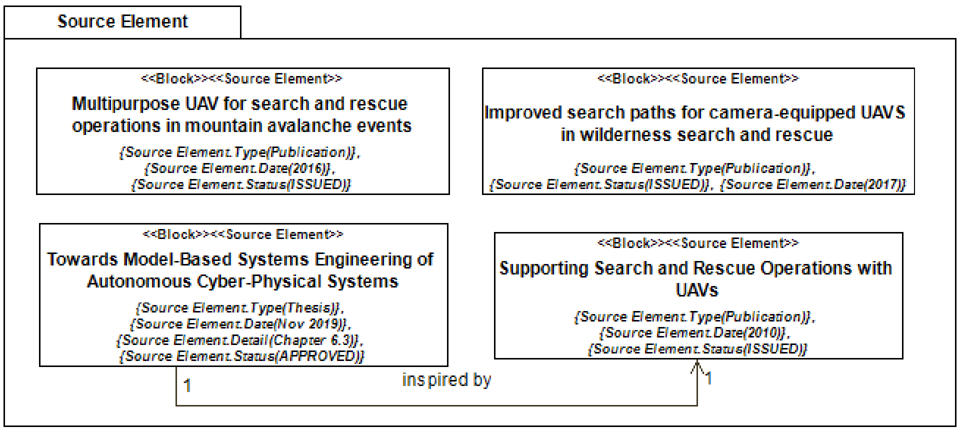
6.3.2 Requirement Acquisition

Similar to the first case study, the V-model life cycle will be used here, albeit quite loosely. The main process would include the requirement, design, implementation, and verification. However, there will be no explicit assessment and process moving from one phase to the other. This section deals with the first phase of the V-model, capturing the system requirements. The viewpoints concerned with the requirements engineering in ACPSAF are encapsulated into the Need Perspective. The Need Perspective allows system Needs to be structured in the form of Need Descriptions, together with its source information and related rules. This section contains the source element view, definition rule set view, requirement description view, context definition view, requirement context view, validation view, and traceability view.

Source Element View

The source element view captures the source of relevant information that will be used to correctly capture the Needs of the project. The Source Elements are especially important for the system designer when the project does not have a detailed system specification, as this source will serve as an inspiration and guide to capture the Needs. Source Element View for the SAR UAV is shown in Figure 6.58. There are four source elements for this project that can be anything that inspires or affects a ‘Need’.

Each Source Element is modelled as a block and is assigned with <<Source Element>> stereotype. Three of the Source Elements come from publications and mainly serve as an inspiration and reference point for the project. There is also a Source Element that refers to this thesis, as the requirements for this case study are recorded here.



Source Element
<<Block>><<Source Element>>
Multipurpose UAV for search and rescue operations in mountain avalanche events
{Source Element.Type(Publication)}, {Source Element.Date(2016)}, {Source Element.Status(ISSUED)}
<<Block>><<Source Element>>
Improved search paths for camera-equipped UAVS in wilderness search and rescue
{Source Element.Type(Publication)}, {Source Element.Status(ISSUED)}, {Source Element.Date(2017)}
<<Block>><<Source Element>>
Towards Model-Based Systems Engineering of Autonomous Cyber-Physical Systems
{Source Element.Type(Thesis)}, {Source Element.Date(Nov 2019)}, {Source Element.Detail(Chapter 6.3)}, {Source Element.Status(APPROVED)}
<<Block>><<Source Element>>
Supporting Search and Rescue Operations with UAVs
{Source Element.Type(Publication)}, {Source Element.Date(2010)}, {Source Element.Status(ISSUED)}
1
inspired by
1

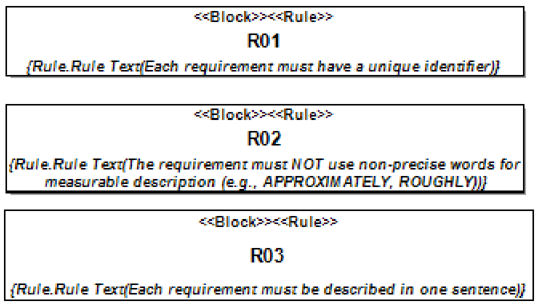
Figure 6.58: SAR UAV case study: Source Element View

There are several tags applied to the Source Elements. The *Type* captures the document types of the Source Elements. *Status* represents the acceptance of the proposed Source Elements, where *ISSUED* would mean that it is under consideration and *APPROVED* means the Source Element is approved and should be considered into the requirement. The tag *detail* allows any extra information to be recorded.

Definition Rule View

The Definition Rule Set View defines Rules that the Need Description must conform to. The rules can be provided in multiple forms, such as natural language or even a mathematical formula. These rules are usually used to ensure the clarity of Need Descriptions, but it can also be beneficial for system designers that are required to follow a certain documentation style.

For the purpose of this project, we do not have a specific requirement on the rules for the Need Description, so we include three simple rules for demonstration. The rules we propose enforce a small degree of clarity by stating that each Need Description must be accompanied with a unique ID, and that each Need Description must be expressed in a sentence. In reality, a sentence can be overly complicated. Therefore, some methods such as the Flesch-Kinkaid Level score could be included in the rule to manage the complexity of the Need Description. We can also rule out the use of specific words such as ‘Approximately’ to reduce ambiguity in the Need Description. The Definition Rule Set View for this case study is shown in Figure 6.59. Note that the Rules defined here are targeted only for the Need Description, so the Rule definition itself does not have to conform to the Rules.

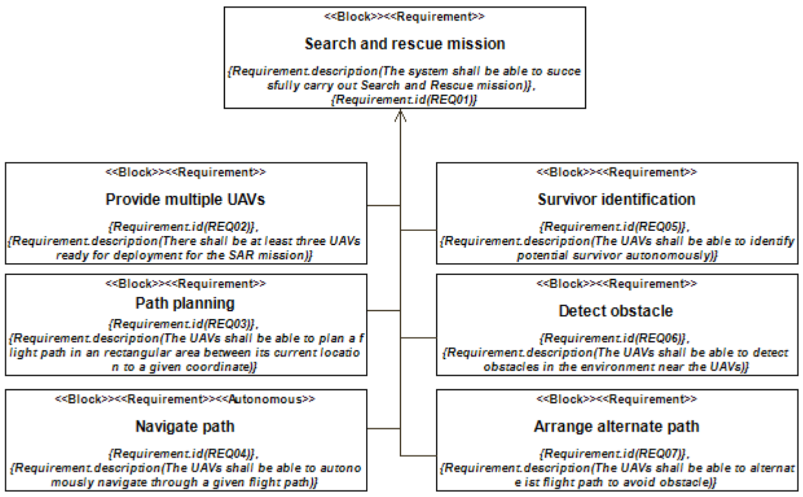


<<Block>><<Rule>>
R01
{Rule.Rule Text(Each requirement must have a unique identifier)}
<<Block>><<Rule>>
R02
{Rule.Rule Text(The requirement must NOT use non-precise words for measurable description (e.g., APPROXIMATELY, ROUGHLY))}
<<Block>><<Rule>>
R03
{Rule.Rule Text(Each requirement must be described in one sentence)}

Figure 6.59: SAR UAV case study: Definition Rule Set View

Requirement Description View

The Requirement Description View captures structured descriptions of each Need Description. The Requirement Description View for this project is given in Figure 6.60. The Need Description is realised here as a SysML Block, identified with the <<requirement>> stereotype.



<<Block>><<Requirement>>
Search and rescue mission
{Requirement.description(The system shall be able to succesfully carry out Search and Rescue mission)}, {Requirement.id(REQ01)}
<<Block>><<Requirement>>
Provide multiple UAVs
{Requirement.id(REQ02)}, {Requirement.description(There shall be at least three UAVs ready for deployment for the SAR mission)}
<<Block>><<Requirement>>
Survivor identification
{Requirement.id(REQ05)}, {Requirement.description(The UAVs shall be able to identify potential survivor autonomously)}
<<Block>><<Requirement>>
Path planning
{Requirement.id(REQ03)}, {Requirement.description(The UAVs shall be able to plan a flight path in an rectangular area between its current location to a given coordinate)}
<<Block>><<Requirement>>
Detect obstacle
{Requirement.id(REQ06)}, {Requirement.description(The UAVs shall be able to detect obstacles in the environment near the UAVs)}
<<Block>><<Requirement>><<Autonomous>>
Navigate path
{Requirement.id(REQ04)}, {Requirement.description(The UAVs shall be able to autonomously navigate through a given flight path)}
<<Block>><<Requirement>>
Arrange alternate path
{Requirement.id(REQ07)}, {Requirement.description(The UAVs shall be able to alternate ist flight path to avoid obstacle)}

Figure 6.60: SAR UAV case study: Requirement Description View

The requirements shown here are all the new features to be added into the existing UAV swarm model. Typically, it is a good idea to include other requirements that relate to the basic performances of the UAVs as well. However, we make the assumption and only explore the scenarios where UAVs perform within its known and well-characterised flight characteristics. For example, we will not explore anomaly cases, such as when one of the flight rotors malfunctions and observe whether the UAV will be able to adjust its flight control to compensate for the loss of balance to continue its operation.

functionalities (e.g., moving the rotors) will work perfectly and do not require testing.

Context Definition View

The Source Element View captures the source of relevant information that will be used to correctly capture the Needs of the project. The Context Definition View identifies the Contexts that will be explored in the Requirement Context View. These Contexts may take many forms, such as the Stakeholder Roles and System Elements, as suggested in ACPSAF. The Stakeholder Role in this case study is relatively simple, only involving the control centre and survivors (Figure 6.62). For the System Elements, we introduce a special type of UAV to explore UAV swarms with heterogeneous UAVs. Figure 6.62 shows that the System context involves a control centre, Search UAV, and Rescue UAV. The difference between the two types of UAVs is not captured here as the structure and behaviour of system elements are covered in the CPS Perspective.

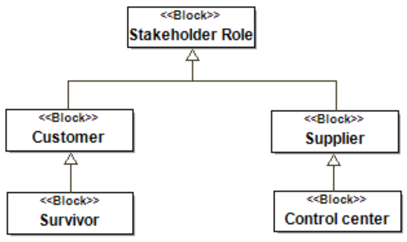


Figure 6.61: SAR UAV case study: Context Definition View for Stakeholder Role

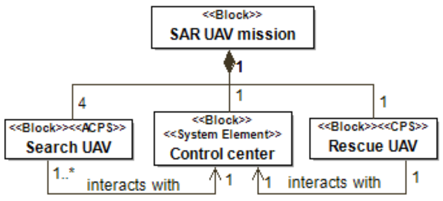


Figure 6.62: SAR UAV case study: Context Definition View for System Element

Requirement Context View

The Requirement Context View gives Needs meaning by viewing it from a particular Context and is realised using a Use Case diagram. Each Context identified in the Context Definition View could potentially have their own Requirement Context View. However, the minimum would be to provide sufficient Requirement Context View to cover all the requirements stated in the Requirement Description View.

For this case study, two Use Case diagrams can be used to capture the Requirement Context View. The first diagram is taken from the context from the Control Centre and is shown in Figure 6.63. It shows that the Use Cases that the Control Center need to fulfil for SAR missions would be to send the search area to the search UAVs, receive the survivor's location from the search UAVs, and send rescue UAVs to the location of the survivor.

Developing a search UAV is easier when compared to a Rescue UAV. A Rescue UAV must be able to pick and fly rescue targets to safety, requiring a special type of UAV to act like a passenger aircraft. The risk associated with the operation also makes it a much more regulated one. Note that in our requirement context, we stop at sending rescue UAVs to a survivor instead of the safe delivery of the rescue target. This is a simplification to the case study, as the difficulties in the rescue journey (e.g., safety) are mostly surrounding non-autonomous challenges that are not the focus of the case study.

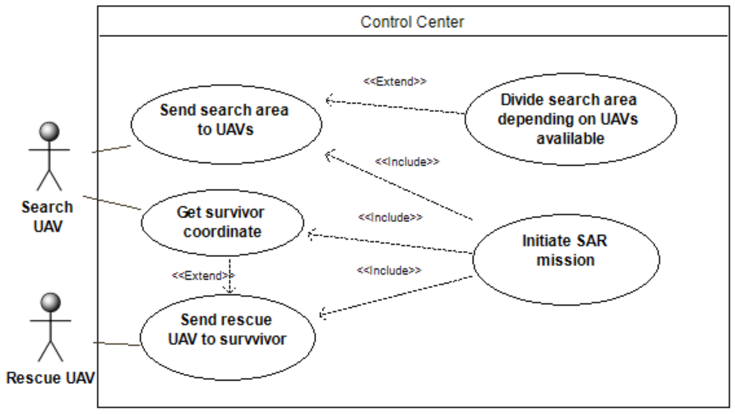


Figure 6.63: SAR UAV case study: Requirement Context View for Control Centre Context

Figure 6.64 shows the Requirement Context View from two contexts. A Use Case diagram typically only contains one system boundary to keep the diagram from being cluttered. However, it is certainly possible to include more than just one Context in the diagram. The Context of *Search UAV* and *Rescue UAV* are both considered in Figure 6.64. The Use Case for *Rescue UAV* is very straightforward as it is only required to receive the location of the survivor and navigate to the survivor. This is a simple Use Case as the base model for the UAV swarm already supports the feature to control UAVs to navigate to a GPS coordinate. On the other side, the Search UAV context contains more Use Cases that have to be fulfilled to meet the requirements.

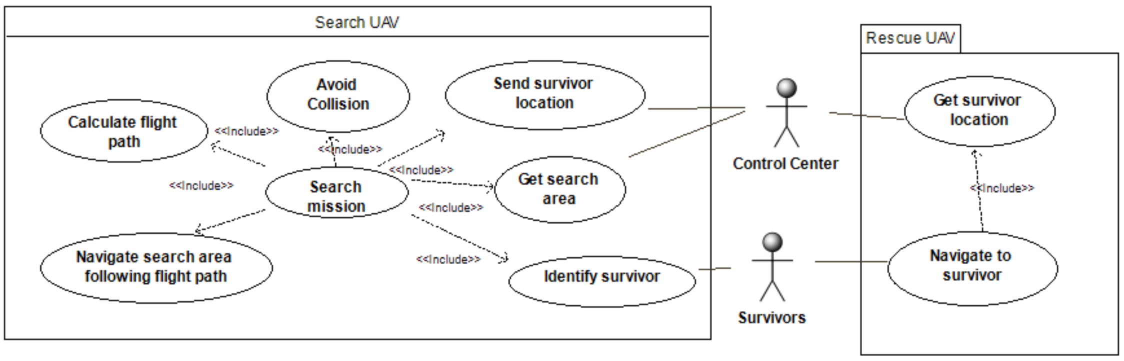


Figure 6.64: SAR UAV case study: Requirement Context View for UAVs Context

Traceability View

The Traceability View shows traceability both to and from Need Descriptions, which are often realised using requirements diagrams as Need Descriptions are realised as requirements in SysML. However, as the requirements diagram is not supported in the SysML tools that we use (Modelio open source), BDD is used to capture the Traceability View. The traceability between the Use Cases defined in the Requirement Context View and Need Description defined in the Requirement Description View is shown in Figure 6.65.

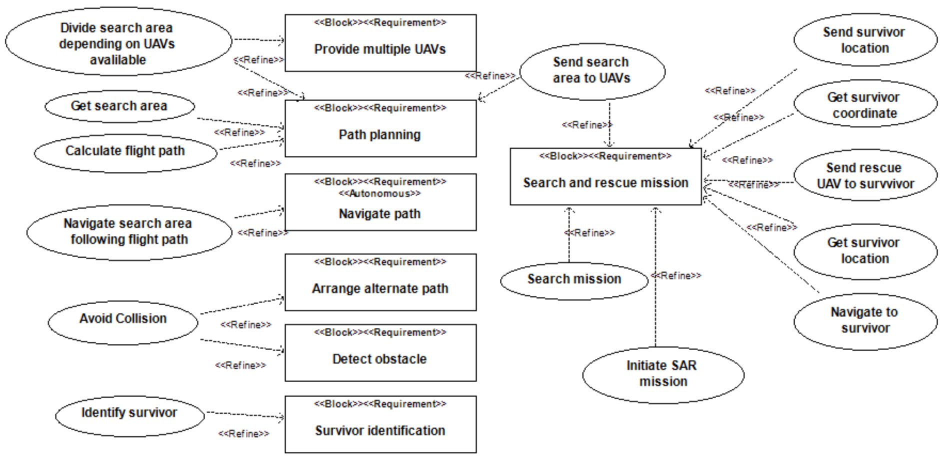


Figure 6.65: SAR UAV case study: Traceability View between Use Cases and Need Descriptions

Figure 6.65 shows that the Need Description in the project is traceable to at least one Use Case. It is common to see that one Need Description is traceable to multiple Use Cases and vice versa. However, the Need Description *Search and rescue mission* is being traced to a high number of Use Cases. This indicates that the project could benefit from refining *Search and rescue mission* into several smaller Need Descriptions. In particular, as there are several Use Cases addressing interaction with the survivor, *Search and rescue mission* can be refined to describe better requirements in relation to survivors. It is also possible that the Use Cases are addressing issues that are not really required from the Need Description. In this case, the Use Cases should be removed instead.

6.3.3 System Design

The next step in the V-model process is system design and development. The system design process can be split into multiple stages, working from higher-level specification down into lower level implementation. However, instead of presenting the model following the chronological process, it is split into sections that focus on the structural or behavioural aspect of the system. This structure allows us to present the case study better as the autonomy behaviour is the focus of the research.

6.3.3.1 System Structure

This section covers the architecture of the SAR UAV, focusing on the structural design of the system. The View related to this section is CPS Identification View, CPS Structure View, CPS Definition View, and CPS Configuration View.

CPS Identification View

The CPS Identification View identifies Systems and the relationships between them. It is important to note that because a System can be decomposed into multiple subsystems, it is not uncommon that multiple CPS Identification View(s) are produced to show the System Elements making up a System. We demonstrate this by showing the SAR UAV system with two SysML BDD. Figure 6.66 shows the CPS Identification View from the top level system, which is identical to the one shown in Context Definition Viewpoint as the System Context. The diagram shows that Search UAV is an ACPS identified by the <<ACPS>> stereotype, while Rescue UAV is a CPS without autonomy feature as shown by the <<CPS>> stereotype. The decision to categorise Rescue UAV as CPS here is arguable, as a UAV with the ability to travel to a given GPS coordinate is considered autonomy. The only reason we identify it as CPS is simple because there is no autonomy requirement defined in the Need Description that applies to the Rescue UAV. All the new features for Autonomy Requirements are directed for Search UAVs.

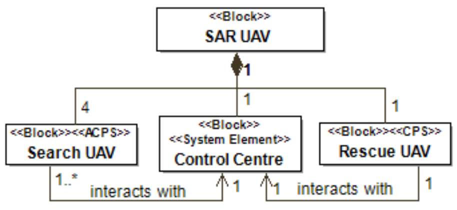


Figure 6.66: SAR UAV case study: CPS Identification View for SAR UAV

Focusing on Search UAV, Figure 6.67 shows that more System Elements can be identified. For this case, it is actually possible to combine both diagrams into one as the model is still relatively small. The Rescue UAV shares the same structure as the Search UAV, so only one diagram is shown here.

CPS Structure View

The CPS Structure View is used to define the structure of ACPS, showing its System Elements composition and the relationship between them, its System Properties, and System Functions. Figure 6.68 shows that the *Control Centre* interacts with one or more *Search UAVs* and *Rescue UAVs*. The *Control Centre* also provides an interface that will be defined in the Interface Definition View.

Figure 6.69 shows the CPS Structure View for both Search UAV and Rescue UAV as they both share the same structure. It shows that the *Navigation subsystem* is further decomposed into *GPS* and *Lidar*, which serves as

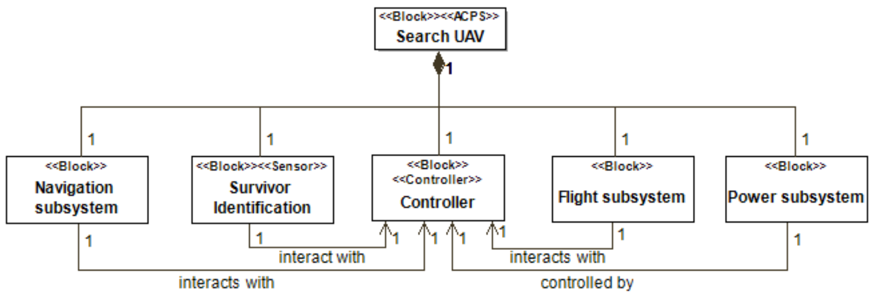


Figure 6.67: SAR UAV case study: CPS Identification View for Search UAV and Rescue UAV

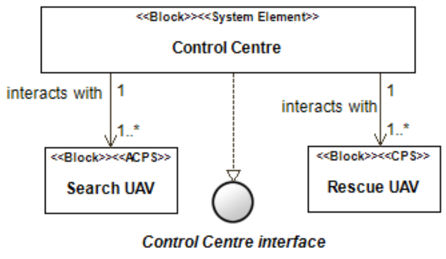


Figure 6.68: SAR UAV case study: CPS Structure View for Control Centre

the UAV sensors. The *Flight Subsystem* contains the UAV actuators that are made of *Rotors*. The UAV also has *Survivor Identification*, which is a special Sensor used to detect humans. As mentioned in the case study assumption, we assume that there is a component that can reliably detect humans near the UAVs that is ready for use. The controller also provides an interface called the *UAV interface* that will be defined next.

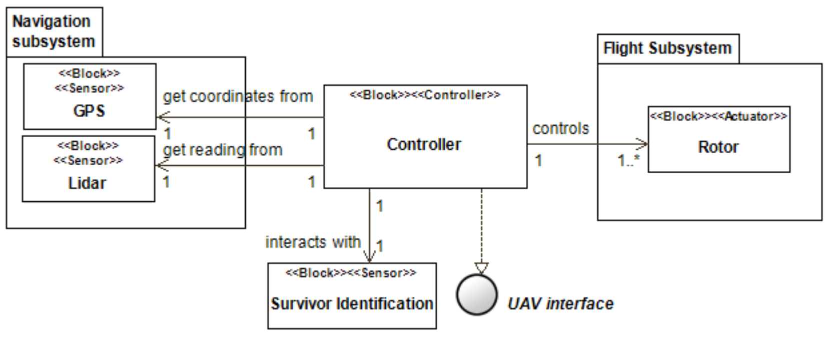
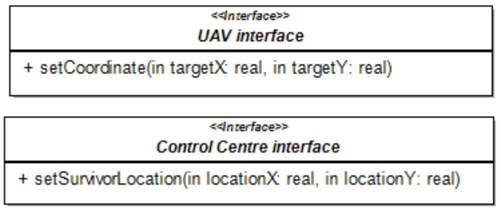


Figure 6.69: SAR UAV case study: CPS Structure View for Search UAV and Rescue UAV

Interface Definition View

The Interface Definition View defines the Interface between Systems and between System Elements within a System. The Interfaces used in the System and System Elements of the SAR UAV case study are shown in Figure 6.70. *UAV interface* defines a simple System Function that takes in two real numbers as an input. As shown in CPS Structure View, this interface is implemented by the UAV as a way for the Control Centre to send the search area coordinates. The *Control Centre interface* is quite similar as it takes in the same input. However, this Interface is implemented by the *Control Centre* to allow UAVs to communicate the location of survivors found by the UAVs.



<<Interface>>
UAV interface
+ setCoordinate(in targetX: real, in targetY: real)
<<Interface>>
Control Centre interface
+ setSurvivorLocation(in locationX: real, in locationY: real)

Figure 6.70: SAR UAV case study: Interface Definition View

CPS Configuration View

The CPS Configuration View shows how a System is configured based on its Structure, together with the Interfaces and Item Flow between System Elements. Figure 6.71 shows the configuration between *Control Centre*, *Rescue UAV*, and *Search UAV*. The configuration shows that four *Search UAVs* and one *Rescue UAV* are used in the SAR UAV mission. For our case study, we only consider this configuration for the SAR mission. If a user wants to explore several configurations, several CPS Configuration Views would be created. Figure 6.71 shows that each UAV communicates with the *Control Centre* through the *UAV interface* and *Control Centre interface*.

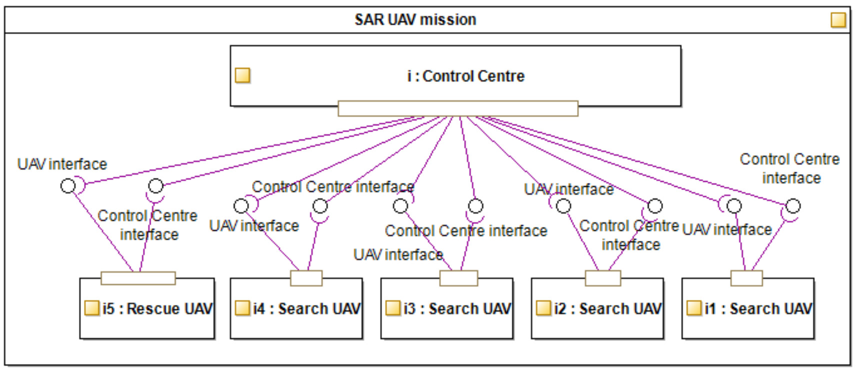


Figure 6.71: SAR UAV case study: CPS Configuration View for SAR UAV

Figure 6.72 shows the internal configuration of the *Search UAV*. Similar to previous Views, the CPS Configuration View for Rescue UAV is not shown here as the Rescue UAV shares the same configuration as Search UAV. Most of the configurations here are quite expected as they closely resemble the CPS Structure View. However, this configuration shows that the Search UAV is a quadcopter that is characterised by the four *Rotors*.

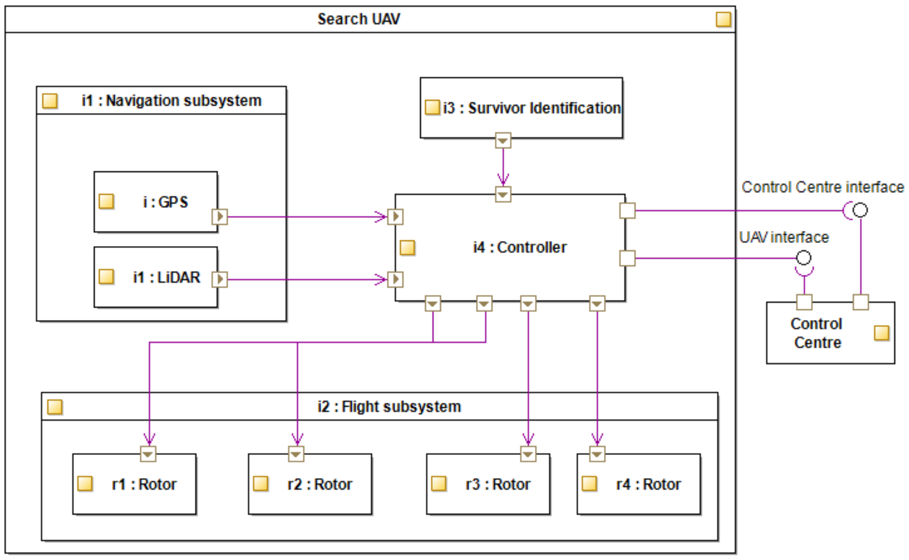
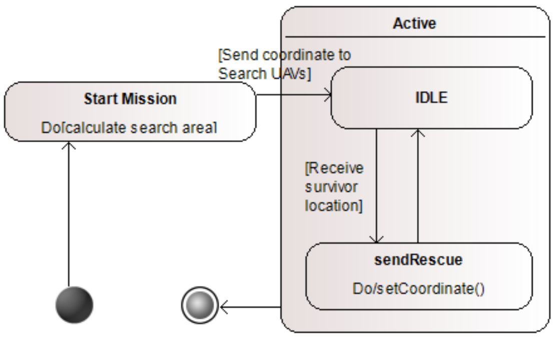


Figure 6.72: SAR UAV case study: CPS Configuration View for Search UAV

6.3.3.2 Non-autonomous behaviour

This case study does not add a lot into the non-autonomous behaviour of the project from the base Swarm of UAVs model. As shown in the System Identification View, the SAR UAV case study involves the Control Centre, Search UAV, and Rescue UAV. There are no new behaviours added into the Rescue UAV as the ability to take and navigate to a GPS coordinate (without collision avoidance) is already part of the base model. New behaviour is required for the Search UAV to fulfil the requirements, but these are all autonomous behaviours. Non-autonomous behaviour is only introduced to the control centre, and we capture this as CPS Behaviour View with the State Machine Diagram as shown in Figure 6.73. The diagram shows that the control centre would start the mission by calculating the search area for the Search UAVs and get into an IDLE mode after sending the coordinates of the search area. It will stay IDLE until it receives information regarding the location of the survivor from the Search UAV. It will then send this information to the Rescue UAV and return to the IDLE state.



Active
[Send coordinate to Search UAVs]
Start Mission
Do/calculate search area/
IDLE
[Receive survivor location]
sendRescue
Do/setCoordinate()

Figure 6.73: SAR UAV case study: CPS Behaviour View for Control Centre

6.3.3.3 Autonomous behaviour

Methods to architecting autonomy related behaviour in ACPSAF are presented in the Autonomy Perspective. During requirement acquisition, we have identified an autonomy requirement (refer to Figure 6.60) indicated as ‘Navigate path’. Although several other Requirements such as ‘Path planning’ are not identified as an Autonomy Requirement, they are all required to achieve the overall SAR operation autonomously.

Autonomous Structure View

The autonomous structure view provides a structural overview of the Sensor, Controller, and Actuator that an ACPS needs to fulfil an Autonomy Requirement. In this case, we need to show the System Elements needed for ‘Navigate path’. However, ‘Navigate path’ is affected by the requirements ‘Path planning’ and ‘Navigate path’ follow the path that results from ‘Path planning’. Other requirements such as ‘Detect obstacle’, ‘Survivor identification’, and ‘Arrange alternate path’ will affect the ‘Path planning’, so these requirements are interconnected to deliver the Autonomy Requirement. The Autonomous Structure View (Figure 6.74) for the Search UAV is actually very similar to the System Structure View as we only include System Elements needed for the Autonomy Requirement.

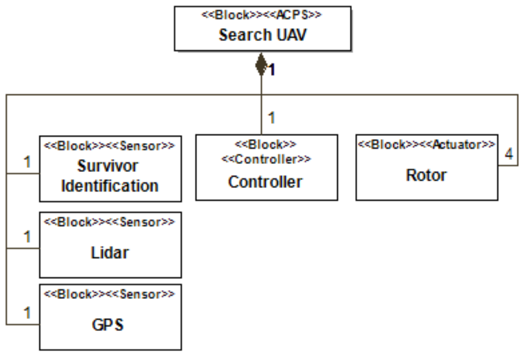


Figure 6.74: SAR UAV case study: Autonomous Structure View for Search UAV

Autonomous Context View

The Autonomy Context View defines the Use Cases that the *Controller* has to satisfy in order to fulfil the Autonomy Requirement. The Autonomy Context View for this case study is shown in Figure 6.75. Autonomy Context View is defined with the Use Case Diagram, with the controller as the system boundary. The *Sensors* and *Actuators* identified in the Autonomy Structure View are included as actors. The first Use Case could be directly derived from the Autonomy Requirement. In this case, it is the ‘Navigate through flight path’ that includes ‘Calculate flight path’. The ‘Calculate flight path’ considers several aspects to come up with the flight path. First, it has to consider how to cover the search area relative to its current location. It also needs to avoid collision, so it takes into consideration the location of any observable obstacle and modifies the flight path accordingly. It would also change its flight path if a survivor is detected.

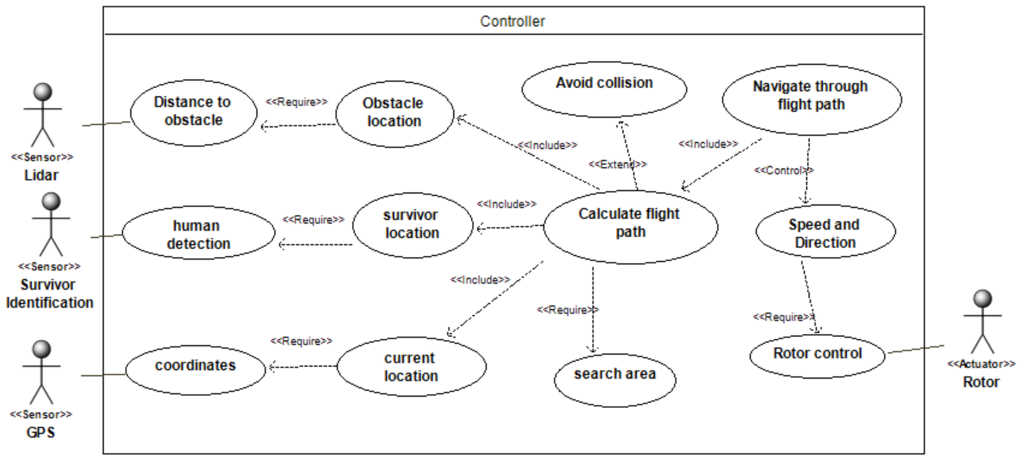


Figure 6.75: SAR UAV case study: Autonomous Context View

Autonomous Context View states that the Use Cases refined from the Autonomy Requirement have to be connected to Sensors and Actuators identified in the Autonomy Structure View. To help with this process, we can start by creating Use Cases of the measurement that the Sensors provide. For example, GPS would give measurement in terms of coordinates, so the Use Case *coordinates* is defined and connected to *GPS*. For the Actuator, it is easier to work from Use Case to the Actuators. We start from the Use Case 'Navigate through flight path' and ask about what has to be controlled to achieve this Use Case. For UAV navigation, this would be the 'Speed and Direction' of the UAV. From here, it should be easy to define the Use Case that allows the 'Speed and Direction' of the UAV to be influenced. In this case, it is the 'Rotor control' provided by the actuator 'Rotor'. The rest of the Use Cases, such as the 'Calculate flight path' and 'Avoid collision', is the result of refinement from the main Use Case in relation to the Requirement.

Autonomy Behaviour View

The Autonomy Behaviour View is treated the same way as the CPS Behaviour View. All types of SysML behaviour diagrams and parametric diagrams can be used to capture the behaviour of the Use Cases described in the Autonomy Context View. Figure 6.76 shows the behaviour of the *Search UAV* with the SysML Activity Diagram. The behaviours shown are relatively high level, but the behaviour here shows that all the Use Cases from the Autonomy Context View are considered.

6.3.4 Verification and Validation

ACPSAF focuses on verification and validation using co-modelling and simulation techniques. However, ACPSAF does not have a process to systematically translate SysML models into the domain model. ACPS engineers have to use their expertise to create the domain model from the SysML model. At this point, we

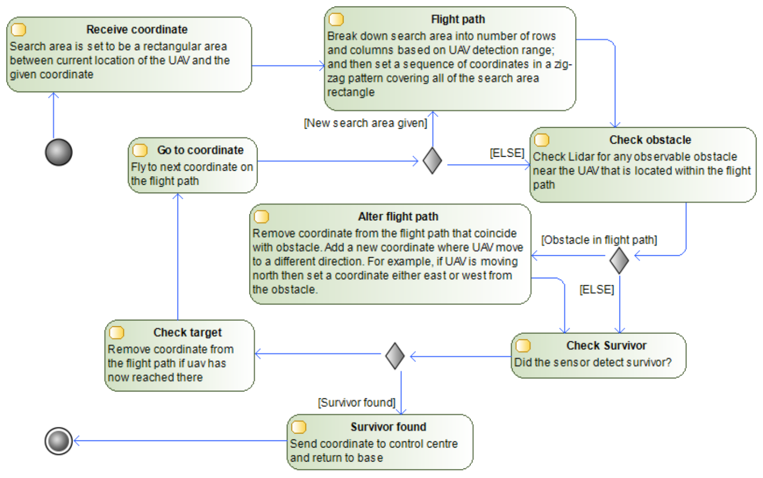

Receive coordinate
Search area is set to be a rectangular area between current location of the UAV and the given coordinate
Flight path
Break down search area into number of rows and columns based on UAV detection range; and then set a sequence of coordinates in a zig-zag pattern covering all of the search area rectangle
[New search area given]
Go to coordinate
Fly to next coordinate on the flight path
[ELSE]
Check obstacle
Check Lidar for any observable obstacle near the UAV that is located within the flight path
Alter flight path
Remove coordinate from the flight path that coincide with obstacle. Add a new coordinate where UAV move to a different direction. For example, if UAV is moving north then set a coordinate either east or west from the obstacle.
[Obstacle in flight path]
[ELSE]
Check target
Remove coordinate from the flight path if uav has now reached there
Check Survivor
Did the sensor detect survivor?
[Survivor found]
Survivor found
Send coordinate to control centre and return to base

Figure 6.76: SAR UAV case study: Autonomous Behaviour View

assume that the system model from the previous section has been translated into a domain model. This model will not be included here but can be found in Appendix C. The 20-sim model has not been modified from the original Swarm of UAVs pilot study. The autonomy behaviour has been introduced entirely on the VDM-RT model, which is the DE model of the co-simulation. The VDM-RT model for the Control Centre is shown in Appendix Section C.1, Search UAV in Section C.3, and Rescue UAV in Section C.2.

ACPSAF provides Viewpoints to describe the architecture of the domain model and state that there must be traceability to the SysML model. The Architecture Structure View shows the structure of the domain model with tags specifying whether it is a DE or CT model and the platform in which it is developed. The Architecture Structure View also shows the connection port of each model where data will be exchanged during co-simulation. Figure 6.77 shows the Architecture Structure View for this case study.

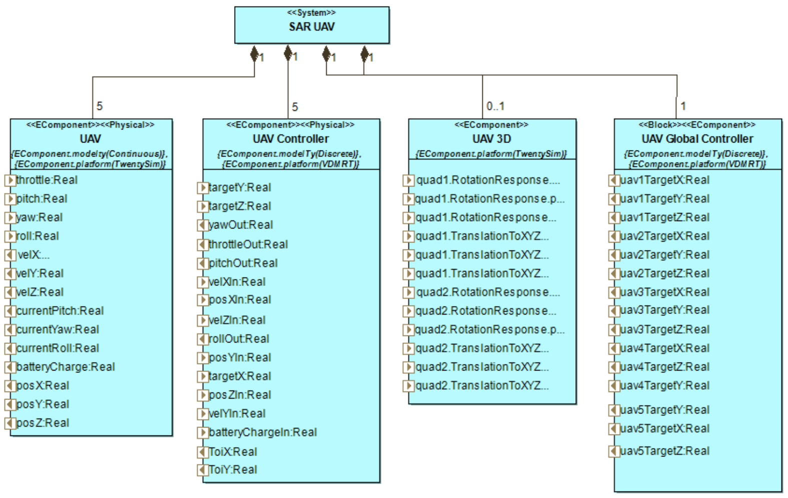


Figure 6.77: SAR UAV case study: Architecture Structure View

The exact instances created for each of the models shown in the Architecture Structure View as well as the connection between them are shown in Figure 6.78 as a Connection View. Due to the large number of ports and connectors in this pilot, it is not appropriate to represent them in a single diagram. As such, the Connection View only shows the connections with one UAV at a time. Replicating the diagram from Figure 6.37 with the number of UAV instances (four more times) will reproduce the complete configuration.

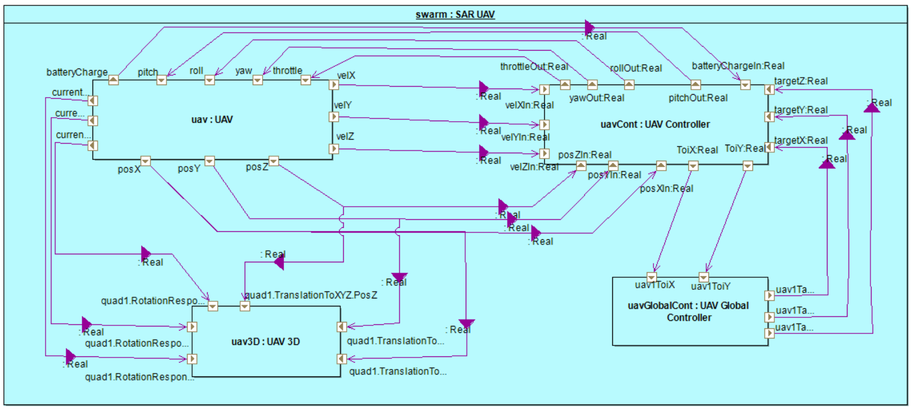


Figure 6.78: SAR UAV case study: Connection View

To ensure the domain model reflects those of the SysML model, each of the Comodel blocks should be traceable to one or more System Elements making up the ACPS. For complete traceability, all System Elements should be covered in the System Traceability View and should have a Comodel block traced into it. Figure 6.79 shows the System Traceability View for this case study.

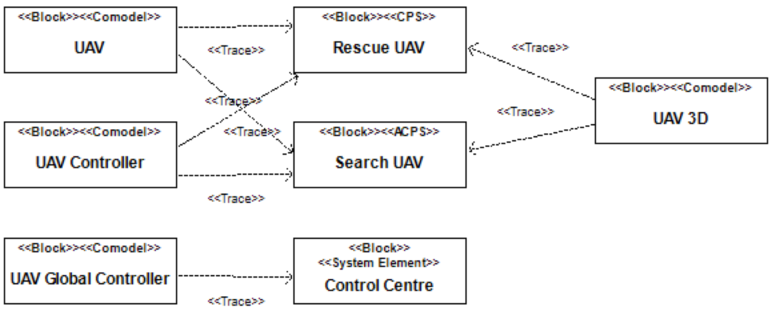


Figure 6.79: SAR UAV case study: System Traceability View

With the co-model ready, the FMUs can be imported to the INTO-CPS application for co-simulation. The first test would be a simple scenario where the Control Centre starts a SAR mission by sending each Search UAV a specific coordinate that indicates its respective search area. The Validation View showing this Scenario with the SysML Sequence Diagram is shown in Figure 6.80.

The UAVs by default start at position 5 on both x and y axes. For this scenario, we set the search area to be a square of 15 by 15 towards all directions from the starting point. This means that the search area would

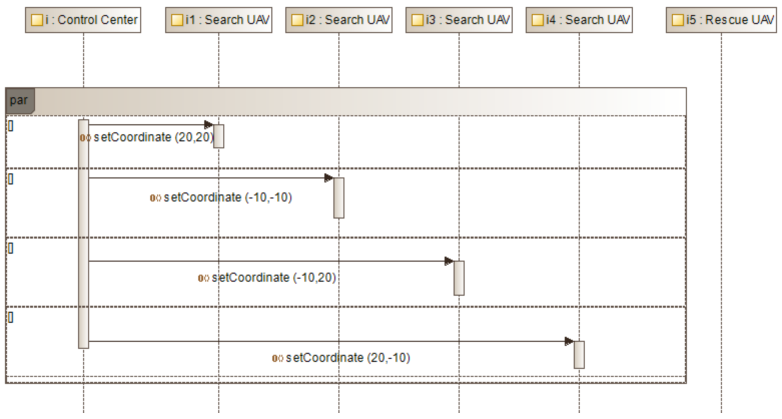


Figure 6.80: SAR UAV case study: Validation View of startMission Scenario

be a square from <-10,-10> to <20,20>. The Scenario does not set any obstacles and survivors on the map, so the results should be that the UAVs simply navigate around the search area. In practice, a full 360 degree coverage may not always be the preferred search path. For example, if the UAVs are dispatched from the side or corner of the search regions (rather than the centre), then the search region would no longer be 360 degrees from the starting point. These types of tactical planning would fall into the responsibility of the control station.

The result of the initial co-simulation for 360 coverage is shown in Figure 6.81. The graph shown in Figure 6.81 is not generated by the INTO-CPS application. Although the INTO-CPS application has the functionality to generate a live graph from the co-simulation result, the x-axis of the graph is fixed on the simulation time. However, the best way to show the navigation of the UAVs over the 2D plane is to set the axis on the graph as shown in Figure 6.81.

The graph was generated using an external tool using the co-simulation result generated by INTO-CPS in CSV format. The graph shows that the UAVs navigate through their search area in a specific pattern. The gaps between each row traversed by the UAVs are dependent on the effective range of the sensor responsible for detecting the survivors. The graph also shows that the UAVs are not turning at a perfect 90 degree due to its momentum. A smarter controller can be designed to mitigate this. However, this is not a concern for this case study. For the next test, we introduce some obstacles in the environment to validate the collision avoidance behaviour.

In the co-model, the obstacles are implemented in the DE-model instead of the CT-model. The main reason for this decision is that the modification process in CT-model to represent changes in the environment is difficult to model and tedious to modify. We encounter similar situations in the first case study where Loads are used to generate faults in the smart grid. To simulate different faults occurring at different places, the Load has to be physically moved in the model, the FMUs re-compiled and imported again to the INTO-CPS

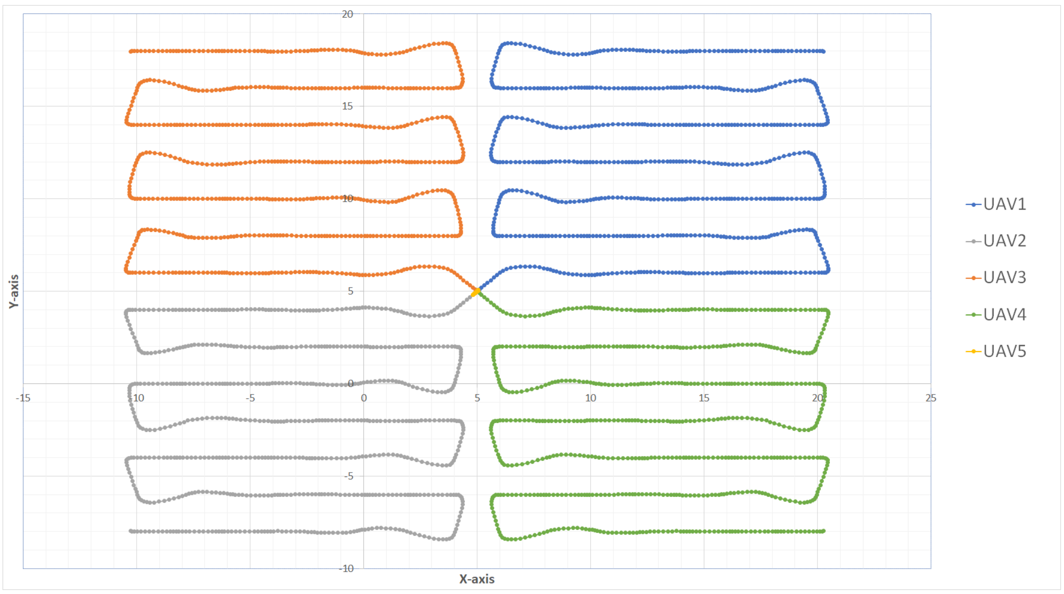


Figure 6.81: SAR UAV case study: INTO-CPS co-simulation result for startMission Scenario

app for co-simulation. While this is an acceptable process for several scenarios, it is not a scalable method. So for this case study, the environment is in the DE-model together in the UAV model. The weakness of this approach is that we do not actually implement and test the functionality of the Lidar in detecting the obstacle. However, this is acceptable as our research is concerned about the decision-making process of ACPS and not the reliability of sensors used. Figure 6.82 shows the simulation result for obstacles located in <0,2>, <10,6> and <12,14>. Note that while the flight path in this scenario is very close to the ideal path, we will cover a more complex set of obstacles that could potentially disrupt the operation in the next section.

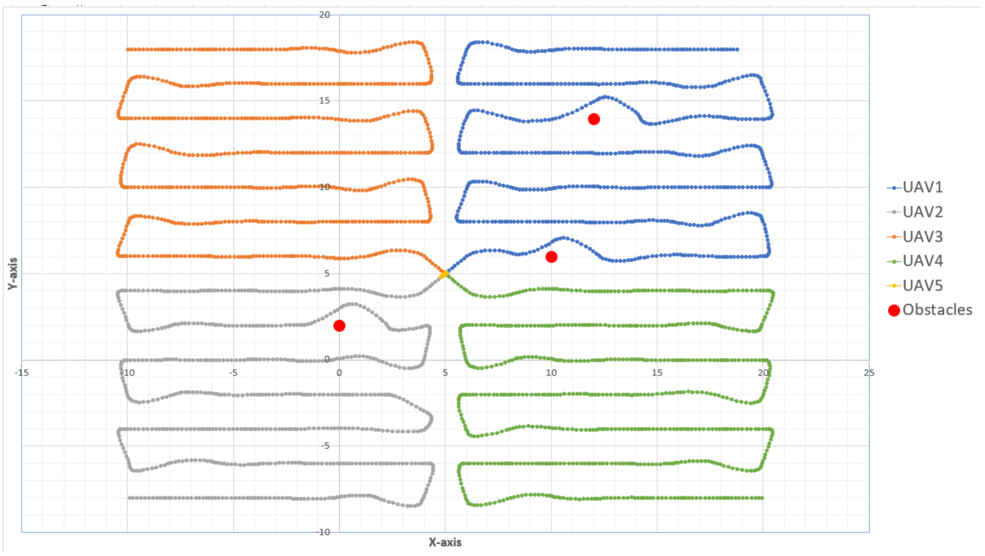


Figure 6.82: SAR UAV case study: startMission Scenario with collision avoidance

The next test case would be the full SAR Scenario with a survivor in the search area. The beginning of the

SAR Scenario will be the same as the *startMission* Scenario where the Control Centre sends GPS coordinates to the Search UAVs. However, in this Scenario there is a survivor in the area that *UAV1* is deployed in. When *UAV1* reaches the survivor, it should communicate its finding to the Control Centre. The Control Centre would then pass this coordinate to *UAV5*, which is the Rescue UAV that would go to the survivor. This Scenario is captured in SysML with the Sequence Diagram shown in Figure 6.83. The co-simulation result for this scenario is shown in Figure 6.84.

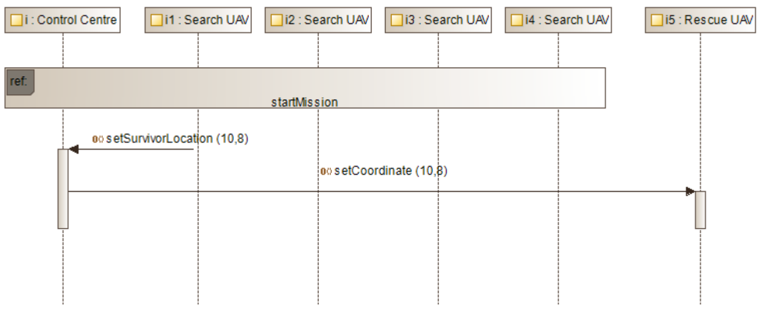


Figure 6.83: SAR UAV case study: SAR Scenario

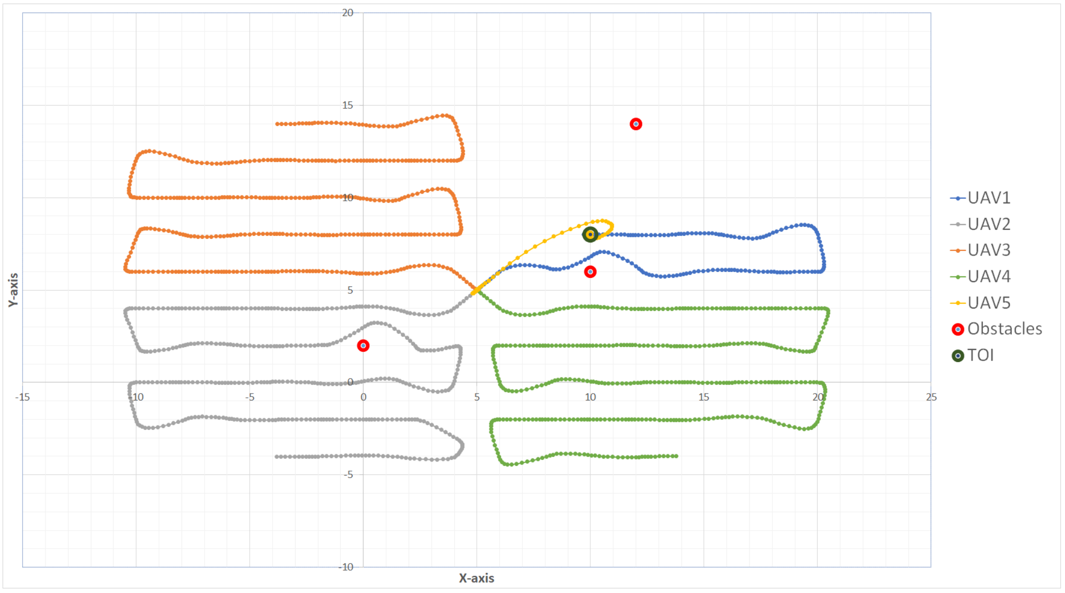


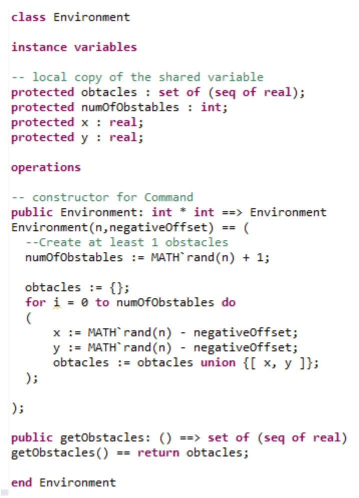
Figure 6.84: SAR UAV case study: INTO-CPS co-simulation result for SAR Scenario

6.3.5 Automated Test Case Generation

In this section, we explore the use of automated test case generation in the co-simulation of the SAR UAV case study. Automated test case generation for an autonomous system is not a new concept as there is literature on this topic found from over a decade ago [198]. However, due to the rise of popularity in some autonomous systems such as the autonomous car and UAVs, the use of automated test case generation as a means to evaluate autonomous system is gaining more attention in the recent years [19, 273, 200]. The main challenge in automated test case generation is not simply about how to achieve it, but rather about how interesting scenarios can be generated. While there are techniques that can be used to automatically generate generic test cases [365], most of the recent literature in automated test case generation for autonomous systems are concerned in finding the method to generate meaningful scenarios for a particular domain. For example, the work presented by Althoff and Lutz (2018) [200] looks at a constructive algorithm where scenarios generated are optimised towards certain characteristics for testing collision avoidance of road vehicles.

Developing a domain-specific algorithm for the automated generation of meaningful test case scenarios is outside the scope of this case study. However, we could implement a simple test case generation method as a way to demonstrate how this can be achieved in the setting of our case study. One of the simplest algorithms for the test case generation is the Random Test Data Generation [366]. As suggested by its name, Random Test Data Generation chooses inputs arbitrarily until hopefully, a useful input is found. It is a fast way to generate a large number of test cases, but may fail to contribute significant evaluation if test requirements are not incorporated. However, this is sufficient for our case study as we are only interested in demonstrating the feasibility of test case generation and not actually concerned with using the technique to improve our model.

An important aspect to automated test case generation is choosing the type of parameter to be generated. As automated test case generation in autonomous systems is typically aimed at modelling the unknown environment, the choice would be in the generation of obstacles and/or survivors. For our case study, we choose the obstacles as there is an opportunity to observe the performance of the collision avoidance and the possibility of whether the UAV could still find survivors trapped in between obstacles. As we already have the environment model for obstacles in the VDM-RT code, all we need to do is implement the Random Test Data Generation algorithm in the environment class that stores the coordinates of the obstacles. The VDM-RT snipped code for this test case generation is shown in Figure 6.85.



```
class Environment

instance variables

-- local copy of the shared variable
protected obtacles : set of (seq of real);
protected numOfObstables : int;
protected x : real;
protected y : real;

operations

-- constructor for Command
public Environment: int * int ==> Environment
Environment(n,negativeOffset) == (
  --Create at least 1 obstacles
  numOfObstables := MATH`rand(n) + 1;

  obtacles := {};
  for i = 0 to numOfObstables do
  (
      x := MATH`rand(n) - negativeOffset;
      y := MATH`rand(n) - negativeOffset;
      obtacles := obtacles union {[ x, y ]};
  );

);

public getObstacles: () ==> set of (seq of real)
getObstacles() == return obtacles;

end Environment
```

Figure 6.85: SAR UAV case study: VDM-RT snippet code on the random generation for obstacles

With the automated test case generation included in the model, we execute the co-simulation several times and present one simulation result where the simulation showed an unexpected UAV behaviour. The co-simulation result is shown in Figure 6.86. The graph shows that *UAV2* fails to avoid collision from an obstacle at <4,-7>. As the Rescue UAV does not have a collision avoidance feature, it also shows that *UAV5* (the Rescue UAV) barely missed the obstacle near the survivor at <9,9>.

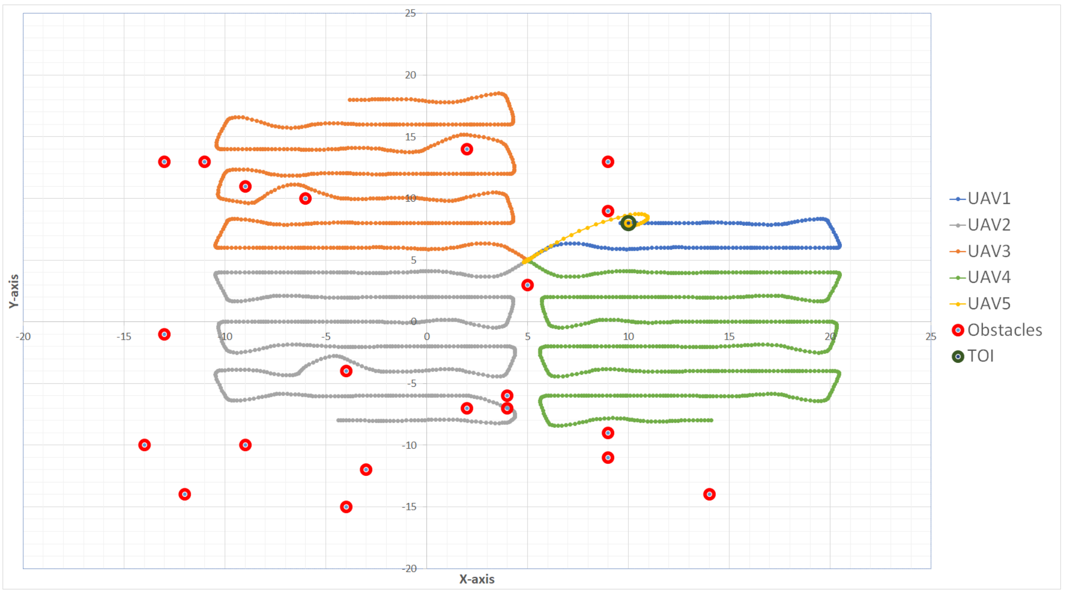


Figure 6.86: SAR UAV case study: automated test case generation of SAR obstacles

6.3.6 Evaluation

This case study demonstrates how ACPSAF can be used to support the design and architecting of SAR UAV ACPS. The case study was engineered using MBSE methodology with ACPSAF, SysML, and INTO-CPS tool chain. The case study also produces at least one View for each of the Viewpoints in ACPSAF. Here, we evaluate the case study against each of the case study requirements defined in Section 6.3.1.2.

REQ1 Each case study should demonstrate the application of external ACPS techniques in conjunction with ACPSAF.

PARTIAL - the SAR UAV case study was initially carried out with ACPSAF and then was modified with the introduction of automated test case generation. Although it demonstrates how automated test case generation can be used in a co-model developed with ACPSAF, the technique used is more of a generic principle rather than an external technique for ACPS. When compared to the first case study, the smart grid model incorporates the Dependability Cage. The Dependability Cage is considered an external ACPS technique as the framework demands for a certain architecture to be fulfilled to enable the runtime monitoring framework. However, the automated test case generation applied here is a generic approach that is specialised for this case study. Nevertheless, current literature in the automated test case generation for autonomous systems is very application specific that there is no better alternative than adopting the generic approach.

REQ2 The case study should strictly follow the ontology provided in ACPSAF without modification.
FULFILLED - the case study does not add or remove the ACPS Ontology used in ACPSAF.

REQ3 The case study should be designed with SysML
FULFILLED - the case study is designed with SysML 1.2 using Modelio.

REQ4 The case study should be carried out through the MBSE life cycle from requirements, design, verification, and validation.
PARTIAL - similar to the first case study, this case study hinted that it has been carried out following the V-model life cycle. It shows that there is a distinction between different phases, from requirement acquisition to system design and V&V. However, the process to which the system evolves through the different phases in the life cycle are not well-defined and documented.

REQ5 The case study should use all the provided Perspective and Viewpoints.
FULFILLED - all ACPSAF Perspective and Viewpoints are engaged in the case study.

REQ6 The case study should produce at least one View for each Viewpoints.
FULFILLED - at least one View was created for each Viewpoint in ACPSAF.

REQ7 The case study should show how Autonomy Requirement can be identified with ACPSAF.
FULFILLED - this is completed in the Requirement Description View.

REQ8 The case study should use the co-modelling and co-simulation approach proposed in ACPSAF to verify and validate ACPS.
FULFILLED - this is carried out in Section 6.3.4.

REQ9 Two case study should be carried out, demonstrating different types of autonomy.
PARTIAL - the case study demonstrated ACPS with self-adaptive architecture to achieve autonomy in SAR scenario. As the principle behind the first case study and the second case study is based on self-adaptive architecture, the two case studies do not validate that ACPSAF is effective beyond self-adaptive ACPS. Initially, a self-organising CPS was considered for the second case study for the completion of REQ9. However, we are unable to find a satisfactory example of self-organising CPS for the case study. Many of the literature on self-organising CPS are still exploring the benefits and feasibility of the technology with no or very limited study in production environment [367, 368], and the few works that do have a real-life pilot study were difficult to measure and validate for effectiveness [369].

REQ10 Two case studies should be carried out, demonstrating different domains of ACPS.

FULFILLED - the case study demonstrated a smart grid ACPS from the energy infrastructure domain. Together with the first case study, it showed that ACPSAF is applicable to at least two different ACPS domains.

REQ11 The case study should be carried out by a systems engineer.

NOT FULFILLED - this case study was carried out by the author, which invalidates the requirement due to familiarity with the framework.

The evaluation list shows that most of the requirements are fulfilled with some exceptions. The full evaluation of the research will be carried out in Chapter 7. Here, we provide a personal summary on the strengths and weaknesses of ACPSAF that we identify from conducting the case study. The strength and limitations of ACPS highlighted in the evaluation of the first case study will not be raised again here.

- Application on existing CPS. The case study shows that ACPSAF can be used on existing CPS to introduce an autonomy capability. Users can either make the complete model of the existing system, or treat existing components as a black box system. The latter method is used for this case study, which allows the ACPS engineer to focus on the new features without having to understand the complexity of the existing components. The ability to be used for existing CPS to introduce autonomy will be quite beneficial as many ACPS are based on large, existing CPSs.

6.4 Summary

This chapter presents two case studies as a form of demonstration and validation for ACPSAF. The first case study is based on a self-healing smart grid, where configurations to maximise energy supply to customers need to be achieved autonomously, especially in the presence of fault. The second case study is based on SAR UAVs. Each UAV is capable of calculating its own flight plan, navigating through a given search area, avoiding collision with obstacles, and communicating its findings.

Each of the case studies also demonstrates how ACPSAF can be integrated with external autonomous techniques. The self-healing smart grid case study integrates an external runtime monitoring framework called the Dependability Cage to influence the ACPS behaviour when operating under untested behaviour. The UAV case study integrates an automated test case generation technique to automatically generate and test the ACPS against different SAR scenarios. The SAR UAVs integrate automated test case generation to generate randomly placed obstacles in the search area that the UAVs must avoid while searching for the survivor. At the end of each case study, we evaluate the extent to which the case study meets the evaluation criteria defined in Section 6.1.

7 Conclusion and Future Work

This thesis has reported a contribution towards the goal of developing MBSE approaches for autonomous CPSs. The thesis opened by introducing the concept of ACPS and highlighting challenges in engineering such systems (Chapter 1). The state of the art in model-based engineering for ACPSs was assessed by the means of a literature review described in Chapter 2. This led to the identification of a research methodology, objectives, and evaluation plan (Chapter 3), where we hypothesised that "Using Architectural Framework for ACPS improves the quality and efficiency of the engineering process of ACPS.". To be specific, we proposed an AF that is designed specifically to the need of ACPS using the autonomy-explicit approach by promoting shared understanding through Ontology, best practice with MBSE, and cross-domain methods (e.g., co-modelling and simulation from CPS). The underlying intention here was that an AF incorporating these features could be of value in managing development risks associated with ACPSs. The bulk of our work focuses on two main concrete contributions: the Ontology intended to support MBSE of ACPS as described in Chapter 4 and the architectural framework ACPSAF described in Chapter 5.

In Chapter 3, we considered a range of research methodologies relevant to the domain and adopted Type 3 DRM, a methodology that focuses on the creation of a solution (in this case, the ACPSAF) where current support is partial or non-existent. In type 3 DRM research, only minimum validation is performed against the developed solution, with the goal not being to prove the success of the solution, but to get an indication of the applicability, usability, and usefulness of the solution, and any indications of issues that need detailed evaluation. In our case, we developed two case studies (Chapter 6) which were used to demonstrate the application of ACPSAF and gain an initial indication of its usefulness.

This chapter reviews the results of the work reported in the rest of the thesis and looks towards the next steps. In Section 7.1, we consider the limitations of the approach as developed here. In Section 7.2, we consider the extent to which the work reported here has met the research questions that were set out in Chapter 3, and this motivates a discussion of potential future work in Section 7.3.

7.1 Limitations

In this section, we provide an overview of the limitations of ACPS Ontology, ACPSAF, and the case studies presented in our research. The limitations provided here will impact the conclusions that can be drawn from the research evaluation discussed in Section 7.2.

7.1.1 ACPS Ontology

One weakness of the proposed ontology is that the ACPS Ontology is derived using a review-only method. This means that the ontology is built purely by reviewing existing literature and ontologies without involving qualitative input such as surveys or interviews with domain experts (i.e., ACPS engineers). The importance of qualitative input from experts when building the ontology is inherently dependent on each ontology. Expert input is very useful as it helps reduce definition bias from the ontology engineer. However, with the introduction of a powerful search engine, it is possible to search for the "most influential" concept definition based on publications with the highest citations. In addition to consideration of international standards and advancement in formal ontologies, this tool enables review-based ontologies to be sufficiently rigorous in most cases.

However, this approach (looking at citation counts) may presumably reflect the concepts and usage of the research community and may differ from the needs of practitioners. We still consider this as a weakness in the context of ACPS Ontology, as one advantage of conducting an expert survey as part of ontology engineering is that it helps promote a state of the art ontology definition. This is especially important for rapidly-evolving and typically new technology, where the definition of a specific ontology may evolve quickly. It is possible that review-based results in ontology engineering favour popular, old definitions that do not reflect current use. Our attempt to mitigate this is to conduct a literature review in the domain of autonomy and CPS to ensure that we consider the state of the art definition and clarify differences on terms that are sometimes used interchangeably (e.g., CPS vs embedded system vs IoT) in the literature.

7.1.2 ACPSAF

The ACPSAF is offered at the level of a laboratory prototype and is quite some way from being ready for deployment in practice. Two weaknesses are worth mentioning in this regard: the lack of an ACPSAF-specific MBSE process and the lack of a test for AF completeness.

7.1.2.1 ACPSAF process

By ACPSAF process, we mean the MBSE process to using ACPSAF in ACPS projects. A process provides clear guidance on how and when to use the AF to ensure that the AF is being used correctly and effectively. A rigorous process could offer variations that consider aspects such as the scale of the project (e.g., stakeholder size), level of formality desired, skills/experience of the engineers, and other organisational facets in its design. The MBSE process is often considered separately from the architectural framework. However, this may be seen as a weakness because if a system is unusual and complex enough to require an AF, then it may also require a tailored process for it to be used effectively. However, an architecture-centric process is one of the least mature aspects in MBSE as there are many existing MBSE processes that are based on Capability Maturity Model Integration (CMMI) level 0 (e.g., Ad Hoc, Big Design Up Front).

The process used in the case studies simply goes from one viewpoint to the next following the presentation order of the viewpoints in the AF definition. This is an ad hoc, reactive process where the processes are not properly planned, documented, monitored, and controlled. This approach could cause inefficient use of the AF, and at its worst, cause the AF to be used incorrectly in a way that could lead to project failure. The two case studies probably do not suffer this issue because of their relative simplicity (see section 7.1.4) and the fact that the author (as the stakeholder) has a clearer understanding of the AF. However, this would eventually become an issue with more complex and large-scale ACPS projects. The development of the ACPSAF process is a high priority for further work (Section 7.3).

7.1.2.2 ACPSAF completeness

The ACPSAF is developed following the framework development method of Framework for architectural Framework (FAF). However, we are unable to find formal verification showing that the FAF is complete, so we cannot assume that ACPSAF is complete even if we follow FAF rigorously. Therefore, it will be beneficial for the rigour of the framework that some form of test of completeness is used against ACPSAF to check for features such as scope of coverage. Formal verification can be introduced to verify the mathematical rigour, and philosophical work such as those by Merleau-Ponty [370] can be considered for philosophical rigour. Some views in the ACPSAF could also be improved by considering regulation in the ACPS domain (e.g., how does regulation in the ACPS domain potentially impact the rules defined in the Rule Definition View?).

7.1.3 System first approach

Our ACPS approach is influenced by the idea that many ACPS projects are based on upgrading existing systems/CPS with added autonomy. This is especially visible in the domain of electrical grids and autonomous on-road vehicles, but not so much in other domains such as robotics. In the case of systems with added autonomy, the ACPS project often starts with an existing system as the design constraint. For example, the design of autonomous, on-road vehicles would benefit from starting the design with a skeleton vehicle that would meet all the minimum requirements for on-road vehicles first before exploring all of the possible design parameters for autonomy.

However, in cases where there is no pre-existing constraint to the system design, it will be beneficial to start with the essence of the autonomy objective before coming up with all the creative and unique ways that can satisfy the objective and reducing the design choice while considering the constraint. After that, the best design can finally be picked from the remaining choices. This way of thinking would promote the design of ACPS that is unique to ACPS. We believe that our current ontology is flexible enough to support both views (i.e., ACPS as CPS with added autonomy, and ACPS as a new, unique system). However, we have not properly considered what might need to be modified from ACPSAF in order to support both approaches. Since both the case studies also follow the CPS-first approach, we do not have the indication from our research on the readiness of ACPSAF to support both approaches.

This is not to say that ACPSAF cannot be used to design ACPS from scratch (ACPSAF users can start from the autonomy perspective viewpoints first before going back to the CPS perspective viewpoints). However, the process to do this has not been shown from the case study and there may be additional viewpoints that could make this more effective. Therefore, there is a risk that the current ACPSAF as it is does not promote the creativity in the ACPS engineering project to design new and unique ACPSs.

7.1.4 Case studies

Case studies were used in the context of the DRM Descriptive Study II activity to perform an initial evaluation of ACPSAF. Limitations of these studies must be borne in mind when drawing conclusions about our proposed framework. In this section, we discuss four such limitations relating to the content of the conditions under which they were undertaken with a single stakeholder, and their relative simplicity.

7.1.4.1 Single stakeholder

A significant limitation that applies to both case studies is that they are selected and developed by a single stakeholder, namely the author. This means that observations from the case study cannot be used to evaluate the effectiveness of ACPSAF in its intended content, which is the engineering of ACPS projects that typically involve multiple stakeholders who often have different expertises, interests, budget, etc. To put it simply, ACPSAF aims to improve collaborative work in ACPS engineering (e.g., communication), but since the case study was performed by one stakeholder, effectiveness of the ACPSAF in improving collaborative work cannot be evaluated.

7.1.4.2 Smart grid simplification

Several simplifications have been introduced in the smart grid case study that differentiates it from a typical smart grid in a real ACPS project. The overview of these simplifications and its impacts on the ACPSAF evaluation is as listed:

- Topology. The grid topology used for the case study is not based on a real grid. The topology has been designed in such a way that it is easy to demonstrate the self-healing function.
- Grid components. The components and configuration used in the case study do not reflect the complexity in realistic smart grids, such as differences in the number and types of equipment used.
- Stakeholder. The number and variety of stakeholders in a real smart grid project would introduce much more complexity to the system. Stakeholders such as distribution network operators, contractors, and local councils would all have different interests, and some may or may not affect the autonomy objectives.
- Views captured. Since the target case study itself is simple, the views captured using the ACPSAF are also very simple. It does not show whether each of the viewpoints would be appropriate for realistic, more complex ACPSs. This limitation extends to the second case study as well.

7.1.4.3 SAR UAV

Similar to the first case study, this case study also has simplifications made that affect the evaluation:

- Rescue function. Although the case study is based on the broadly-termed Search and Rescue UAVs, the case study stops at the search function. The rescue function in UAV would touch deeper into the territory of autonomy permission, where the ACPS may have the ability to rescue the target, but may not have the permission to do so (e.g., lack of approval from the authority, unsafe weather, consent issue, etc).
- Self-organising. The case study misses the opportunity to demonstrate a truly self-organising CPS as it relies on a central controller to coordinate and reassign the search area.
- Stakeholder. Similar to the first case study, a more varied set of stakeholders would enable the possibility of exploring a more complex ACPS. For example, some of the UAVs could be owned by separate stakeholders who could recall their UAVs in the middle of the operation. A contract would then need to be established in the SAR protocol to ensure a successful mission (e.g., the leaving UAV must notify other UAVs of its retirement).
- Control simplification. Simplifications have been made such that the obstacle avoidance controller is less sophisticated than would be expected. For example, the case study only considers 2D manoeuvre when it should ideally consider 3D movement. It is also unclear how the control would deal with external factors such as high wind weather.

7.1.4.4 Autonomy complexity

Section 2.3.5.2 has identified that ML ACPS would require a different design approach. However, this characteristic has not been translated into the ontology and ACPSAF. As a result, the current ACPSAF does not support ML in the sense of reinforced learning. It is still possible to integrate independent ML into ACPS with ACPSAF, such as installing an image classification ML program into the UAV's on-board computer to enable human recognition. However, what is not possible is the type of Reactive Learning where the ACPS itself learns from past experiences to adapt and adjust its future behaviour. We believe that this should be included in further work as current ACPSAF can be used as the basis for ML support.

7.2 Evaluation

In this section, we evaluate the ACPS Ontology and ACPSAF based on the case studies validation with considerations of the limitations stated in the previous section. The DRM methodology does not explicitly state the format of the evaluation for Initial DS-II research, but indicates that the evaluation should show (i). an indication of the applicability, usability, and usefulness of the Support; and (ii). indication of the issues, factors, and links that need detailed evaluation. Therefore, the following section will address these two points and the evaluations pertaining to the ACPS challenges.

7.2.1 Applicability, usability, and usefulness of the Support

The applicability of ACPSAF can be objectively evaluated based on the performance of ACPSAF on the case study. The case study shows that ACPSAF can be used for:

- Two domains of ACPS. We infer that ACPSAF is non-domain specific.
- Cross-discipline method. The case study shows a combination of methods typically used in CPS (i.e., co-modelling) and autonomous systems (i.e., automated test case generation, real time monitoring system). This hypothetically means that experts from each domain can ensure that best practices from their respective domain are being observed in the ACPS project.
- Small scale ACPS. The case studies are on a much smaller scale than typical ACPS projects in terms of complexity and/or scale. Therefore, it is unclear whether there would be a scalability issue. This complexity applies for both functional (i.e., more components) and organisational (i.e., more stakeholders) aspects.
- ACPS with self-adaptive behaviour. The author believes that ACPSAF would be applicable for self-organising ACPSs as well. However, this is not conducted in the case study and therefore cannot be validated.
- All types of autonomy intelligence except for machine learning. ACPSs with a Reinforced Learning type of ML would require a different design architecture that is currently not supported by ACPSAF.

The ultimate usability of ACPSAF cannot be objectively evaluated as the ACPSAF has not yet been used by its target user group. Therefore, the indication of usability can only be in the form of the author's reflections on the AF and case studies.

- The lack of a defined rigorous process for applying ACPSAF will likely cause issues during development, especially when used by a first-time user. For example, what level of understanding (e.g., SysML knowledge) would a stakeholder need before they are ready to use the ACPSAF?

- Putting aside domain knowledge, a stakeholder (depending on their class/role) may need an understanding of MBSE (especially because of the lack of process), SysML, and FMI compliant tools (if co-modelling and simulation is used).
- This knowledge and skill cannot be derived from just the information within the framework, so stakeholders would need to find external sources to acquire them.

Given the experience at developing and applying ACPSAF, the author believes that the usefulness of ACPSAF may be largely driven by finding the most suitable type of ACPS projects to use ACPSAF and developing a suitable process for them. For example, perhaps ACPSAF would be suitable for smaller ACPS projects that could span to varying ACPS domains. An example of this could be projects focusing on general-purpose robotics (or multiple robots for varying domains) where the development team and system is relatively smaller in scale, but have to deal with the intricacies of domain problems.

Another possibility is that maybe ACPSAF could be suitable for ACPS projects that are research-based or prototype-oriented in nature, where the stakeholders typically have limited expertise and resources and are conducting the project for exploratory purposes (e.g., for learning, or testing initial proof of concept for funding). As ACPS becomes increasingly common, there may be more applications of ACPS that are more modular than traditional CPS, such as appliances in a smart home. However, these are only deductions from the author based on the case studies, and they can only be validated with a proper evaluation plan, which will be discussed in Section 7.3.

7.2.2 Issues, factors, and links that need detailed evaluation

Many of the issues with the ACPSAF (and other aspects of the research) have been addressed in the previous section about limitations. However, these can be generalised into the need for a test of completeness for ACPSAF, the need for support for more complex autonomy ACPSs, the need for an ACPSAF process, and the need for a proper validation case study. From those, the need for an ACPSAF process and proper validation would be in the top priority in order to move a step closer towards finding a more conclusive answer to the research hypothesis. The need for an ACPSAF process will become apparent when ACPSAF is used in a proper case study. Therefore, the process is likely to be developed either before or during the case study.

7.2.3 ACPS challenges

In Section 2.3.2, we mentioned significant challenges in ACPS as:

> Challenge 1: How can we have assurance in the behaviour of an ACPS even in uncertain environments?

> Challenge 2: How can we have assurance in the behaviour of an ACPS even when the system's behaviour changes at run-time?

In our case studies, we attempted to incorporate state of the art methods from both the CPS (co-modelling) and autonomous system domains (automated test case generation and run-time monitoring system) to address some of these challenges. Here, we provide our observations on the effectiveness of those methods in addressing the above challenges.

We are unable to find a method in the literature that provides a guarantee for Challenge 1. However, several methods have been developed to increase assurance in the behaviour of ACPS up to the level of acceptance. One of the most promising methods is related to automated testing or its variants. The premise of a dependable ACPS is not that the behaviour can be guaranteed, but rather, it is more dependable that its human counterpart. A well-known example of this is in the field of on-road vehicles where the goal is to prove that an autonomous vehicle will be more reliable than (any) human driver. The challenge then becomes: how do you verify that an autonomous vehicle is more reliable than any human driver? This is where variants of automated testing are used to test the system against large numbers of unique use cases and measure the system's performance against human statistics. However, developing realistic and meaningful use cases automatically is another challenge in itself.

So in regards to Challenge 1, automated test case generation can be incorporated with ACPSAF (as shown in the UAV case study) to increase the assurance in the behaviour of an ACPS with the aim of developing an ACPS that is more dependable than a human operator.

Challenge 2 can only be partially addressed based on our literature review. One of the more promising methods in gaining assurance in the behaviour of ACPS is by means of run-time monitoring, where a safety net perimeter is set around the behaviour of the ACPS to guard against unexpected behaviour. The reason why this is only a partial solution is because this approach has two main issues. First is that run-time monitoring can be an anti-autonomy pattern when used carelessly. For example, ACPSs with ML, which are expected to perform better than a human operator, will definitely be performing "unexpected behaviour" as it continually learns and eventually outperforms its designer. Preventing an unexpected behaviour in ACPSs could easily cause the system to lose its novelty. Another approach to run-time monitoring is to simply pause the operation and wait for approval/rejection from a human supervisor. In a way, this is akin to permission management on potentially unsafe behaviour. This approach is the source of the second issue, in which this approach will not be suitable for ACPSs that need a very quick reaction. For example, pausing operation may be acceptable for UAVs in a rescue scenario where the UAV requests for permission before attempting to rescue the target. However, in self-driving cars, unexpected behaviour in the presence of an anomaly can be resolved very quickly (either good or bad) in fast traffic situations. There is just not enough time for human supervision in some cases.

So for Challenge 2, the run-time monitoring system, such as the Dependability Cage method used in smart grid case study, can be used to gain assurance in the behaviour of some types or domains of ACPS. It may be possible to improve the run-time monitoring approach with a very well-defined boundary that does not hinder the self-adaptive ability of the ACPS, but also safeguards it from system failure at the same time. However, we are unable to find the method to find these parameters other than fine-tuning the ACPS on a case by case basis.

7.3 Further Work

The work described in this thesis can be extended in a number of ways. In this section, we propose areas of improvement and future work. We indicate the rough approach to achieve each of the suggestions and consider the implications they have on the work presented in this thesis. We discuss these areas in order of relative importance.

7.3.1 Comprehensive Descriptive Study II

As highlighted in the previous sections, the evaluation of the ACPSAF is limited to the lack of a proper validating case study. Here, we provide a rough overview of what the next step in ACPSAF development could look like:

- It may be best to begin with deciding the type of ACPS project to focus on. Based on the existing evaluation, there will be a higher chance of success (in terms of finding a suitable project and success of the ACPS project itself) if focus is placed on ACPS projects of a smaller scale.
- If focus is chosen on less traditional types of CPS (e.g., small scale CPS), make sure that the need for ACPSAF exists for such projects. This should ideally be done through a survey or interview with the stakeholder(s) of the potential projects.
- Before the case study, develop an ACPSAF process that would be suitable for that type of project (i.e., based on scale, complexity, etc). More details on the ACPSAF process will be covered in the next section.
- It would be best if the author does not get involved in the case study until completion. However, this may be unavoidable in the early iteration until a sufficiently rigorous ACPSAF process is developed.
- A qualitative research tool, such as questionnaires, can be used to gather evaluation about the ACPSAF. The questions can be made in the form of competency questions that address different aspects of the ACPSAF (e.g., is the Ontology rich enough to describe the target ACPS? Does the ACPSAF improve communication between stakeholders? etc.)
- The evaluation may raise the need for modifications/improvements in the ACPSAF, and there may be several case studies and several iterations of improvement. With each iteration, the ACPSAF should improve in quality and maturity. Methods such as Technology Readiness Level (TRL) can be used to measure the

maturity of ACPSAF and decide when the framework is ready to transition from development to deployment [371]. At the moment, ACPSAF would measure between TRL 3 and TLR 4, and full deployment is recommended when a technology is measured to be at TLR 9.

7.3.2 ACPSAF process

In this thesis, we have demonstrated how an MBSE activity can be carried out using a selection of modelling languages (SysML), tools (INTO-CPS), and architecture frameworks (ACPSAF). To maximise the effectiveness of the MBSE activity, it is also important to specify the process in which these tools and frameworks should be used. A well-thought out MBSE process would be architecture centric, with a well-defined process for using the MBSE framework, language, and tools. On top of systematic progression between each design phase, the MBSE process should also consider the context of the project, organisation, and stakeholder's competency.

The MBSE process is the least mature aspect of MBSE compared to language, framework, and tools. MBSE processes are typically ad hoc, at level 1-2 of the Capability Maturity Model Integration (CMMI) scale; this is the case in our research as well. The process that we adopt for our work is the generic V-process for MBSE that is only concerned with the development stages in the project life-cycle. Our work would benefit from MBSE processes for ACPS that deal with non-functional project concerns such as project scheduling, risk management, and organisational training.

As there is no existing MBSE process for ACPS-specific application, the best approach would be to start from existing MBSE processes for generic SE or CPS applications and re-engineer them for ACPS. As there is no standardised method to evaluate the competency of MBSE processes, the proposed MBSE process needs to be validated as well.

7.3.3 Complex autonomy

Support for complex permission In Section 2.3.3, we briefly discuss the difference between the capability vs the right (permission) of autonomy. In our work, we have considered ACPSs with non-conflicting autonomy objectives. For example, the UAVs in the SAR UAV case study are able to cooperate easily because of a shared goal. However, a hypothetical UAV swarm made of independent UAVs with an independent goal that may collaborate only on a temporary, non-binding operation towards a temporary shared goal may induce scenarios where some UAVs decide to exit the formation in the middle of operation. On top of the ability to continually reconfigure the swarm formation, there may be a need to establish some form of contract policy to prevent unexpected accidents. For example, a UAV in the centre of a tightly-packed swarm of UAVs formation should have to follow an exit protocol and coordinate with other UAVs to safely exit the swarm before it is allowed for full flight control.

This is best discussed in combination with the Systems-of-Systems (SoS) architecture, where multiple constituent systems combine into a new and more complex system to offer more functionality than the sum of each constituent system individually. The swarm of UAVs example is one such system.

Support for Machine Learning

One of the ACPS challenges is to guarantee the behaviour of ACPS even when it changes its behaviour at run-time. Changes in behaviour can be caused by changes in the environment and in the ACPS's autonomy policy. ACPSs with machine learning capabilities such as Reinforcement Learning could learn from the environment and change their internal decision-making policy to maximise autonomy performance. This results in new behaviour that is not verified during design time and can be unpredictable. Although ML ACPSs can be designed with ACPSAF, it does not provide explicit support to define such architecture. We have included a suggestion of how our current ontology could be extended to include concepts from reinforcement learning in Chapter 4.6. The next step would be to add Viewpoints to support the description of such architecture. It is also possible to integrate ACPSAF with an existing framework that already supports ML [66].

7.3.4 Completeness test

This is driven by limitations stated in Section 7.1.2.2, where the completeness of ACPSAF has not been validated. A good starting point is to find out whether FAF (the framework used to design ACPSAF) has been validated to be complete. At the time of writing, we are unable to find this proof. The next action would be to perform a test of completeness, especially one that checks for scope of coverage, mathematical equations, and philosophical rigour.

Bibliography

[1] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An architecture-based approach to self-adaptive software," *IEEE Intelligent Systems and their Applications*, vol. 14, pp. 54–62, May 1999.

[2] E. A. Lee, "Cyber physical systems: Design challenges," Tech. Rep. UCB/EECS-2008-8, EECS Department, University of California, Berkeley, Jan 2008.

[3] J. Shi, J. Wan, H. Yan, and H. Suo, "A survey of cyber-physical systems," in *2011 International Conference on Wireless Communications and Signal Processing (WCSP)*, pp. 1–6, Nov 2011.

[4] E. Makiö-Marusik, B. Ahmad, R. Harrison, J. Mäkiö, and A. Walter, "Competences of cyber physical systems engineers - survey results," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pp. 491–496, May 2018.

[5] I. Akkaya, P. Derler, S. Emoto, and E. A. Lee, "Systems engineering for industrial cyber-physical systems using aspects," *Proceedings of the IEEE*, vol. 104, pp. 997–1012, May 2016.

[6] R. S. Ross, M. McEvilley, and J. C. Oren, "Systems security engineering: Considerations for a multidisciplinary approach in the engineering of trustworthy secure systems," tech. rep., NIST, 2016.

[7] M. Törngren, M. E. Grimheden, J. Gustafsson, and W. Birk, "Strategies and considerations in shaping cyber-physical systems education," *SIGBED Rev.*, vol. 14, pp. 53–60, Jan. 2017.

[8] J. Estefan, "Survey of model-based systems engineering (mbse) methodologies," *INCOSE MBSE Focus Group*, vol. 25, Jan 2008.

[9] Y. Mordecai and D. Dori, "Minding the cyber-physical gap: Model-based analysis and mitigation of systemic perception-induced failure," *Sensors*, vol. 17, p. 1644, Jul 2017.

[10] L. Jinzhi, *A Framework for Cyber-physical System Tool-chain Development :A Service-oriented and Model-based Systems Engineering Approach.* PhD thesis, KTH Royal Institute of Technology, May 2019.

[11] B. Connett and B. O'Halloran, "Systems engineering design: Architecting trustworthiness in cyber physical systems using an extended aggregated modality," *Procedia Computer Science*, vol. 140, pp. 4–12, 2018. Cyber Physical Systems and Deep Learning Chicago, Illinois November 5-7, 2018.

[12] A. L. Ramos, J. Ferreira, and J. Barcelo, "Model-based systems engineering: An emerging approach for modern systems," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 42, pp. 101–111, Jan 2012.

[13] C. Zheng, J. L. Duigou, P. Hehenberger, M. Bricogne, and B. Eynard, "Multidisciplinary integration during conceptual design process: A survey on design methods of cyber-physical systems," in *Proceedings of the DESIGN 2016 14th International Design Conference*, 2016.

[14] P. H. Nguyen, S. Ali, and T. Yue, "Model-based security engineering for cyber-physical systems: A systematic mapping study," *Information and Software Technology*, vol. 83, pp. 116–135, 2017.

[15] H. He and J. Yan, "Cyber-physical attacks and defences in the smart grid: a survey," *IET Cyber-Physical Systems: Theory Applications*, vol. 1, no. 1, pp. 13–27, 2016.

[16] G. Callow, G. Watson, and R. Kalawsky, "System modelling for run-time verification and validation of autonomous systems," in *2010 5th International Conference on System of Systems Engineering*, pp. 1–7, Jun 2010.

[17] A. Aniculaesei, J. Grieser, A. Rausch, K. Rehfeldt, and T. Warnecke, "Towards a holistic software systems engineering approach for dependable autonomous systems," in *2018 IEEE/ACM 1st International Workshop on Software Engineering for AI in Autonomous Systems (SEFAIAS)*, pp. 23–30, May 2018.

[18] J. Katupitiya, R. Eaton, and T. Yaqub, "Systems engineering approach to agricultural automation: New developments," in *2007 1st Annual IEEE Systems Conference*, pp. 1–7, Apr 2007.

[19] P. Helle, W. Schamai, and C. Strobel, "Testing of autonomous systems - challenges and current state-of-the-art," *INCOSE International Symposium*, vol. 26, pp. 571–584, Jul 2016.

[20] W. Damm and R. Galbas, "Exploiting learning and scenario-based specification languages for the verification and validation of highly automated driving," in *2018 IEEE/ACM 1st International Workshop on Software Engineering for AI in Autonomous Systems (SEFAIAS)*, pp. 39–46, May 2018.

[21] K. Zhang, J. Li, Z. Lu, M. Luo, and X. Wu, “A scene-driven modeling reconfigurable hardware-in-loop simulation environment for the verification of an autonomous cps,” in *Proceedings - 2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics*, vol. 1, pp. 446–451, 2013.

[22] W. Xia, H. Li, and B. Li, “A control strategy of autonomous vehicles based on deep reinforcement learning,” in *2016 9th International Symposium on Computational Intelligence and Design (ISCID)*, vol. 2, pp. 198–201, Dec 2016.

[23] C. Wang, J. Wang, and X. Zhang, “A deep reinforcement learning approach to flocking and navigation of uavs in large-scale complex environments,” in *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 1228–1232, Nov 2018.

[24] X. Ruan, D. Ren, X. Zhu, and J. Huang, “Mobile robot navigation based on deep reinforcement learning,” in *2019 Chinese Control And Decision Conference (CCDC)*, pp. 6174–6178, Jun 2019.

[25] S. Ku, S. Jung, and C. Lee, “Uav trajectory design based on reinforcement learning for wireless power transfer,” in *2019 34th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*, pp. 1–3, Jun 2019.

[26] H. X. Pham, H. M. La, D. Feil-Seifer, and L. Van Nguyen, “Reinforcement learning for autonomous uav navigation using function approximation,” in *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–6, Aug 2018.

[27] J. Kim, S. Cha, M. Ryu, and M. Jo, “Pre-training framework for improving learning speed of reinforcement learning based autonomous vehicles,” in *2019 International Conference on Electronics, Information, and Communication (ICEIC)*, pp. 1–2, Jan 2019.

[28] N. Ebell and M. Pruckner, “Coordinated multi-agent reinforcement learning for swarm battery control,” in *2018 IEEE Canadian Conference on Electrical Computer Engineering (CCECE)*, pp. 1–4, May 2018.

[29] L. Gao, J. Zeng, J. Wu, and M. Li, “Cooperative reinforcement learning algorithm to distributed power system based on multi-agent,” in *2009 3rd International Conference on Power Electronics Systems and Applications (PESA)*, pp. 1–4, May 2009.

[30] M. A. Yahya, M. A. Yahya, and A. Dahanayake, “Autonomic computing: A framework to identify autonomy requirements,” *Procedia Computer Science*, vol. 20, pp. 235–241, 2013. Complex Adaptive Systems.

[31] E. Vassev and M. Hinchey, "Autonomy requirements engineering," in *2013 IEEE 14th International Conference on Information Reuse Integration (IRI)*, pp. 175–184, Aug 2013.

[32] J. M. Beer, A. D. Fisk, and W. A. Rogers, "Toward a framework for levels of robot autonomy in human-robot interaction," *J. Hum.-Robot Interact.*, vol. 3, pp. 74–99, July 2014.

[33] N. A. Qureshi, A. Perini, N. A. Ernst, and J. Mylopoulos, "Towards a continuous requirements engineering framework for self-adaptive systems," in *2010 First International Workshop on Requirements*, pp. 9–16, Sep. 2010.

[34] E. Vassev and M. Hinchey, *Engineering Requirements for Autonomy Features*, pp. 379–403. Springer International Publishing, 2015.

[35] ISO, IEC, IEEE, "42010: 2011. Systems and Software Engineering, Architecture Description," *International Standard*, 2011.

[36] J. Fitzgerald, J. Bryans, P. G. Larsen, and H. Salim, "Collaborative systems of systems need collaborative design," in *Collaborative Systems for Smart Networked Environments* (L. M. Camarinha-Matos and H. Afsarmanesh, eds.), (Berlin, Heidelberg), pp. 16–23, Springer Berlin Heidelberg, 2014.

[37] H. Salim and J. S. Fitzgerald, "Towards multi-models for self-* cyber-physical systems," in *The 15th Overture Workshop: New Capabilities and Applications for Model-based Systems Engineering* (J. S. Fitzgerald, P. W. V. Tran-Jørgensen, and T. Oda, eds.), no. CS-TR-1513, pp. 44–58, School of Computing, Newcastle University, Oct 2017.

[38] G. Di Marzo Serugendo, J. Fitzgerald, and A. Romanovsky, "Metaself: An architecture and a development method for dependable self-* systems," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, (New York, NY, USA), pp. 457–461, ACM, 2010.

[39] R. Mallett, J. Hagen-Zanker, R. Slater, and M. Duvendack, "The benefits and challenges of using systematic reviews in international development research," *Journal of Development Effectiveness*, vol. 4, p. 445, Sep 2012.

[40] R. Silva and F. Neiva, "Systematic literature review in computer science - a practical guide," tech. rep., Federal University of Juiz de Fora, Nov 2016.

[41] A. Kofod-Petersen, "How to do a structured literature review in computer science," tech. rep., alexandra instituttet, May 2015.

[42] H. Chen, "Applications of cyber-physical system: A literature review," *Journal of Industrial Integration and Management*, vol. 02, Dec 2017.

[43] H. Muccini, M. Sharaf, and D. Weyns, “Self-adaptation for cyber-physical systems: A systematic literature review,” in *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS ’16, (New York, NY, USA), pp. 75–81, ACM, 2016.

[44] R. Lima, D. De Vargas, and L. Fontoura, “System of systems requirements: A systematic literature review using snowballing,” in *SEKE*, Jul 2017.

[45] M. I. Babar, M. Ghazali, and D. N. A. Jawawi, “Systematic reviews in requirements engineering: A systematic review,” in *2014 8th. Malaysian Software Engineering Conference (MySEC)*, pp. 43–48, Sep. 2014.

[46] S. Chakraborty, “Eda for cyber-physical systems,” in *2017 7th International Symposium on Embedded Computing and System Design (ISED)*, pp. 1–2, Dec 2017.

[47] J. Liu and J. Lin, “Design optimization of wirelesshart networks in cyber-physical systems,” *Journal of Systems Architecture*, vol. 97, pp. 168–184, 2019.

[48] K. Bauer and K. Schneider, “Teaching cyber-physical systems: A programming approach,” in *Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education*, WESE ’12, (New York, NY, USA), pp. 1–8, ACM, 2013.

[49] J. Fitzgerald, P. Larsen, and M. Verhoef, *From embedded to cyber-physical systems: Challenges and future directions*. Springer Berlin Heidelberg, 2014.

[50] A. Abid, M. Hammadi, J. Choley, A. Rivière, M. Barkallah, J. Louati, and M. Haddar, “A holonic-based method for design process of cyber-physical reconfigurable systems,” in *2016 IEEE International Symposium on Systems Engineering (ISSE)*, pp. 1–5, Oct 2016.

[51] B. M. H. Alhafidh and W. H. Allen, “High level design of a home autonomous system based on cyber physical system modeling,” in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 45–52, Jun 2017.

[52] K. McKeever, Y. Zeleke, M. Bunting, and J. Sprinkle, “Experience report: Constraint-based modeling of autonomous vehicle trajectories,” in *Proceedings of the Workshop on Domain-Specific Modeling*, DSM 2015, (New York, NY, USA), pp. 17–22, ACM, 2015.

[53] M. Schranz, M. Sende, A. Bagnato, E. Brosse, and A. Eckel, “Modeling cps swarms: An automotive use case,” *Ada User Journal*, vol. 40, no. 3, pp. 165–168, 2019.

[54] Y. Z. Lun, A. D'Innocenzo, F. Smarra, I. Malavolta, and M. D. D. Benedetto, "State of the art of cyber-physical systems security: An automatic control perspective," *Journal of Systems and Software*, vol. 149, pp. 174–216, 2019.

[55] J. Wang, Zhao Qianchuan, and Zhao Yin, "An effective framework to simulate the cyber-physical systems with application to the building and energy saving," in *Proceedings of the 32nd Chinese Control Conference*, pp. 8637–8641, Jul 2013.

[56] C. A. González, M. Varmazyar, S. Nejati, L. C. Briand, and Y. Isasi, "Enabling model testing of cyber-physical systems," in *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, MODELS '18, (New York, NY, USA), pp. 176–186, ACM, 2018.

[57] B. Schatz, "The role of models in engineering of cyber-physical systems-challenges and possibilities," in *Tagungsband - Dagstuhl-Workshop MBEES 2014: Modellbasierte Entwicklung Eingebetteter Systeme X*, pp. 19–25, 2014.

[58] R. Alguliyev, Y. Imamverdiyev, and L. Sukhostat, "Cyber-physical systems and their security issues," *Computers in Industry*, vol. 100, pp. 212–223, 2018.

[59] J. Brings, "Verifying cyber-physical system behavior in the context of cyber-physical system-networks," *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pp. 556–561, 2017.

[60] B. Dowdeswell, R. Sinha, and E. Haemmerle, "Torus: Tracing complex requirements for large cyber-physical systems," in *2016 21st International Conference on Engineering of Complex Computer Systems (ICECCS)*, pp. 23–32, Nov 2016.

[61] C. Ponsard, G. Dallons, and P. Massonet, "Goal-oriented co-engineering of security and safety requirements in cyber-physical systems," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9923 LNCS, pp. 334–345, 2016.

[62] M. Glatt, D. Kull, B. Ravani, and J. Aurich, "Validation of a physics engine for the simulation of material flows in cyber-physical production systems," *Procedia CIRP*, vol. 81, pp. 494–499, Jan 2019.

[63] Zhenyu Wu, Tie Qiu, Jiping Wu, Zheng Nie, and Guang Hu, "A state-prediction-based control strategy for uavs in cyber-physical systems," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 691–694, Oct 2016.

[64] A. Tolk, F. Barros, A. D'Ambrogio, A. Rajhans, P. J. Mosterman, S. S. Shetty, M. K. Traoré, H. Vangheluwe, and L. Yilmaz, "Hybrid simulation for cyber physical systems: A panel on where are we going regarding complexity, intelligence, and adaptability of cps using simulation," in *Proceedings of the Symposium on Modeling and Simulation of Complexity in Intelligent, Adaptive and Autonomous Systems*, MSCIAAS '18, (San Diego, CA, USA), pp. 1–19, Society for Computer Simulation International, 2018.

[65] S. Mittal and A. Tolk, *Complexity challenges in cyber physical systems: Using modeling and simulation (M&S) to support intelligence, adaptation and autonomy*. Wiley, 2019.

[66] W. D. Lin, Y. H. Low, Y. T. Chong, and C. L. Teo, "Integrated cyber physical simulation modelling environment for manufacturing 4.0," in *2018 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pp. 1861–1865, Dec 2018.

[67] H. Tu, Y. Xia, J. Wu, and X. Zhou, "Robustness assessment of cyber-physical systems with weak interdependency," *Physica A: Statistical Mechanics and its Applications*, vol. 522, Feb 2019.

[68] D. Khalyeyev, P. Hnetynka, and T. Bures, "A virtual playground for testing smart cyber-physical systems," in *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pp. 85–88, Apr 2018.

[69] O. Cardin, "Classification of cyber-physical production systems applications: Proposition of an analysis framework," *Computers in Industry*, vol. 104, pp. 11–21, 2019.

[70] A. Rahatulain and M. Onori, "Viewpoints and views for the architecture description of cyber-physical manufacturing systems," *Procedia CIRP*, vol. 72, pp. 450–455, 2018. 51st CIRP Conference on Manufacturing Systems.

[71] M. Kit, I. Gerostathopoulos, T. Bures, P. Hnetynka, and F. Plasil, "An architecture framework for experimentations with self-adaptive cyber-physical systems," in *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 93–96, May 2015.

[72] I. Ruchkin, "Towards integration of modeling methods for cyber-physical systems," in *CEUR Workshop Proceedings*, vol. 1531, 2015.

[73] J. Brings, M. Daun, T. Bandyszak, V. Stricker, T. Weyer, E. Mirzaei, M. Neumann, and J. S. Zernickel, "Model-based documentation of dynamicity constraints for collaborative cyber-physical system architectures: Findings from an industrial case study," *Journal of Systems Architecture - Embedded Systems Design*, vol. 97, pp. 153–167, 2019.

[74] T. Guan and G. Yang, "Integration-oriented modeling of cyber-physical interactive process," in *2013 IEEE International Conference on Green Computing and Communications (GreenCom) and IEEE Internet of Things(iThings) and IEEE Cyber, Physical and Social Computing(CPSCom)*, (Los Alamitos, CA, USA), pp. 1492–1495, IEEE Computer Society, Aug 2013.

[75] Y. Liu, "Toward a unified object model for cyber-physical systems," *Proceedings - International Conference on Software Engineering*, Jan 2011.

[76] M. Bunting, Y. Zeleke, K. McKeever, and J. Sprinkle, "A safe autonomous vehicle trajectory domain specific modeling language for non-expert development," in *DSM 2016 - Proceedings of the International Workshop on Domain-Specific Modeling, co-located with SPLASH 2016*, pp. 42–48, 2016.

[77] J.-M. Thiriet and S. Mocanu, "Some considerations on dependability issues and cyber-security of cyber-physical systems," in *2018 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, pp. 1–6, 2018.

[78] T. Sanislav, G. Mois, and L. Miclea, "An approach to model dependability of cyber-physical systems," *Microprocessors and Microsystems*, vol. 41, pp. 67–76, 2016.

[79] Y. Ma, D. Gunatilaka, B. Li, H. Gonzalez, and C. Lu, "Holistic cyber-physical management for dependable wireless control systems," *ACM Trans. Cyber-Phys. Syst.*, vol. 3, Sep 2018.

[80] H. Bayanifar and H. Kühnle, "Enhancing dependability and security of cyber-physical production systems," *IFIP Advances in Information and Communication Technology*, vol. 499, pp. 135–143, 2017.

[81] J. Fitzgerald, C. Gamble, R. Payne, P. G. Larsen, S. Basagiannis, and A. E.-D. Mady, "Collaborative model-based systems engineering for cyber-physical systems, with a building automation case study," *INCOSE International Symposium*, vol. 26, no. 1, pp. 817–832, 2016.

[82] Technical Operations INCOSE, "Systems engineering vision 2020," Tech. Rep. INCOSE-TP-2004-004-02, International Council on Systems Engineering (INCOSE), Sep 2007.

[83] INCOSE, "Systems engineering handbook - a guide for system life cycle processes and activities, version 3.2.2," in *International Council on Systems Engineering (INCOSE)*, 2011.

[84] INCOSE, *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. Wiley, Jul 2015.

[85] S. Friedenthal, R. Griego, and M. Sampson, "Incose model based systems engineering (mbse) initiative," *INCOSE*, Jan 2009.

[86] V. Gunes, S. Peter, T. Givargis, and F. Vahid, “A survey on concepts, applications, and challenges in cyber-physical systems,” *KSII Transactions on Internet and Information Systems*, vol. 8, pp. 4242–4268, Dec 2014.

[87] M. V. Cengarle, S. Bensalem, J. McDermid, R. Passerone, A. Sangiovanni-Vincentelli, and M. Törngren, “Characteristics, capabilities, potential applications of cyber-physical systems: a preliminary analysis,” tech. rep., CyPhERS Deliverable, D2.1, Nov 2013.

[88] H. Gill, “From vision to reality: cyber-physical systems,” in *Presentation, HCSS National Workshop on New Research Directions for High Confidence Transportation CPS: Automotive, Aviation and Rail*, 2008.

[89] E. Lee and S. Seshia, *Introduction to Embedded Systems, A Cyber-Physical Systems Approach.* University of Berkley: http://LeeSeshia.org, 2011. ISBN 978-0-557-70857-4.

[90] E. Geisberger and M. Broy, *agendaCPS: Integrierte Forschungsagenda Cyber-Physical Systems*, vol. 1. Springer-Verlag, 2012.

[91] E. A. Lee, “CPS foundations,” in *Proceedings of the 47th Design Automation Conference*, DAC ’10, (New York, NY, USA), pp. 737–742, ACM, 2010.

[92] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, “Cyber-physical systems: The next computing revolution,” in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pp. 731–736, 2010.

[93] E. O. Méndez and S. Ren, “Design of cyber-physical interface for automated vital signs reading in electronic medical records systems,” in *2012 IEEE International Conference on Electro/Information Technology*, pp. 1–10, May 2012.

[94] J. Wang, H. Abid, S. Lee, L. Shu, and F. Xia, “A secured health care application architecture for cyber-physical systems,” *Control Engineering and Applied Informatics*, vol. 13, Dec 2011.

[95] G. S. Avrunin, L. A. Clarke, L. J. Osterweil, J. M. Goldman, and T. Rausch, “Smart checklists for human-intensive medical systems,” in *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)*, pp. 1–6, Jun 2012.

[96] S. Haque, S. Aziz, and M. Rahman, “Review of cyber-physical system in healthcare,” *International Journal of Distributed Sensor Networks*, vol. 2014, p. 20, Apr 2014.

[97] A. Colombo, S. Karnouskos, O. Kaynak, Y. Shi, and S. Yin, “Industrial cyberphysical systems: A backbone of the fourth industrial revolution,” *IEEE Industrial Electronics Magazine*, vol. 11, pp. 6–16, Mar 2017.

[98] H. Haskamp, F. Orth, J. Wermann, and A. W. Colombo, “Implementing an opc ua interface for legacy plc-based automation systems using the azure cloud: An icps-architecture with a retrofitted rfid system,” in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pp. 115–121, May 2018.

[99] X. Yao, J. Zhou, Y. Lin, Y. Li, H. Yu, and Y. Liu, “Smart manufacturing based on cyber-physical systems and beyond,” *Journal of Intelligent Manufacturing*, Dec 2017.

[100] J.-R. Jiang, “An improved cyber-physical systems architecture for industry 4.0 smart factories,” *Advances in Mechanical Engineering*, vol. 10, no. 6, 2018.

[101] F. Tao, Q. Qi, L. Wang, and A. Nee, “Digital twins and cyber-physical systems toward smart manufacturing and industry 4.0: Correlation and comparison,” *Engineering*, vol. 5, no. 4, pp. 653–661, 2019.

[102] P. Tsakalides, A. Panousopoulou, G. Tsagkatakis, and L. Montestruque, eds., *Smart Water Grids: A Cyber-Physical Systems Approach.* CRC Press, Apr 2018.

[103] Z. Wang, D. Watkins, K. Ong, and X. Shi, “Cyber-physical systems for water sustainability: Challenges and opportunities,” *IEEE Communications Magazine*, vol. 53, May 2015.

[104] R. Taormina, S. Galelli, N. O. Tippenhauer, E. Salomons, and A. Ostfeld, “Characterizing cyber-physical attacks on water distribution systems,” *Journal of Water Resources Planning and Management*, vol. 143, Feb 2017.

[105] B. Syed, A. Pal, K. Srinivasarengan, and P. Balamuralidhar, “A smart transport application of cyber-physical systems: Road surface monitoring with mobile devices,” in *2012 Sixth International Conference on Sensing Technology (ICST)*, pp. 8–12, Dec 2012.

[106] Z. Wang, Y. Zhang, and K. Du, “Cyber-physical traffic systems: Architecture and implementation techniques,” in *Advances in Wireless Sensor Networks* (R. Wang and F. Xiao, eds.), (Berlin, Heidelberg), pp. 490–500, Springer Berlin Heidelberg, 2013.

[107] K. Dey, R. Fries, and S. Ahmed, “11 - future of transportation cyber-physical systems - smart cities/regions,” in *Transportation Cyber-Physical Systems* (L. Deka and M. Chowdhury, eds.), pp. 267–307, Elsevier, 2018.

[108] W. Mi, N. Zhou, and Y. Jin, “A real-time cyber-physical system for indoor environment monitoring,” *DEStech Transactions on Computer Science and Engineering*, Mar 2017.

[109] T. Sanislav, G. Mois, S. Folea, L. Miclea, G. Gambardella, and P. Prinetto, "A cloud-based cyber-physical system for environmental monitoring," in *2014 3rd Mediterranean Conference on Embedded Computing (MECO)*, pp. 6–9, Jun 2014.

[110] G. Mois, T. Sanislav, and S. C. Folea, "A cyber-physical system for environmental monitoring," *IEEE Transactions on Instrumentation and Measurement*, vol. 65, pp. 1463–1471, Jun 2016.

[111] E. Gelenbe, G. Gorbil, and F. Wu, "Emergency cyber-physical-human systems," in *2012 21st International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–7, Jul 2012.

[112] V. V. Estrela, J. Hemanth, O. Saotome, E. G. H. Grata, and D. R. F. Izario, "Emergency response cyber-physical system for flood prevention with sustainable electronics," in *Proceedings of the 3rd Brazilian Technology Symposium*, (Cham), pp. 319–328, Springer International Publishing, 2019.

[113] J. Zander, P. J. Mosterman, T. Padir, Y. Wan, and S. Fu, "Cyber-physical systems can make emergency response smart," *Procedia Engineering*, vol. 107, pp. 312–318, 2015. Humanitarian Technology: Science, Systems and Global Impact 2015, HumTech2015.

[114] Y. Yu, Y. Liu, and C. Qin, "Basic ideas of the smart grid," *Engineering*, vol. 1, no. 4, pp. 405–408, 2015.

[115] Z. Xue-song, C. Li-qiang, and M. You-jie, "Research on smart grid technology," in *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, vol. 3, pp. 599–603, Oct 2010.

[116] IEEE Smart Grid R&D Committee Members, "IEEE Smart Grid Survey Structure of Emerging Technologies," tech. rep., IEEE Smart Grid, Feb 2017.

[117] D. Goswami, R. Schneider, A. Masrur, M. Lukasiewycz, S. Chakraborty, H. Voit, and A. Annaswamy, "Challenges in automotive cyber-physical systems design," in *2012 International Conference on Embedded Computer Systems (SAMOS)*, pp. 346–354, Jul 2012.

[118] S. Chakraborty, M. A. Al Faruque, W. Chang, D. Goswami, M. Wolf, and Q. Zhu, "Automotive cyber-physical systems: A tutorial introduction," *IEEE Design Test*, vol. 33, pp. 92–108, Aug 2016.

[119] K. Sampigethaya and R. Poovendran, "Aviation cyber-physical systems: Foundations for future aircraft and air transport," *Proceedings of the IEEE*, vol. 101, pp. 1834–1855, Aug 2013.

[120] E. Atkins, "Cyber-physical aerospace: Challenges and future directions in transportation and exploration systems," tech. rep., CMU/SEI-2011-TR-002, Jan 2006.

[121] E. M. Atkins and J. M. Bradley, “Aerospace cyber-physical systems education,” in *AIAA Infotech@Aerospace (I@A) Conference*, American Institute of Aeronautics and Astronautics, Aug. 2013.

[122] L. Zhang, “Modeling railway cyber physical systems based on aadl,” in *2013 19th International Conference on Automation and Computing*, pp. 1–6, Sep. 2013.

[123] B. Chen, C. Schmittner, Z. Ma, W. G. Temple, X. Dong, D. L. Jones, and W. H. Sanders, “Security analysis of urban railway systems: The need for a cyber-physical perspective,” in *Computer Safety, Reliability, and Security*, (Cham), pp. 277–290, Springer International Publishing, 2015.

[124] M. Brinkmann and A. Hahn, “Testbed architecture for maritime cyber physical systems,” in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, pp. 923–928, Jul 2017.

[125] T. Yang, H. Feng, J. Zhao, R. Deng, Y. Wang, and Z. Su, “Genetic optimization-based scheduling in maritime cyber physical systems,” *International Journal of Distributed Sensor Networks*, vol. 13, Jul 2017.

[126] R. A. Iureva, A. S. Kremlev, A. A. Margun, S. M. Vlasov, S. D. Vasilkov, A. V. Penskoi, D. E. Konovalov, and P. Y. Korepanov, “Interdisciplinary approach to cyber-physical systems training,” in *ICINCO*, 2019.

[127] F. Slomka, S. Kollmann, S. Moser, and K. Kempf, “A multidisciplinary design methodology for cyber-physical systems,” in *Proc. 4th Int. Workshop Model Based Arch. Construction Embedded System*, 2011.

[128] P. Marwedel, *Embedded System Design*. Kluwer Academic Publishers, 2003.

[129] P. Marwedel, “Embedded and cyber-physical systems in a nutshell,” *47th DAC*, Jan 2010.

[130] E. A. Lee, “The future of embedded software,” May 2006. ARTEMIS 2006 Annual Conference.

[131] G. C. da Silva and P. C. Kaminski, “From embedded systems (es) to cyber-physical systems (cps): An analysis of transitory stage of automotive manufacturing in the industry 4.0 scenario,” in *SAE Technical Paper*, SAE International, Oct 2016.

[132] I. Colak, “Introduction to smart grid,” in *2016 International Smart Grid Workshop and Certificate Program (ISGWCP)*, pp. 1–5, Mar 2016.

[133] M. F. Golnaraghi, *Automatic control systems*. New York: McGraw-Hill Education, 2017.

[134] L. Atzori, A. Iera, and G. Morabito, “The internet of things: A survey,” *Computer Networks*, pp. 2787–2805, Dec 2010.

[135] T. L. Koreshoff, T. Robertson, and T. W. Leong, “Internet of things: A review of literature and products,” in *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*, OzCHI ’13, (New York, NY, USA), pp. 335–344, ACM, 2013.

[136] A. Whitmore, A. Agarwal, and L. Xu, “The internet of things-a survey of topics and trends,” *Information Systems Frontiers*, vol. 17, Apr 2014.

[137] C. Shih, J. Chou, N. Reijers, and T. Kuo, “Designing cps/iot applications for smart buildings and cities,” *IET Cyber-Physical Systems: Theory Applications*, vol. 1, no. 1, pp. 3–12, 2016.

[138] J. E. I. Esquer, F. F. González-Navarro, B. L. F. Ríos, L. Burtseva, and M. A. A. Vargas, “Tracking the evolution of the internet of things concept across different application domains,” in *Sensors*, 2017.

[139] C. Greer, M. Burns, D. Wollman, and E. Griffor, “Cyber-physical systems and internet of things,” tech. rep., National Institute of Standards and Technology, 2019.

[140] F. Mattern and C. Floerkemeier, *From the Internet of Computers to the Internet of Things*, pp. 242–259. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[141] X. Zhang and M. Ali, “A bean optimization-based cooperation method for target searching by swarm uavs in unknown environments,” *IEEE Access*, vol. 8, pp. 43850–43862, 2020.

[142] J. Hultman, A. Nyberg, and M. Svensson, “A software architecture for autonomous systems,” in *Proceedings of the 6th International Symposium on Unmanned Untethered Submersible Technology,*, pp. 279–292, 1989.

[143] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[144] P. Checkland, *Systems Thinking, Systems Practice: Includes a 30-year retrospective*. Chichester New York: John Wiley, 1999.

[145] P. Zhang, “Chapter 18 - industrial control system operation routines,” in *Advanced Industrial Control Technology* (P. Zhang, ed.), pp. 735–779, Oxford: William Andrew Publishing, 2010.

[146] A. Chhokra, N. Mahadevan, A. Dubey, and G. Karsai, “Qualitative fault modeling in safety critical cyber physical systems,” in *Proceedings of the 12th System Analysis and Modelling Conference*, SAM ’20, (New York, NY, USA), p. 128–137, Association for Computing Machinery, 2020.

[147] G. A. Bekey, “Autonomous robots - from biological inspiration to implementation and control,” in *Intelligent robotics and autonomous agents*, 2005.

[148] C. Santos, A. Mehrsai, A. Barros, M. Araújo, and E. Ares, “Towards industry 4.0: an overview of european strategic roadmaps,” *Procedia Manufacturing*, vol. 13, pp. 972–979, 2017. Manufacturing Engineering Society International Conference 2017, MESIC 2017, 28-30 June 2017, Vigo (Pontevedra), Spain.

[149] J. Lee, B. Bagheri, and H.-A. Kao, “A cyber-physical systems architecture for industry 4.0-based manufacturing systems,” *SME Manufacturing Letters*, vol. 3, Dec 2014.

[150] H. Kagermann, W. Wahlster, and J. Helbig, “Recommendations for implementing the strategic initiative industrie 4.0 – securing the future of german manufacturing industry,” final report of the industrie 4.0 working group, acatech – National Academy of Science and Engineering, Apr 2013.

[151] J. Herwan, S. Kano, R. Oleg, H. Sawada, and N. Kasashima, “Cyber-physical system architecture for machining production line,” in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pp. 387–391, May 2018.

[152] H. Akillioglu and M. Onori, “Evolvable production systems and impacts on production planning,” in *2011 IEEE International Symposium on Assembly and Manufacturing (ISAM)*, pp. 1–6, May 2011.

[153] H. Hibi, “An autonomous production system that coexists harmoniously with human-development of autonomous mobile robot system,” in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 2, pp. 2865–2870, Sep. 2003.

[154] Y. Bai and M. Hsueh, “Using an adaptive iterative learning algorithm for planning of the path of an autonomous robotic vacuum cleaner,” in *The 1st IEEE Global Conference on Consumer Electronics 2012*, pp. 401–405, Oct 2012.

[155] W. H. C. Wickramaarachchi, M. A. P. Chamikara, and R. A. C. H. Ratnayake, “Towards implementing efficient autonomous vacuum cleaning systems,” in *2017 IEEE International Conference on Industrial and Information Systems (ICIIS)*, pp. 1–6, Dec 2017.

[156] S. Vallor, “The future of military virtue: Autonomous systems and the moral deskilling of the military,” in *2013 5th International Conference on Cyber Conflict (CYCON 2013)*, pp. 1–15, Jun 2013.

[157] G. de Boisboissel, “Is it sensible to grant autonomous decision-making to military robots of the future?,” in *2017 International Conference on Military Technologies (ICMT)*, pp. 738–742, May 2017.

[158] F. Li and Y. Du, “From alphago to power system ai: What engineers can learn from solving the most complex board game,” *IEEE Power and Energy Magazine*, vol. 16, pp. 76–84, Mar 2018.

[159] L. P. Robert, “Are automated vehicles safer than manually driven cars?,” *AI & SOCIETY*, vol. 34, pp. 687–688, Sep 2019.

[160] Zhenxing Zhao, Congying Gao, and Fu Duan, “A survey on autonomic computing research,” in *2009 Asia-Pacific Conference on Computational Intelligence and Industrial Applications (PACIIA)*, vol. 2, pp. 288–291, Nov 2009.

[161] Y. Bamberger, J. Baptista, R. Belmans, B. Buchholz, M. Chebbo, J. Doblado, V. Efthymiou, L. Gallo, E. Handschin, N. Hatziargyriou, N. Jenkins, T. Kapetanovic, U. Keussen, N. Leffler, M. Moscoso-Osterkorn, P. Nabuurs, J. Østergaard, C. Sabelli, N. Elustondo, P. Smith, and M. Wasiluk-Hassa, *Vision and Strategy for Europe’s Electricity Networks of the Future: European Technology PlatformSmart-Grids*. Office for Official Publications of the European Communities, 2006.

[162] D. Kaur, S. N. Islam, M. A. Mahmud, and Z. Dong, “Energy forecasting in smart grid systems: A review of the state-of-the-art techniques,” *CoRR*, vol. abs/2011.12598, 2020.

[163] Ran Li, Xiaobo Zhang, Qi Zhao, and Guowei Liu, “Economic optimization of self-healing control of power grid based on multi-agent system,” in *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pp. 1334–1337, Oct 2016.

[164] A. Khair and M. Rihan, “Efficient detection of islanding for self-healing grid-a case study in indian grid,” in *2018 4th International Conference on Computational Intelligence Communication Technology (CICT)*, pp. 1–5, Feb 2018.

[165] H. Wang, H. Zhang, L. Chen, L. Chen, Y. Zhao, and B. Duan, “A method of distribution reconfiguration with micro grid considering dynamic behavior of thermal loads,” in *2018 International Conference on Smart Grid and Electrical Automation (ICSGEA)*, pp. 1–5, Jun 2018.

[166] H. Chen, H. Leng, H. Tang, J. Zhu, H. Gong, and H. Zhong, “Research on model management method for micro-grid,” in *2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pp. 163–166, Dec 2017.

[167] S. Zhao, J. Dang, and H. Ma, “Research of china high-speed train transport operation system based on autonomous integration,” in *2009 International Symposium on Autonomous Decentralized Systems*, pp. 1–5, Mar 2009.

[168] M. Matsumoto and N. Kitamura, “Autonomous decentralized train control technology,” in *2009 International Symposium on Autonomous Decentralized Systems*, pp. 1–5, Mar 2009.

[169] A. G. S. Júnior, A. P. D. Araújo, M. V. A. Silva, R. V. Aroca, and L. M. G. Gonçalves, "N-boat: An autonomous robotic sailboat," in *2013 Latin American Robotics Symposium and Competition*, pp. 24–29, Oct 2013.

[170] T. Yang, C. H. Foh, F. Heliot, C. Y. Leow, and P. Chatzimisios, "Self-organization drone-based unmanned aerial vehicles (uav) networks," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pp. 1–6, May 2019.

[171] S. El Hamdani and N. Benamar, "Autonomous traffic management: Open issues and new directions," in *2018 International Conference on Selected Topics in Mobile and Wireless Networking (MoWNeT)*, pp. 1–5, Jun 2018.

[172] H. March and R. Ribera-Fumaz, "Smart contradictions: The politics of making barcelona a self-sufficient city," *European Urban and Regional Studies*, vol. 23, no. 4, pp. 816–830, 2016.

[173] R. Petrolo, V. Loscri, and N. Mitton, *Cyber-Physical Objects as Key Elements for a Smart Cyber-City*, pp. 31–49. Cham: Springer International Publishing, 2016.

[174] O. Younis and N. Moayeri, "Employing cyber-physical systems: Dynamic traffic light control at road intersections," *IEEE Internet of Things Journal*, vol. 4, pp. 2286–2296, Dec 2017.

[175] A. A. Ghaemi, "A cyber-physical system approach to smart city development," in *2017 IEEE International Conference on Smart Grid and Smart Cities (ICSGSC)*, pp. 257–262, Jul 2017.

[176] J. Y. Kim, H. Lee, J. Son, and J. Park, "Smart home web of objects-based iot management model and methods for home data mining," in *2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 327–331, Aug 2015.

[177] D. Schweizer, M. Zehnder, H. Wache, H. Witschel, D. Zanatta, and M. Rodriguez, "Using consumer behavior data to reduce energy consumption in smart homes: Applying machine learning to save energy without lowering comfort of inhabitants," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pp. 1123–1129, Dec 2015.

[178] T. Liang, B. Zeng, J. Liu, L. Ye, and C. Zou, "An unsupervised user behavior prediction algorithm based on machine learning and neural network for smart home," *IEEE Access*, vol. 6, pp. 49237–49247, 2018.

[179] W. Li, T. Logenthiran, V. Phan, and W. L. Woo, "Implemented iot-based self-learning home management system (shms) for singapore," *IEEE Internet of Things Journal*, vol. 5, pp. 2212–2219, Jun 2018.

[180] L. Yang, N. Elisa, and N. Eliot, “Chapter 7 - privacy and security aspects of e-government in smart cities,” in *Smart Cities Cybersecurity and Privacy* (D. B. Rawat and K. Z. Ghafoor, eds.), pp. 89–102, Elsevier, 2019.

[181] Q. Huang, L. Wang, and Y. Yang, “Secure and privacy-preserving data sharing and collaboration in mobile healthcare social networks of smart cities,” *Security and Communication Networks*, vol. 2017, pp. 1–12, 08 2017.

[182] R. Saini and D. Mishra, “Chapter 4 - privacy-aware physical layer security techniques for smart cities,” in *Smart Cities Cybersecurity and Privacy* (D. B. Rawat and K. Z. Ghafoor, eds.), pp. 39–56, Elsevier, 2019.

[183] S. Kim and S. Park, “CPS(Cyber Physical System) based Manufacturing System Optimization,” *Procedia Computer Science*, vol. 122, pp. 518 – 524, 2017. 5th International Conference on Information Technology and Quantitative Management, ITQM 2017.

[184] K. Sharma, “20 - information technology-operation technology convergence,” in *Overview of Industrial Process Automation (Second Edition)* (K. Sharma, ed.), pp. 359–375, Elsevier, second edition ed., 2017.

[185] R. Schoop, R. Neubert, and A. W. Colombo, “A multiagent-based distributed control platform for industrial flexible production systems,” in *IECON’01. 27th Annual Conference of the IEEE Industrial Electronics Society (Cat. No.37243)*, vol. 1, pp. 279–284, Nov 2001.

[186] P. L. ao, “Agent-based distributed manufacturing control: A state-of-the-art survey,” *Engineering Applications of Artificial Intelligence*, vol. 22, no. 7, pp. 979–991, 2009. Distributed Control of Production Systems.

[187] L. Ribeiro, J. Barata, and A. Colombo, “MAS and SOA: A Case Study Exploring Principles and Technologies to Support Self-Properties in Assembly Systems,” in *2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, pp. 192–197, Oct 2008.

[188] A. W. Colombo, S. Karnouskos, J. ao Marco Mendes, and P. L. ao, “Chapter 4 - industrial agents in the era of service-oriented architectures and cloud-based industrial infrastructures,” in *Industrial Agents* (P. L. ao and S. Karnouskos, eds.), pp. 67–87, Boston: Morgan Kaufmann, 2015.

[189] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, and K. Ueda, “Cyber-physical systems in manufacturing,” *CIRP Annals*, vol. 65, no. 2, pp. 621–641, 2016.

[190] L. Monostori, “Cyber-physical production systems: Roots, expectations and r&d challenges,” *Procedia CIRP*, vol. 17, pp. 9–13, 2014. Variety Management in Manufacturing.

[191] N. Kalra and S. Paddock, “Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?,” *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, Dec 2016.

[192] S. Chen and F. Chen, “Simulation-based assessment of vehicle safety behavior under hazardous driving conditions,” *Journal of Transportation Engineering-asce - J TRANSP ENG-ASCE*, vol. 136, Apr 2010.

[193] X. Lyu, Y. Ding, and S.-H. Yang, “Safety and security risk assessment in cyber-physical systems,” *IET Cyber-Physical Systems: Theory & Applications*, vol. 4, no. 3, pp. 221–232, 2019.

[194] J. Harrison, *Handbook of practical logic and automated reasoning*. Cambridge New York: Cambridge University Press, 2009.

[195] M. U. Sanwal and O. Hasan, “Formal verification of cyber-physical systems: Coping with continuous elements,” in *Computational Science and Its Applications – ICCSA 2013* (B. Murgante, S. Misra, M. Carlini, C. M. Torre, H.-Q. Nguyen, D. Taniar, B. O. Apduhan, and O. Gervasi, eds.), (Berlin, Heidelberg), pp. 358–371, Springer Berlin Heidelberg, 2013.

[196] L. Dennis, M. Fisher, M. Slavkovik, and M. Webster, “Formal verification of ethical choices in autonomous systems,” *Robotics and Autonomous Systems*, vol. 77, pp. 1–14, 2016.

[197] S. Khastgir, S. Birrell, G. Dhadyalla, and P. Jennings, “Development of a drive-in driver-in-the-loop fully immersive driving simulator for virtual validation of automotive systems,” in *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*, pp. 1–4, May 2015.

[198] K. J. Barltrop, K. H. Friberg, and G. A. Horvath, “Automated generation and assessment of autonomous systems test cases,” in *2008 IEEE Aerospace Conference*, pp. 1–10, Mar 2008.

[199] A. Kleyner, “How stress variance in the automotive environment will affect a ‘true’ value of the reliability demonstrated by accelerated testing,” *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, vol. 7, pp. 552–559, May 2014.

[200] M. Althoff and S. Lutz, “Automatic generation of safety-critical test scenarios for collision avoidance of road vehicles,” in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1326–1333, Jun 2018.

[201] E. Vassev, *Autonomy Requirements Engineering for Space Missions (NASA Monographs in Systems and Software Engineering)*. Springer, Aug 2014.

[202] J. Kephart and D. Chess, “The vision of autonomic computing,” *Computer*, vol. 36, pp. 41–50, Feb 2003.

[203] J. Albus and P. J. Antsaklis, “Panel discussion: Autonomy in engineering systems: What is it and why is it important? setting the stage: Some autonomous thoughts on autonomy,” in *Proceedings of the 1998 IEEE International Symposium on Intelligent Control (ISIC) held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA) Intell*, pp. 520–521, Sep. 1998.

[204] T. B. Sheridan, “Adaptive automation, level of automation, allocation authority, supervisory control, and adaptive control: Distinctions and modes of adaptation,” *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 41, pp. 662–667, Jul 2011.

[205] R. Parasuraman, T. B. Sheridan, and C. D. Wickens, “A model for types and levels of human interaction with automation,” *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 30, pp. 286–297, May 2000.

[206] T. Sheridan, W. L. Verplank, and T. L. Brooks, “Human and computer control of undersea teleoperators,” *Department of Mechanical Engineering, MIT*, Dec 1978.

[207] SAE, *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, Sep 2016.

[208] J. Frohm, V. Lindström, M. Winroth, and J. Stahre, “Levels of automation in manufacturing,” *Ergonomia - International Journal of Ergonomics and Human Factors*, vol. 30, pp. 181–207, Jan 2008.

[209] P. J. Durst, W. Gray, A. Nikitenko, J. Caetano, M. Trentini, and R. King, “A framework for predicting the mission-specific performance of autonomous unmanned systems,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1962–1969, Sep. 2014.

[210] J. Diebold, *Automation: the advent of the automatic factory*. Van Nostrand, 1952.

[211] W. Hilzinger, *Informational Aspects of Automation*. Michigan State University of Agriculture and Applied Science. School of Mechanical Engineering, 1955.

[212] G. Amber and P. Amber, *Anatomy of automation*. Prentice-Hall, 1962.

[213] M. R. Endsley, “Level of automation: Integrating humans and automated systems,” *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 41, no. 1, pp. 200–204, 1997.

[214] C. E. Billings, *Aviation automation : the search for a human-centered approach*. Mahwah, N.J. : Lawrence Erlbaum Associates Publishers, 1997. Includes indexes.

[215] R. M. Marsh and H. Mannari, "Technology and size as determinants of the organizational structure of japanese factories," *Administrative Science Quarterly*, vol. 26, no. 1, pp. 33–57, 1981.

[216] P. Antsaklis, K. Passino, and S. Wang, "An introduction to autonomous control systems," *IEEE Control Systems Magazine*, vol. 11, no. 4, pp. 5–13, 1991.

[217] G. Di Marzo Serugendo, M.-P. Gleizes, and A. Karageorgos, *Self-organising Systems*, pp. 7–32. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[218] D. Ye, M. Zhang, and A. V. Vasilakos, "A survey of self-organization mechanisms in multiagent systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, pp. 441–461, Mar 2017.

[219] M. S. Innocente and P. Grasso, "Self-organising swarms of firefighting drones: Harnessing the power of collective intelligence in decentralised multi-robot systems," *Journal of Computational Science*, vol. 34, pp. 80–101, 2019.

[220] ISO, "ISO/IEC/IEEE International Standard - Systems and software engineering – Taxonomy of systems of systems," *ISO/IEC/IEEE 21841:2019(E)*, pp. 1–20, Jul 2019.

[221] D. Kemp, D. Camm, R. Evans, and J. Elphick, "2.1.1 steampunk system of systems engineering: A case study of successful system of systems engineering in 19th century britain," *INCOSE International Symposium*, vol. 23, no. 1, pp. 617–633, 2013.

[222] J. Dahmann and M. Henshaw, "Introduction to systems of systems engineering," *INSIGHT*, vol. 19, no. 3, pp. 12–16, 2016.

[223] N. J. Nilsson, "Teleo-reactive programs for agent control," *J. Artif. Int. Res.*, vol. 1, p. 139–158, Jan 1994.

[224] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, pp. 11–33, 2004.

[225] J. van Amerongen, *Dynamical Systems for Creative Technology*. Controllab Products, Enschede, Netherlands, 2010.

[226] J. Holt and S. Perry, *SysML for Systems Engineering*. IET, 2008.

[227] R. Stevens, P. Brook, K. Jackson, and S. Arnold, *System Engineering – Coping with Complexity*, vol. ISBN 0-13-095085-8. Pearson Education, 1998.

[228] S. Schach, “Software engineering with java, richard d,” *Irwin, USA*, 1997.

[229] R. S. Pressman, *Software engineering: a practitioner’s approach*. Palgrave Macmillan, 2005.

[230] J. Sanders and E. Curran, *Software quality: a framework for success in software development and support*. ACM Press/Addison-Wesley Publishing Co., 1994.

[231] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified modeling language reference manual, the*. Pearson Higher Education, 2004.

[232] M. Maier, *The art of systems architecting*. Boca Raton: CRC Press, 2009.

[233] ISO, IEC, “15288: 2008. Systems and software engineering-System life cycle processes,” *2nd International Organization for Standardization/International Electrotechnical Commission*, 2008.

[234] D. Ota and M. Gerz, “Benefits and challenges of architecture frameworks,” in *Sixteenth International Command and Control Research and Technology Symposium*, p. 40, Jun 2011.

[235] J. Holt, *Modelling enterprise architectures*. The Institution of Engineering and Technology, 2010.

[236] J. A. Zachman, “Concise definition of the zachman framework,” *Zachman International*, 2008.

[237] B. Biggs, “Ministry of defence architectural framework (modaf),” in *IEE Seminar on UML Systems Engineering*, pp. 43–82, Jan 2005.

[238] DoD Architecture Framework Working Group and others, “The dodaf architecture framework version 2.02,” *Department of Defense*, 2010.

[239] V. Haren, *TOGAF Version 9.1*. Van Haren Publishing, 2011.

[240] J. Holt and S. Perry, *SysML for Systems Engineering, 2nd Edition: A Model-Based Approach*. IET, 2 ed., 2013.

[241] J. Holt, S. Perry, and M. Brownsword, *Model-based requirements engineering*, vol. 9. IET, 2012.

[242] J. Krogstie, G. Sindre, and H. Jørgensen, “Process models representing knowledge for action: a revised quality framework,” *European Journal of Information Systems*, vol. 15, pp. 91–102, Feb 2006.

[243] S. Friedenthal, A. Moore, and R. Steiner, “Chapter 3 - getting started with sysml,” in *A Practical Guide to SysML (Second Edition)* (S. Friedenthal, A. Moore, and R. Steiner, eds.), The MK/OMG Press, pp. 29–49, Boston: Morgan Kaufmann, second edition ed., 2012.

[244] S. Bonnet, J. Voirin, D. Exertier, and V. Normand, “Not (strictly) relying on sysml for mbse: Language, tooling and development perspectives: The arcadia/capella rationale,” in *2016 Annual IEEE Systems Conference (SysCon)*, pp. 1–6, Apr 2016.

[245] K. Evensen and K. Weiss, “A comparison and evaluation of real-time software systems modeling languages,” in *International Journal of Software Engineering and its Applications*, Apr 2010.

[246] P. De Saqui-Sannes and J. Hugues, “Combining SysML and AADL for the Design, Validation and Implementation of Critical Systems,” in *ERTS2 2012*, (Toulouse, France), p. 117, Feb. 2012.

[247] R. Behjati, T. Yue, S. Nejati, L. Briand, and B. Selic, “Extending sysml with aadl concepts for comprehensive system architecture modeling,” in *Modelling Foundations and Applications* (R. B. France, J. M. Kuester, B. Bordbar, and R. F. Paige, eds.), (Berlin, Heidelberg), pp. 236–252, Springer Berlin Heidelberg, 2011.

[248] L. S. Committee *et al.*, “Lifecycle modeling language (lml) specification,” *Lifecycle Modeling Language website*, 2015.

[249] W. K. Vaneman, “Enhancing model-based systems engineering with the lifecycle modeling language,” in *2016 Annual IEEE Systems Conference (SysCon)*, pp. 1–7, Apr 2016.

[250] W. W. Royce, “Managing the development of large software systems: Concepts and techniques,” in *Proceedings of the 9th International Conference on Software Engineering*, ICSE ’87, (Los Alamitos, CA, USA), pp. 328–338, IEEE Computer Society Press, 1987.

[251] B. W. Boehm, “A spiral model of software development and enhancement,” *Computer*, vol. 21, pp. 61–72, May 1988.

[252] K. Forsberg and H. Mooz, “4.4.4 application of the ‘vee’ to incremental and evolutionary development,” *INCOSE International Symposium*, vol. 5, no. 1, pp. 848–855, 1995.

[253] A. Alshamrani, A. Bahattab, and I. Fulton, “A comparison between three sdlc models waterfall model, spiral model, and incremental/iterative model,” in *2015 International Journal of Computer Science Issues*, 2015.

[254] K. Petersen, C. Wohlin, and D. Baca, “The waterfall model in large-scale development,” in *Product-Focused Software Process Improvement* (F. Bomarius, M. Oivo, P. Jaring, and P. Abrahamsson, eds.), (Berlin, Heidelberg), pp. 386–400, Springer Berlin Heidelberg, 2009.

[255] G. Pomberger, *Boehm’s Spiral Model Revisited*, pp. 89–98. Wiesbaden: DUV, 2006.

[256] A. Aerts, M. Reniers, and M. Mousavi, "Chapter 19 - model-based testing of cyber-physical systems," in *Cyber-Physical Systems* (H. Song, D. B. Rawat, S. Jeschke, and C. Brecher, eds.), Intelligent Data-Centric Systems, pp. 287–304, Boston: Academic Press, 2017.

[257] J. Jensen, D. Chang, and E. Lee, "A Model-Based Design Methodology for Cyber-Physical Systems," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pp. 1666–1671, 2011.

[258] M. Broy, "Engineering Cyber-Physical Systems: Challenges and Foundations," in *Complex Systems Design & Management* (M. Aiguier, Y. Caseau, D. Krob, and A. Rauzy, eds.), pp. 1–13, Springer Berlin Heidelberg, 2013.

[259] Y. Ni and J. F. Broenink, "A co-modelling method for solving incompatibilities during co-design of mechatronic devices," *Advanced Engineering Informatics*, vol. 28, no. 3, pp. 232–240, 2014. Multi-view Modeling for Mechatronic Design.

[260] S. Robinson, *Simulation: the practice of model development and use*. Palgrave Macmillan, 2014.

[261] P. J. Ashenden, *The designer's guide to VHDL*, vol. 3. Morgan Kaufmann, 2010.

[262] J. Liu, "Continuous time and mixed-signal simulation in ptolemy ii," Tech. Rep. UCB/ERL M98/74, EECS Department, University of California, Berkeley, 1998.

[263] D. Thomas and P. Moorby, *The Verilog® Hardware Description Language*. Springer Science & Business Media, 2008.

[264] J. Banks, J. S. Carson II, L. Barry, *et al.*, *Discrete-event system simulation–fourth edition*. Pearson, 2005.

[265] M. Chiodo, P. Giusto, A. Jurecska, H. C. Hsieh, A. Sangiovanni-Vincentelli, and L. Lavagno, "Hardware-software codesign of embedded systems," *IEEE Micro*, vol. 14, pp. 26–36, August 1994.

[266] J. S. Fitzgerald, P. G. Larsen, K. Pierce, and M. Verhoef, *A Formal Approach to Collaborative Modelling and Co-simulation for Embedded Systems*. Newcastle University, Computing Science, 2011.

[267] M. P. Christiansen, *Co-modelling of Agricultural Robotic Systems*. PhD thesis, Aarhus University, Jun 2015.

[268] J. Fitzgerald and K. Pierce, "Co-modelling and co-simulation in embedded systems design," in *Collaborative Design for Embedded Systems*, pp. 15–25, Springer, 2014.

[269] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, "Co-simulation: A survey," *ACM Comput. Surv.*, vol. 51, pp. 1–33, May 2018.

[270] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Claub, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf, "The functional mockup interface for tool independent exchange of simulation models," in *Proceedings of the 8th International Modelica Conference*, pp. 105–114, Mar 2011.

[271] J. Fitzgerald, P. Larsen, K. Pierce, M. Verhoef, and S. Wolff, "Collaborative modelling and co-simulation in the development of dependable embedded systems," in *Integrated Formal Methods - IFM 2010*, Oct 2010.

[272] R. Payne, C. Gamble, K. Pierce, J. Fitzgerald, M. Mansfield, S. Foster, K. Ye, C. Thule, R. Nilsson, K. Lausdahl, F. Lapschies, and F. Foldager, "D3.6 — Examples Compendium 3," tech. rep., INTO-CPS, Dec 2017.

[273] K. Betts and M. Petty, "Automated search-based robustness testing for autonomous vehicle software," *Modelling and Simulation in Engineering*, vol. 2016, pp. 1–15, Jan 2016.

[274] S. F. Brown, "Naivety in systems engineering research: are we putting the methodological cart before the philosophical horse," in *Seventh Annual Conference on Systems Engineering Research*, 2009.

[275] T. L. Ferris, "On the methods of research for systems engineering," in *7th Annual Conference on Systems Engineering Research 2009 (CSER 2009)*, 2009.

[276] N. Cross, *The Development of Design Methodology in Architecture, Urban Planning and Industrial Design*, pp. 173–180. Dordrecht: Springer Netherlands, 1986.

[277] L. T. M. Blessing and A. Chakrabarti, *DRM, a Design Research Methodology*. Springer Publishing Company, Incorporated, 1st ed., 2009.

[278] L. Frankel and M. Racine, "The complex field of research: for design, through design, and about design," in *Design & Complexity - Proceedings of the Design Research Society Conference, 2010*, pp. 518–530, 2010.

[279] P. Downton, *Design Research.* Melburne: RMIT University Press, 2003.

[280] B. Archer, "The nature of research," *CoDesign*, pp. 6–13, 1995.

[281] R. Buchanan, *Strategies of Design Research: Productive Science and Rhetorical Inquiry*, pp. 55–66. Springer, Dec 2007.

[282] W. Jonas, *Design Research and its Meaning to the Methodological Development of the Discipline*, pp. 187–206. Springer, Dec 2007.

[283] H. W. J. Rittel and M. M. Webber, "Dilemmas in a general theory of planning," *Policy Sciences*, vol. 4, pp. 155–169, Jun 1973.

[284] B. Lawson, *How Designers Think.* Oxford, England: Architectural Press, 4 ed., Oct. 2005.

[285] Y. Reich, "The study of design research methodology," *Journal of Mechanical Design*, vol. 117, Mar 1995.

[286] N. Bayazit, "Investigating design: A review of forty years of design research," *Design Issues*, vol. 20, pp. 16–29, Dec 2004.

[287] N. F. M. Roozenburg and J. Eekels, *Product Design: Fundamentals and Methods (Product Development: Planning, Design, Engineering).* Wiley, Jun 1995.

[288] J. Mccarthy and P. J. Hayes, "Some philosophical problems from the standpoint of artificial intelligence," in *Machine Intelligence*, pp. 463–502, Edinburgh University Press, 1969.

[289] J. h. Alexander, M. j. Freiling, S. j. Shulman, J. l. Staley, S. Rehfuss, S. Messick, and S. l. Messick, "Knowledge level engineering: Ontological analysis," in *AAAI-86: Proceedings of the 5th National Conference on Artificial Intelligence*, AAAI Press, 1986.

[290] M. Seligman, "The fifth generation," *PC World*, May 1983.

[291] P. Szolovits, *On Knowledge Base Management Systems: Integrating Artificial Intelligence and D Atabase Technologies.* New York, NY, USA: Springer-Verlag New York, Inc., 1986.

[292] G. Avram, R. Public, H. Tudor, and L. , "Empirical study on knowledge based systems," *The Electronic Journal of Information Systems Evaluation*, vol. 8, Jan 2004.

[293] C. Tan, L. Wahidin, S. Khalil, N. Tamaldin, J. Hu, and G. Rauterberg, "The application of expert system: a review of research and applications," *ARPN Journal of Engineering and Applied Sciences*, vol. 11, no. 4, pp. 2448–2453, 2016.

[294] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities," *ScientificAmerican.com*, May 2001.

[295] D. Corsar and D. Sleeman, "Developing knowledge-based systems using the semantic web," in *Proceedings of the 2008 International Conference on Visions of Computer Science: BCS International*

Academic Conference, VoCS'08, (Swindon, UK), pp. 29–40, BCS Learning & Development Ltd., 2008.

[296] N. Guarino, "Formal ontology, conceptual analysis and knowledge representation," *International Journal of Human-Computer Studies*, vol. 43, no. 5, pp. 625–640, 1995.

[297] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing?," *International Journal of Human-Computer Studies*, vol. 43, no. 5, pp. 907–928, 1995.

[298] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[299] W. Borst and W. Borst, *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD thesis, University of Twente, Netherlands, Sep 1997.

[300] R. Studer, V. R. Benjamins, and D. Fensel, "Knowledge engineering: principles and methods," *Data & Knowledge Engineering*, vol. 25, pp. 161–197, Mar 1998.

[301] N. Guarino, D. Oberle, and S. Staab, "What is an ontology?," in *Handbook on Ontologies*, 2009.

[302] G. Heijst, *The Role of Ontologies in Knowledge Engineering*. PhD thesis, University of Armsterdam, Jan 1995.

[303] M. Uschold and M. Gruninger, "Ontologies: Principles, methods and applications," *KNOWLEDGE ENGINEERING REVIEW*, vol. 11, pp. 93–136, 1996.

[304] M. Gallaher, A. O'Connor, J. Dettbarn, and L. Gilday, "Cost analysis of inadequate interoperability in the us capital facilities industry," *NIST*, Jan 2004.

[305] L. van Ruijven, "Ontology for systems engineering," *Procedia Computer Science*, vol. 16, pp. 383–392, 2013. 2013 Conference on Systems Engineering Research.

[306] V. Maniraj and S. Ramakrishnan, "Ontology languages - a review," *International Journal of Computer Theory and Engineering*, vol. 2, pp. 887–891, Jan 2010.

[307] M. Imran and B. Young, "The application of common logic based formal ontologies to assembly knowledge sharing," *Journal of Intelligent Manufacturing*, vol. 26, pp. 139–158, Feb 2015.

[308] M. Kifer, "Rule interchange format: The framework," in *Web Reasoning and Rule Systems* (D. Calvanese and G. Lausen, eds.), (Berlin, Heidelberg), pp. 1–11, Springer Berlin Heidelberg, 2008.

[309] T. D. Wang, B. Parsia, and J. A. Hendler, "A survey of the web ontology landscape," in *International Semantic Web Conference*, 2006.

[310] J. Angele, M. Kifer, and G. Lausen, *Ontologies in F-Logic*, pp. 45–70. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[311] C. Menzel, R. Mayer, and M. Painter, "Idef5 ontology description capture method: Concepts and formal foundations," *Armstrong Laboratory*, p. 37, Nov 1992.

[312] A. B. Benevides and G. Guizzardi, "A model-based tool for conceptual modeling and domain ontology engineering in ontouml," in *ICEIS*, 2009.

[313] A. Gomez-Perez, O. Corcho-Garcia, and M. Fernandez-Lopez, *Ontological Engineering*. Berlin, Heidelberg: Springer-Verlag, 2003.

[314] Y. Sure, S. Staab, and R. Studer, *On-To-Knowledge Methodology (OTKM)*, pp. 117–132. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[315] N. F. Noy and D. L. Mcguinness, "Ontology development 101: A guide to creating your first ontology," tech. rep., Stanford Knowledge Systems Laboratory, 2001.

[316] H. S. Pinto, S. Staab, and C. Tempich, "Diligent: Towards a fine-grained methodology for distributed, loosely-controlled and evolving engineering of ontologies," in *Proceedings of the 16th European Conference on Artificial Intelligence*, ECAI'04, (Amsterdam, The Netherlands, The Netherlands), pp. 393–397, IOS Press, 2004.

[317] M. Cristani and R. Cuel, "A survey on ontology creation methodologies," *Int. J. Semantic Web Inf. Syst.*, vol. 1, pp. 49–69, Apr 2005.

[318] K. Todorova, "Towards a methodology for ontology development," in *Innovations in E-learning, Instruction Technology, Assessment, and Engineering Education* (M. Iskander, ed.), (Dordrecht), pp. 205–210, Springer Netherlands, 2007.

[319] R. Andryani and S. N. Edi, "Survey on development method of ontology," in *The 4th ICIBA 2015, International Conference on Information Technology and Business Applications*, 2015.

[320] M. Fernández-López, A. Gómez-Pérez, and N. Juristo, "Methontology: From ontological art towards ontological engineering," in *Proceedings of the Ontological Engineering AAAI-97 Spring Symposium Series*, American Asociation for Artificial Intelligence, Mar 1997. Ontology Engineering Group ? OEG.

[321] E. Simperl, M. Mochol, T. Bürger, and I. O. Popov, "Achieving maturity: The state of practice in ontology engineering in 2009," in *On the Move to Meaningful Internet Systems: OTM 2009*, (Berlin, Heidelberg), pp. 983–991, Springer Berlin Heidelberg, 2009.

[322] M. C. Suárez-Figueroa, A. Gómez-Pérez, and M. Fernández-López, *The NeOn Methodology for Ontology Engineering*, pp. 9–34. Springer Berlin Heidelberg, Jan 2012.

[323] C. Bezerra, F. Freitas, and F. Santana, "Evaluating ontologies with competency questions," in *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, vol. 3, pp. 284–285, Nov 2013.

[324] C. Bezerra and F. L. G. de Freitas, "Verifying description logic ontologies based on competency questions and unit testing," in *ONTOBRAS*, 2017.

[325] Y. Ren, A. Parvizi, C. Mellish, J. Z. Pan, K. van Deemter, and R. Stevens, "Towards competency question-driven ontology authoring," in *ESWC*, 2014.

[326] P. E. van der Vet and N. J. I. Mars, "Bottom-up construction of ontologies," *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, pp. 513–526, Jul 1998.

[327] C. Hildebrandt, S. Törsleff, B. Caesar, and A. Fay, "Ontology building for cyber-physical systems: A domain expert-centric approach," *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pp. 1079–1086, 2018.

[328] R. Laleau, F. Semmak, A. Matoussi, D. Petit, H. Ahmed, and B. Tatibouët, "A first attempt to combine sysml requirements diagrams and b," *ISSE*, vol. 6, pp. 47–54, Mar 2010.

[329] Y. Bijan, J. Yu, J. Stracener, and T. Woods, "Systems requirements engineering-state of the methodology," *Systems Engineering*, vol. 16, no. 3, pp. 267–276, 2013.

[330] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Science of Computer Programming*, vol. 20, no. 1, pp. 3–50, 1993.

[331] B. Penzenstadler and J. Eckhardt, "A requirements engineering content model for cyber-physical systems," in *2012 Second IEEE International Workshop on Requirements Engineering for Systems, Services, and Systems-of-Systems (RESS)*, pp. 20–29, Sep. 2012.

[332] H. Scherer, A. Albers, and N. Bursac, "Model based requirements engineering for the development of modular kits," *Procedia CIRP*, vol. 60, pp. 145–150, 2017. Complex Systems Engineering and Development Proceedings of the 27th CIRP Design Conference Cranfield University, UK 10th - 12th May 2017.

[333] G. A. Lewis, E. Morris, P. Place, S. Simanta, and D. B. Smith, "Requirements Engineering for Systems of Systems," in *Systems Conference, 2009 3rd Annual IEEE*, pp. 247–252, IEEE, March 2009.

[334] E. Hull, K. Jackson, and J. Dick, *Requirements Engineering*. London: Springer London, 2011.

[335] H. J., P. S., and B. M., *Model-Based Requirements Engineering*. IET, 2011.

[336] J. Holt, S. Perry, R. Payne, J. Bryans, S. Hallerstede, and F. O. Hansen, "A model-based approach for requirements engineering for systems of systems," *IEEE Systems Journal*, vol. 9, pp. 252–262, Mar 2015.

[337] S. Friedenthal, A. Moore, and R. Steiner, *Chapter 12. Modeling Functionality with Use Cases*, pp. 295–307. OŔeilly, 12 2015.

[338] L. Delligatti, *SysML Distilled: A Brief Guide to the Systems Modeling Language*. Addison-Wesley Professional, 1st ed., 2013.

[339] D. Kaslow, B. Ayres, P. Cahill, L. Hart, and R. Yntema, "A model-based systems engineering (mbse) approach for defining the behaviors of cubesats," in *2017 IEEE Aerospace Conference*, pp. 1–14, Mar 2017.

[340] Office of the Deputy Under Secretary of Defense for Acquisition and Technology, Systems and Software Engineering, "Systems Engineering Guide for Systems of Systems ," *USA DoD*, Aug 2008.

[341] E. Fosse and C. L. Delp, "Systems engineering interfaces: A model based approach," in *2013 IEEE Aerospace Conference*, pp. 1–8, Mar 2013.

[342] P. Colombo, V. del Bianco, L. Lavazza, and A. Coen-Porisini, "An experience in modeling real-time systems with sysml," in *International Workshop on Modeling and Analysis of Real-Time and Embedded Systems*, pp. 157–174, Jan 2006.

[343] Q. Ribeiro, F. Ribeiro, and M. Soares, "A technique to architect real-time embedded systems with sysml and uml through multiple views," in *Proceedings of the 19th International Conference on Enterprise Information Systems - Volume 2: ICEIS*, pp. 287–294, Jan 2017.

[344] V. Uc-Cetina, "A novel reinforcement learning architecture for continuous state and action spaces," *Advances in Artificial Intelligence*, vol. 2013, Apr 2013.

[345] D. Nunes, S. J. Sa, and F. Boavida, *A practical introduction to human-in-the-loop cyber-physical systems*. Wiley Blackwell, 2018.

[346] F. Dressler, "Cyber physical social systems: Towards deeply integrated hybridized systems," in *2018 International Conference on Computing, Networking and Communications (ICNC)*, pp. 420–424, Mar 2018.

[347] A. M. Madni, M. Sievers, and C. C. Madni, “Adaptive cyber-physical-human systems: Exploiting cognitive modeling and machine learning in the control loop,” *INSIGHT*, vol. 21, no. 3, pp. 87–93, 2018.

[348] S. K. Sowe, E. Simmon, K. Zettsu, F. de Vaulx, and I. Bojanova, “Cyber-physical-human systems: Putting people in the loop,” *IT Professional*, vol. 18, pp. 10–13, Jan 2016.

[349] S. Wang, D. Wang, L. Su, L. Kaplan, and T. F. Abdelzaher, “Towards cyber-physical systems in social spaces: The data reliability challenge,” *Proceedings - Real-Time Systems Symposium*, vol. 2015, pp. 74–85, Jan 2015.

[350] Object Management Group (OMG), “The MDA Foundation Model.” OMG Document Number ORMSC/2010-09-06, 20010.

[351] D. Dori, *Object-Process Methodology: A Holistic Systems Paradigm*. Springer, May 2013.

[352] G. Booch, J. E. Rumbaugh, and I. Jacobson, “The unified modeling language user guide,” *J. Database Manag.*, vol. 10, pp. 51–52, 1999.

[353] N. Amalio, R. Payne, A. Cavalcanti, E. Brosse, and J. Woodcock, “D2.1a — Foundations of the SysML profile for CPS modelling,” tech. rep., INTO-CPS, Dec 2015.

[354] L. Zdanis and R. Cloutier, “The use of behavioral diagrams in sysml,” in *2007 IEEE Long Island Systems, Applications and Technology Conference*, p. 1, Jun 2007.

[355] M. Deshpande, *Elements of Electrical Power Station Design*. PHI Learning, 2009.

[356] C. Gellings, *The Smart Grid: Enabling Energy Efficiency and Demand Response*. Fairmont Press, 2009.

[357] M. M. Ahmed and W. L. Soo, “Supervisory control and data acquisition system (scada) based customized remote terminal unit (rtu) for distribution automation system,” in *2008 IEEE 2nd International Power and Energy Conference*, pp. 1655–1660, Dec 2008.

[358] R. Li, Z. Wang, S. Le Blond, and F. Li, “Development of time-of-use price by clustering techniques,” in *2014 IEEE PES General Meeting | Conference Exposition*, pp. 1–5, 2014.

[359] A. Chikwanha, S. Motepe, and R. Stopforth, “Survey and requirements for search and rescue ground and air vehicles for mining applications,” in *2012 19th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, pp. 105–109, Nov 2012.

[360] M. Wzorek, C. Berger, P. Rudol, and P. Doherty, “Deployment of ad hoc network nodes using uavs for search and rescue missions,” in *2018 International Electrical Engineering Congress (iEECON)*, pp. 1–4, Mar 2018.

[361] G. Shih, P. Tsai, and C. Lin, “A speed up approach for search and rescue,” in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 4178–4183, Oct 2018.

[362] H. A. Lauterbach, C. B. Koch, R. Hess, D. Eck, K. Schilling, and A. Nüchter, “The eins3d project - instantaneous uav-based 3d mapping for search and rescue applications,” in *2019 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–6, Sep. 2019.

[363] A. Amanatiadis, E. G. Karakasis, L. Bampis, T. Giitsidis, P. Panagiotou, G. C. Sirakoulis, A. Gasteratos, P. Tsalides, A. Goulas, and K. Yakinthos, “The HCUAV project: Electronics and software development for medium altitude remote sensing,” in *2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (2014)*, pp. 1–5, Oct 2014.

[364] P. Rudol and P. Doherty, “Human body detection and geolocalization for uav search and rescue missions using color and thermal imagery,” in *2008 IEEE Aerospace Conference*, pp. 1–8, Mar 2008.

[365] P. Mahadik, D. Bhattacharyya, and H. J. Kim, “Techniques for automated test cases generation: A review,” *International Journal of Software Engineering and Its Applications*, vol. 10, pp. 13–20, Dec 2016.

[366] J. Edvardsson, “A survey on automatic test data generation,” in *Proceedings of the Second Conference on Computer Science and Engineering*, 1999.

[367] *Multi-Agent-Based Self-Organizing Manufacturing Network Towards Mass Personalization*, vol. Volume 2: Manufacturing Processes; Manufacturing Systems; Nano/Micro/Meso Manufacturing; Quality and Reliability of *International Manufacturing Science and Engineering Conference*, 06 2021. V002T07A016.

[368] M. Metwaly and M. Afifi, “Integrating cyber-physical systems as a bio-mimicking construction system: Developing and examining of a knowledge-based system,” *Alexandria Engineering Journal*, vol. 59, no. 6, pp. 4283–4300, 2020.

[369] C. Gershenson, “Guiding the self-organization of cyber-physical systems,” *Frontiers in Robotics and AI*, vol. 7, Apr 2020.

[370] A. Gurwitsch, *Maurice Merleau-Ponty, Phénoménologie De La Perception (Paris: Librairie Gallimard, 1945), Xvi and 531 pp.*, pp. 487–490. Dordrecht: Springer Netherlands, 2010.

[371] S. Li, X. Xing, and S. Du, “A multi-dimensional assessment system for technology readiness levels,” in *2017 4th International Conference on Systems and Informatics (ICSAI)*, pp. 798–802, 2017.

Appendix: ACPS Ontology in OWL A

This appendix details the formal ontology for the ACPS Ontology from Chapter 4. Formal ontology here refer to ontology defined with semantically formal languages such as OWL. The main use of the ACPS Ontology is to support MBSE AF defined in Chapter 5. Ontology for MBSE purposes are typically defined in SysML or equivalent MBSE modelling language as it allow the AF user to use the ontology without having to learn a different ontology language. However, a formal ontology provide a more consistent definition of the ontology as the semantics of ontology are based on formal language with strict semantics. Hildebrandt et al. [327] propose a framework for building CPS ontology that separates ontology into Lightweight Ontology and Heavyweight ontology

Although formal ontology is not required in our research to create ACPSAF, we translate our SysML ontology into formal ontology described in OWL. This is to demonstrate how ACPS Ontology could be translated and described using formal ontology language. The ontology presented here has not been verified and validated against its intended use. ACPSAF can serve as a validation to the ontology, however it uses the SysML ontology and not the formal ontology.

A.1 Formal ACPS Ontology in OWL format

```xml
<?xml version="1.0"?>
<Ontology xmlns="http://www.w3.org/2002/07/owl#"
  xml:base="http://webprotege.stanford.edu/project/CzrSveKqJy1aCJmDDJ7dk7"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  ontologyIRI="http://webprotege.stanford.edu/project/CzrSveKqJy1aCJmDDJ7dk7">
 <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
 <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
 <Prefix name="xml" IRI="http://www.w3.org/XML/1998/namespace"/>
 <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
 <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>
 <Declaration>
       <Class IRI="http://webprotege.stanford.edu/R5MXz7cCMCe9pYT0eXhOAl"/>
 </Declaration>
 <Declaration>
       <Class IRI="http://webprotege.stanford.edu/R6mbyMY1RW6prHWXLVo4FQ"/>
 </Declaration>
 <Declaration>
       <Class IRI="http://webprotege.stanford.edu/R73opw8bm1QG44BUq5Qjy21"/>
 </Declaration>
 <Declaration>
       <Class IRI="http://webprotege.stanford.edu/R7OtwE7OEPsByv0QOBZuwbY"/>
 </Declaration>
 <Declaration>
       <Class IRI="http://webprotege.stanford.edu/R7PQC1uiT8O5ng3dBODMPz1"/>
 </Declaration>
 <Declaration>
       <Class IRI="http://webprotege.stanford.edu/R7h1iz4zyiPbY1361dPwfAV"/>
 </Declaration>
 <Declaration>
       <Class IRI="http://webprotege.stanford.edu/R7jI9SUYN6E6udnXGQaEnRV"/>
 </Declaration>
 <Declaration>
       <Class IRI="http://webprotege.stanford.edu/R7qvuBzekNoqGMoeBEKw0dB"/>
 </Declaration>
 <Declaration>
       <Class IRI="http://webprotege.stanford.edu/R8CgNex60UsSEwgvzOw9EBN"/>
 </Declaration>
 <Declaration>
       <Class IRI="http://webprotege.stanford.edu/R8WRnb96ahwyMHwiVJ3oro7"/>
 </Declaration>
 <Declaration>
       <Class IRI="http://webprotege.stanford.edu/R8nglmdJiBPKLMGyP24pK53"/>
 </Declaration>
 <Declaration>
       <Class IRI="http://webprotege.stanford.edu/R8vXBYe2AKxDGLHazgzCMgm"/>
 </Declaration>
 <Declaration>
       <Class IRI="http://webprotege.stanford.edu/R9A8hTFHheRtMKcK3KBwGdv"/>
 </Declaration>
 <Declaration>
       <Class IRI="http://webprotege.stanford.edu/R9ABjRzbUFcRI432tc6FUq6"/>
 </Declaration>
 <Declaration>
       <Class IRI="http://webprotege.stanford.edu/R9C6bCgLKJfSIODJYcl0wpj"/>
 </Declaration>
 <Declaration>
       <Class IRI="http://webprotege.stanford.edu/R9JtlqTJdiLlqC0uvxk15nk"/>
 </Declaration>
 <Declaration>
       <Class IRI="http://webprotege.stanford.edu/R9ZmZemiGXvbsTfk8D7Dlh4"/>
 </Declaration>
 <Declaration>
```

```
      <Class IRI="http://webprotege.stanford.edu/R9c7YIIRdeoytbz4eUQNNxN"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/R9m8ejCV1EPuUhanyDMQwyd"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/R9vbhlK4QUc5JAUW5km0zqE"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RB6NHdiQ8V8XL6NYfRB28Tv"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RBayqrj2J6PypKPtsdgzLA"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RBqFhUrgtyyGiwsbSI4JHOM"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RBuA5VUhS4iH8bJXTxhEQqB"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RByTbdMbn1qHS2UaF7BAlKM"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RCSxOlf2VxhZqL2qC71FXO4"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RCZhTqgDBRqR2Ip75fKiYmA"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RCamf5Z8o5GqHRofUEE7Lcj"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RCgH3Ki1h9jAcTwGcnitsle"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RCmVfB8te7AoMPocy57SwM7"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RCog4HzIQdTadux5GzkEEPX"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RCtFaTmrJpiOJJamK0OuOce"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RDCjQhiPSyshbMxkmXco24y"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RDV8BJbFnUVqOWyIN2krfdT"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RDZ29iMc2khwG2HAbJgYAN1"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RDZWp18uIsiBwKO47bCmseG"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RDyfbn5NdEJaQAb8seqTD06"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RGqS9Z6sxrnxXREoEh1zru"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RIs55NZco7XVhwXXF4kf8Q"/>
</Declaration>
```

```
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RQAgGH0oOfoXdqPIBzexFz"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RRfgUtsB5N7UmG8UNoNRdo"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RXxX1HKlcPRwCrYKlzhR2V"/>
</Declaration>
<Declaration>
      <Class IRI="http://webprotege.stanford.edu/RsT4CZiOHQvMnRIvLkpFdO"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/R86zLr7OTVM80y1uPYMUGXf"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/R87p1A9dskMYIXKmijbU6nh"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/R8JAcgbuuwTcIzumHbxh7Va"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/R8RwWAbJizd2gWvGFzMVPmI"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/R8SM54pJQCCgcChUGilalHS"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/R8ZVypU7Y7PAeEG7juQXucR"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/R8gvMyDnY6MJxszYPbzuu2g"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/R8ycg39lDYYtnngvsXUKKC4"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/R9H54lEvF3U0oywzFNhfK82"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/R9U5uOQFytzOOPL26HYHUNh"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/RBRtQBixVkshYXtb8QCBsgb"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/RBdFqlu7CJ8baln3hmMuzov"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/RBhDgzVE4BjJi7w8Pocqa2D"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/RBv43jxjcs9p2PN9ggRQFjl"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/RCY7N7qsIN6pwWFfAGFHtfR"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/RCazrfh7GRcI0RZadodoMDF"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/RCcmS3rcPyB5DFhEmh9LwIX"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/RDNV14aBkXrGd2r7C2zUDlT"/>
```

```
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/RDXnKKwdknr9vTVyN9eE9u"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/RDYldenUarnwduAyYOKgGl5"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/RDnY83kTwsxIAPRvOXA8sK"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/RDoOPw3TWQpBfoXvtu9c0YV"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/RY6p8bFqvW26CYt3ZxYhSJ"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/RicbE1POHKdjERriypswMl"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/RlaZD3gxwZjX5KnKleWGl3"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/RomXbEJ1U5ocymcMWo7N3P"/>
</Declaration>
<Declaration>
      <ObjectProperty IRI="http://webprotege.stanford.edu/RrQ7oxMm9hlh82zy1FHjnq"/>
</Declaration>
<Declaration>
      <AnnotationProperty IRI="http://webprotege.stanford.edu/R8XZxGw8zr5hZpZD1cruhJM"/>
</Declaration>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R5MXz7cCMCe9pYT0eXhOAl"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RDnY83kTwsxIAPRvOXA8sK"/>
            <Class IRI="http://webprotege.stanford.edu/R6mbyMY1RW6prHWXLVo4FQ"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R6mbyMY1RW6prHWXLVo4FQ"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RDYldenUarnwduAyYOKgGl5"/>
            <Class IRI="http://webprotege.stanford.edu/R8CgNex60UsSEwgvzOw9EBN"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R7OtwE7OEPsByv0QOBZuwbY"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/R8RwWAbJizd2gWvGFzMVPmI"/>
            <Class IRI="http://webprotege.stanford.edu/R9ZmZemiGXvbsTfk8D7Dlh4"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R7OtwE7OEPsByv0QOBZuwbY"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RCY7N7qsIN6pwWFfAGFHtfR"/>
            <Class IRI="http://webprotege.stanford.edu/R73opw8bm1QG44BUq5Qjy21"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R7PQC1uiT8O5ng3dBODMPz1"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RDNV14aBkXrGd2r7C2zUDlT"/>
            <Class IRI="http://webprotege.stanford.edu/RIs55NZco7XVhwXXF4kf8Q"/>
      </ObjectSomeValuesFrom>
```

```
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R7PQC1uiT8O5ng3dBODMPz1"/>
      <ObjectSomeValuesFrom>
             <ObjectProperty IRI="http://webprotege.stanford.edu/RDXnKKwdknr9vTVyN9eE9u"/>
             <Class IRI="http://webprotege.stanford.edu/R9C6bCgLKJfSIODJYcl0wpj"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R7h1iz4zyiPbY1361dPwfAV"/>
      <Class IRI="http://webprotege.stanford.edu/R8vXBYe2AKxDGLHazgzCMgm"/>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R7qvuBzekNoqGMoeBEKw0dB"/>
      <Class IRI="http://webprotege.stanford.edu/R8vXBYe2AKxDGLHazgzCMgm"/>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R8CgNex60UsSEwgvzOw9EBN"/>
      <ObjectSomeValuesFrom>
             <ObjectProperty IRI="http://webprotege.stanford.edu/R8JAcgbuuwTcIzumHbxh7Va"/>
             <Class IRI="http://webprotege.stanford.edu/R8vXBYe2AKxDGLHazgzCMgm"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R8CgNex60UsSEwgvzOw9EBN"/>
      <ObjectSomeValuesFrom>
             <ObjectProperty IRI="http://webprotege.stanford.edu/RBv43jxjcs9p2PN9ggRQFjl"/>
             <Class IRI="http://webprotege.stanford.edu/RDyfbn5NdEJaQAb8seqTD06"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R8WRnb96ahwyMHwiVJ3oro7"/>
      <Class IRI="http://webprotege.stanford.edu/R8nglmdJiBPKLMGyP24pK53"/>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R8nglmdJiBPKLMGyP24pK53"/>
      <ObjectSomeValuesFrom>
             <ObjectProperty IRI="http://webprotege.stanford.edu/RlaZD3gxwZjX5KnKleWGl3"/>
             <Class IRI="http://webprotege.stanford.edu/RCZhTqgDBRqR2Ip75fKiYmA"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R8vXBYe2AKxDGLHazgzCMgm"/>
      <ObjectSomeValuesFrom>
             <ObjectProperty IRI="http://webprotege.stanford.edu/RCazrfh7GRcI0RZadodoMDF"/>
             <Class IRI="http://webprotege.stanford.edu/R7OtwE7OEPsByv0QOBZuwbY"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R9A8hTFHheRtMKcK3KBwGdv"/>
      <ObjectSomeValuesFrom>
             <ObjectProperty IRI="http://webprotege.stanford.edu/R8JAcgbuuwTcIzumHbxh7Va"/>
             <Class IRI="http://webprotege.stanford.edu/RDCjQhiPSyshbMxkmXco24y"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R9A8hTFHheRtMKcK3KBwGdv"/>
      <ObjectSomeValuesFrom>
             <ObjectProperty IRI="http://webprotege.stanford.edu/RDXnKKwdknr9vTVyN9eE9u"/>
             <Class IRI="http://webprotege.stanford.edu/RDZWp18uIsiBwKO47bCmseG"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R9A8hTFHheRtMKcK3KBwGdv"/>
      <ObjectSomeValuesFrom>
```

```
            <ObjectProperty IRI="http://webprotege.stanford.edu/RDXnKKwdknr9vTVyN9eE9u"/>
            <Class IRI="http://webprotege.stanford.edu/RIs55NZco7XVhwXXF4kf8Q"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R9ABjRzbUFcRI432tc6FUq6"/>
      <Class IRI="http://webprotege.stanford.edu/R8vXBYe2AKxDGLHazgzCMgm"/>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R9C6bCgLKJfSIODJYcl0wpj"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RicbE1POHKdjERriypswMl"/>
            <Class IRI="http://webprotege.stanford.edu/RQAgGH0oOfoXdqPIBzexFz"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R9c7YIIRdeoytbz4eUQNNxN"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/R8ycg39lDYYtnngvsXUKKC4"/>
            <Class IRI="http://webprotege.stanford.edu/RCtFaTmrJpiOJJamK0OuOce"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R9c7YIIRdeoytbz4eUQNNxN"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RDNV14aBkXrGd2r7C2zUDlT"/>
            <Class IRI="http://webprotege.stanford.edu/RBqFhUrgtyyGiwsbSI4JHOM"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R9c7YIIRdeoytbz4eUQNNxN"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RDnY83kTwsxIAPRvOXA8sK"/>
            <Class IRI="http://webprotege.stanford.edu/R8nglmdJiBPKLMGyP24pK53"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R9m8ejCV1EPuUhanyDMQwyd"/>
      <Class IRI="http://webprotege.stanford.edu/R8nglmdJiBPKLMGyP24pK53"/>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/R9vbhlK4QUc5JAUW5km0zqE"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RCcmS3rcPyB5DFhEmh9LwIX"/>
            <Class IRI="http://webprotege.stanford.edu/RDV8BJbFnUVqOWyIN2krfdT"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RB6NHdiQ8V8XL6NYfRB28Tv"/>
      <Class IRI="http://webprotege.stanford.edu/RRfgUtsB5N7UmG8UNoNRdo"/>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RB6NHdiQ8V8XL6NYfRB28Tv"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/R8ZVypU7Y7PAeEG7juQXucR"/>
            <Class IRI="http://webprotege.stanford.edu/R9vbhlK4QUc5JAUW5km0zqE"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RBayqrj2J6PypKPtsdgzLA"/>
      <Class IRI="http://webprotege.stanford.edu/R7h1iz4zyiPbY1361dPwfAV"/>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RBayqrj2J6PypKPtsdgzLA"/>
      <ObjectSomeValuesFrom>
```

```
            <ObjectProperty IRI="http://webprotege.stanford.edu/R86zLr7OTVM80y1uPYMUGXf"/>
            <Class IRI="http://webprotege.stanford.edu/RCamf5Z8o5GqHRofUEE7Lcj"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RBuA5VUhS4iH8bJXTxhEQqB"/>
      <Class IRI="http://webprotege.stanford.edu/RXxX1HKlcPRwCrYKlzhR2V"/>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RBuA5VUhS4iH8bJXTxhEQqB"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RDXnKKwdknr9vTVyN9eE9u"/>
            <Class IRI="http://webprotege.stanford.edu/RRfgUtsB5N7UmG8UNoNRdo"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RByTbdMbn1qHS2UaF7BAlKM"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RDXnKKwdknr9vTVyN9eE9u"/>
            <Class IRI="http://webprotege.stanford.edu/R8nglmdJiBPKLMGyP24pK53"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RByTbdMbn1qHS2UaF7BAlKM"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RDnY83kTwsxIAPRvOXA8sK"/>
            <Class IRI="http://webprotege.stanford.edu/RXxX1HKlcPRwCrYKlzhR2V"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RCZhTqgDBRqR2Ip75fKiYmA"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/R8gvMyDnY6MJxszYPbzuu2g"/>
            <Class IRI="http://webprotege.stanford.edu/R9c7YIIRdeoytbz4eUQNNxN"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RCmVfB8te7AoMPocy57SwM7"/>
      <Class IRI="http://webprotege.stanford.edu/R8vXBYe2AKxDGLHazgzCMgm"/>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RCog4HzIQdTadux5GzkEEPX"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RDoOPw3TWQpBfoXvtu9c0YV"/>
            <Class IRI="http://webprotege.stanford.edu/RCgH3Ki1h9jAcTwGcnitsle"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RCtFaTmrJpiOJJamK0OuOce"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RBhDgzVE4BjJi7w8Pocqa2D"/>
            <Class IRI="http://webprotege.stanford.edu/R5MXz7cCMCe9pYT0eXhOAl"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RDCjQhiPSyshbMxkmXco24y"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/R8JAcgbuuwTcIzumHbxh7Va"/>
            <Class IRI="http://webprotege.stanford.edu/RXxX1HKlcPRwCrYKlzhR2V"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RDCjQhiPSyshbMxkmXco24y"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RDXnKKwdknr9vTVyN9eE9u"/>
            <Class IRI="http://webprotege.stanford.edu/R7PQC1uiT8O5ng3dBODMPz1"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RDV8BJbFnUVqOWyIN2krfdT"/>
      <Class IRI="http://webprotege.stanford.edu/RRfgUtsB5N7UmG8UNoNRdo"/>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RDZ29iMc2khwG2HAbJgYAN1"/>
      <Class IRI="http://webprotege.stanford.edu/RBuA5VUhS4iH8bJXTxhEQqB"/>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RDZ29iMc2khwG2HAbJgYAN1"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RDXnKKwdknr9vTVyN9eE9u"/>
            <Class IRI="http://webprotege.stanford.edu/RRfgUtsB5N7UmG8UNoNRdo"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RDZ29iMc2khwG2HAbJgYAN1"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RY6p8bFqvW26CYt3ZxYhSJ"/>
            <Class IRI="http://webprotege.stanford.edu/RCamf5Z8o5GqHRofUEE7Lcj"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RDZWp18uIsiBwKO47bCmseG"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RDXnKKwdknr9vTVyN9eE9u"/>
            <Class IRI="http://webprotege.stanford.edu/R9JtlqTJdiLlqC0uvxk15nk"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RDyfbn5NdEJaQAb8seqTD06"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RrQ7oxMm9hlh82zy1FHjnq"/>
            <Class IRI="http://webprotege.stanford.edu/RXxX1HKlcPRwCrYKlzhR2V"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RGqS9Z6sxrnxXREoEh1zru"/>
      <Class IRI="http://webprotege.stanford.edu/RRfgUtsB5N7UmG8UNoNRdo"/>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RGqS9Z6sxrnxXREoEh1zru"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/R8SM54pJQCCgcChUGilalHS"/>
            <Class IRI="http://webprotege.stanford.edu/RB6NHdiQ8V8XL6NYfRB28Tv"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RGqS9Z6sxrnxXREoEh1zru"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/R8SM54pJQCCgcChUGilalHS"/>
            <Class IRI="http://webprotege.stanford.edu/RDV8BJbFnUVqOWyIN2krfdT"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/RGqS9Z6sxrnxXREoEh1zru"/>
      <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/R9U5uOQFytzOOPL26HYHUNh"/>
            <Class IRI="http://webprotege.stanford.edu/RCamf5Z8o5GqHRofUEE7Lcj"/>
      </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
```

```
        <Class IRI="http://webprotege.stanford.edu/RIs55NZco7XVhwXXF4kf8Q"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RDXnKKwdknr9vTVyN9eE9u"/>
            <Class IRI="http://webprotege.stanford.edu/R9JtlqTJdiLlqC0uvxk15nk"/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="http://webprotege.stanford.edu/RRfgUtsB5N7UmG8UNoNRdo"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/R87p1A9dskMYIXKmijbU6nh"/>
            <Class IRI="http://webprotege.stanford.edu/RCog4HzIQdTadux5GzkEEPX"/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="http://webprotege.stanford.edu/RRfgUtsB5N7UmG8UNoNRdo"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/R8SM54pJQCCgcChUGilalHS"/>
            <Class IRI="http://webprotege.stanford.edu/RRfgUtsB5N7UmG8UNoNRdo"/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="http://webprotege.stanford.edu/RXxX1HKlcPRwCrYKlzhR2V"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RBRtQBixVkshYXtb8QCBsgb"/>
            <Class IRI="http://webprotege.stanford.edu/RCSxOlf2VxhZqL2qC71FXO4"/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="http://webprotege.stanford.edu/RXxX1HKlcPRwCrYKlzhR2V"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="http://webprotege.stanford.edu/RDXnKKwdknr9vTVyN9eE9u"/>
            <Class IRI="http://webprotege.stanford.edu/RRfgUtsB5N7UmG8UNoNRdo"/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="http://webprotege.stanford.edu/RsT4CZiOHQvMnRIvLkpFdO"/>
        <Class IRI="http://webprotege.stanford.edu/RRfgUtsB5N7UmG8UNoNRdo"/>
    </SubClassOf>
    <SubObjectPropertyOf>
        <ObjectProperty IRI="http://webprotege.stanford.edu/RDXnKKwdknr9vTVyN9eE9u"/>
        <ObjectProperty abbreviatedIRI="owl:topObjectProperty"/>
    </SubObjectPropertyOf>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
        <IRI>http://webprotege.stanford.edu/R5MXz7cCMCe9pYT0eXhOAl</IRI>
        <Literal xml:lang="en">Co-simulation</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
        <IRI>http://webprotege.stanford.edu/R6mbyMY1RW6prHWXLVo4FQ</IRI>
        <Literal xml:lang="en">Scenario</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
        <IRI>http://webprotege.stanford.edu/R73opw8bm1QG44BUq5Qjy21</IRI>
        <Literal xml:lang="en">Rule</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
        <IRI>http://webprotege.stanford.edu/R7OtwE7OEPsByv0QOBZuwbY</IRI>
        <Literal xml:lang="en">Need description</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
        <IRI>http://webprotege.stanford.edu/R7PQC1uiT8O5ng3dBODMPz1</IRI>
```

```
      <Literal xml:lang="en">View</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R7h1iz4zyiPbY1361dPwfAV</IRI>
      <Literal xml:lang="en">Requirement</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R7jI9SUYN6E6udnXGQaEnRV</IRI>
      <Literal xml:lang="en">System property</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R7qvuBzekNoqGMoeBEKw0dB</IRI>
      <Literal xml:lang="en">Capability</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R86zLr7OTVM80y1uPYMUGXf</IRI>
      <Literal xml:lang="en">describe need for</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R87p1A9dskMYIXKmijbU6nh</IRI>
      <Literal xml:lang="en">interacts through</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R8CgNex60UsSEwgvzOw9EBN</IRI>
      <Literal xml:lang="en">Use case</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R8JAcgbuuwTcIzumHbxh7Va</IRI>
      <Literal xml:lang="en">describes</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R8RwWAbJizd2gWvGFzMVPmI</IRI>
      <Literal xml:lang="en">is elicited from</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R8SM54pJQCCgcChUGilalHS</IRI>
      <Literal xml:lang="en">interacts with</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R8WRnb96ahwyMHwiVJ3oro7</IRI>
      <Literal xml:lang="en">CT model</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R8XZxGw8zr5hZpZD1cruhJM</IRI>
      <Literal xml:lang="en">test</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R8ZVypU7Y7PAeEG7juQXucR</IRI>
      <Literal xml:lang="en">influences</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R8gvMyDnY6MJxszYPbzuu2g</IRI>
      <Literal xml:lang="en">generates</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R8nglmdJiBPKLMGyP24pK53</IRI>
      <Literal xml:lang="en">Model</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R8vXBYe2AKxDGLHazgzCMgm</IRI>
      <Literal xml:lang="en">Need</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R8ycg39lDYYtnngvsXUKKC4</IRI>
      <Literal xml:lang="en">used by</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R9A8hTFHheRtMKcK3KBwGdv</IRI>
      <Literal xml:lang="en">Architectural Framework</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R9ABjRzbUFcRI432tc6FUq6</IRI>
      <Literal xml:lang="en">Concern</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R9C6bCgLKJfSIODJYcl0wpj</IRI>
      <Literal xml:lang="en">View element</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R9H54lEvF3U0oywzFNhfK82</IRI>
      <Literal xml:lang="en">has interest in</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R9JtlqTJdiLlqC0uvxk15nk</IRI>
      <Literal xml:lang="en">Ontology Element</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R9U5uOQFytzOOPL26HYHUNh</IRI>
      <Literal xml:lang="en">enable</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R9ZmZemiGXvbsTfk8D7Dlh4</IRI>
      <Literal xml:lang="en">Source element</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R9c7YIIRdeoytbz4eUQNNxN</IRI>
      <Literal xml:lang="en">FMU</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R9m8ejCV1EPuUhanyDMQwyd</IRI>
      <Literal xml:lang="en">DE model</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/R9vbhlK4QUc5JAUW5km0zqE</IRI>
      <Literal xml:lang="en">Environment</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RB6NHdiQ8V8XL6NYfRB28Tv</IRI>
      <Literal xml:lang="en">Actuator</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RBRtQBixVkshYXtb8QCBsgb</IRI>
      <Literal xml:lang="en">interests</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RBayqrj2J6PypKPtsdgzLA</IRI>
      <Literal xml:lang="en">Autonomy Requirement</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RBdFqlu7CJ8baln3hmMuzov</IRI>
      <Literal xml:lang="en">represented with</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RBhDgzVE4BjJi7w8Pocqa2D</IRI>
      <Literal xml:lang="en">performs</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RBqFhUrgtyyGiwsbSI4JHOM</IRI>
      <Literal xml:lang="en">FMI standard</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RBuA5VUhS4iH8bJXTxhEQqB</IRI>
      <Literal xml:lang="en">CPS</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RBv43jxjcs9p2PN9ggRQFjl</IRI>
      <Literal xml:lang="en">uses</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RByTbdMbn1qHS2UaF7BAlKM</IRI>
      <Literal xml:lang="en">Co-model</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RCSxOlf2VxhZqL2qC71FXO4</IRI>
      <Literal xml:lang="en">Stakeholder role</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
<AnnotationProperty abbreviatedIRI="rdfs:label"/>
<IRI>http://webprotege.stanford.edu/RCY7N7qsIN6pwWFfAGFHtfR</IRI>
<Literal xml:lang="en">constrained by</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RCZhTqgDBRqR2Ip75fKiYmA</IRI>
      <Literal xml:lang="en">Modelling tool</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RCamf5Z8o5GqHRofUEE7Lcj</IRI>
      <Literal xml:lang="en">Autonomy</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RCazrfh7GRcI0RZadodoMDF</IRI>
      <Literal xml:lang="en">described with</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RCcmS3rcPyB5DFhEmh9LwIX</IRI>
      <Literal xml:lang="en">monitored by</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RCgH3Ki1h9jAcTwGcnitsle</IRI>
      <Literal xml:lang="en">System function</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RCmVfB8te7AoMPocy57SwM7</IRI>
      <Literal xml:lang="en">Goal</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RCog4HzIQdTadux5GzkEEPX</IRI>
      <Literal xml:lang="en">Interface</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RCtFaTmrJpiOJJamK0OuOce</IRI>
      <Literal xml:lang="en">Co-simulation tool</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RDCjQhiPSyshbMxkmXco24y</IRI>
      <Literal xml:lang="en">Architecture</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RDNV14aBkXrGd2r7C2zUDlT</IRI>
      <Literal xml:lang="en">conforms to</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RDV8BJbFnUVqOWyIN2krfdT</IRI>
      <Literal xml:lang="en">Sensor</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RDXnKKwdknr9vTVyN9eE9u</IRI>
      <Literal xml:lang="en">has</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RDYldenUarnwduAyYOKgGl5</IRI>
      <Literal xml:lang="en">validates</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty IRI="http://webprotege.stanford.edu/R8XZxGw8zr5hZpZD1cruhJM"/>
      <IRI>http://webprotege.stanford.edu/RDZ29iMc2khwG2HAbJgYAN1</IRI>
      <Literal>asdsa</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RDZ29iMc2khwG2HAbJgYAN1</IRI>
      <Literal xml:lang="en">ACPS</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RDZWp18uIsiBwKO47bCmseG</IRI>
      <Literal xml:lang="en">Ontology</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RDnY83kTwsxIAPRvOXA8sK</IRI>
      <Literal xml:lang="en">represents</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RDoOPw3TWQpBfoXvtu9c0YV</IRI>
      <Literal xml:lang="en">provides</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RDyfbn5NdEJaQAb8seqTD06</IRI>
      <Literal xml:lang="en">Context</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RGqS9Z6sxrnxXREoEh1zru</IRI>
      <Literal xml:lang="en">Controller</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RIs55NZco7XVhwXXF4kf8Q</IRI>
      <Literal xml:lang="en">Viewpoint</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RQAgGH0oOfoXdqPIBzexFz</IRI>
      <Literal xml:lang="en">Viewpoint Element</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RRfgUtsB5N7UmG8UNoNRdo</IRI>
      <Literal xml:lang="en">System Element</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RXxX1HKlcPRwCrYKlzhR2V</IRI>
      <Literal xml:lang="en">System</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RY6p8bFqvW26CYt3ZxYhSJ</IRI>
      <Literal xml:lang="en">capable of</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RicbE1POHKdjERriypswMl</IRI>
      <Literal xml:lang="en">visualises</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RlaZD3gxwZjX5KnKleWGl3</IRI>
      <Literal xml:lang="en">created with</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
      <AnnotationProperty abbreviatedIRI="rdfs:label"/>
      <IRI>http://webprotege.stanford.edu/RomXbEJ1U5ocymcMWo7N3P</IRI>
```

```
        <Literal xml:lang="en">constrains</Literal>
 </AnnotationAssertion>
 <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
        <IRI>http://webprotege.stanford.edu/RrQ7oxMm9hlh82zy1FHjnq</IRI>
        <Literal xml:lang="en">represents the need for</Literal>
 </AnnotationAssertion>
 <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
        <IRI>http://webprotege.stanford.edu/RsT4CZiOHQvMnRIvLkpFdO</IRI>
        <Literal xml:lang="en">Part</Literal>
 </AnnotationAssertion>
</Ontology>



 <!-- Generated by the OWL API (version 4.5.13) https://github.com/owlcs/owlapi -->
```

B Appendix: Smart grid case study

This appendix details the VDM-RT and 20-sim models used in Chapter 6.2 in the self-healing smart grid case study. Section B.1 presents the VDM-RT code exported with the PdfLatex command built into the Overture tools. Section B.1 presents the 20-sim Models and table containing the configuration values of each component in the model.

B.1 Self-Healing Grid

Self-Healing Grid

November 4, 2019

Contents

1 Actuator

```
class Actuator

instance variables

-- this value will be set through the co-simulation
protected port: RealPort;

-- local copy of the shared variable
protected local_val: real := 0;

operations

-- constructor for PWM

public Actuator : RealPort ==> Actuator
Actuator(p) == (
 port := p;
);
```

```
-- set actuator value

public SetValue: real ==> ()
SetValue(v) ==  local_val := v;


public Read: () ==> real
Read() == return local_val;

-- write local variable to shared variable

public Sync: () ==> ()
Sync() == port.setValue(local_val);

end Actuator
```

2 Configuration

```
class Configuration

instance variables

protected configuration: seq of bool;

operations

-- constructor for PWM

public Configuration : bool * bool * bool ==> Configuration
Configuration(s1,s2,s3) == (
 configuration := [s1,s2,s3];
);


public setConfValue : bool * bool * bool ==> ()
setConfValue(s1,s2,s3) == (
 configuration := [s1,s2,s3];
);


public getConfValue : () ==> seq of bool
getConfValue() == (
 return configuration;
);

end Configuration
```

3 Controller

```
class Controller

types

instance variables
```

```
 switchTime : real := 5E9;

 sensor1Current: Sensor;
 sensor2Current: Sensor;
 sensor3Current: Sensor;

 faultControl: Actuator;
 switch1: Switch;
 switch2: Switch;
 switch3: Switch;

 priorityMap : PriorityMap;

 dependabilityCage : DependabilityCage;

protected counter: real := 0;

operations

 -- constructor for Controller

 public Controller :  Sensor * Sensor * Sensor * Actuator * Switch * Switch * Switch ==>
     Controller
 Controller (current1, current2, current3, fault , SS1 , SS2 , SS3) ==
  (
   sensor1Current := current1;
   sensor2Current := current2;
   sensor3Current := current3;

   faultControl := fault;
   switch1 := SS1;
   switch2 := SS2;
   switch3 := SS3;
   faultControl.SetValue(0.00000000000001);
   priorityMap := new PriorityMap();
   dependabilityCage := new DependabilityCage();
  );

 -- MAIN CONTROL LOOP

 public Step : () ==> ()
 Step() ==
 (
  --induce fault at a predetermined time
   if (time >5E9 and time <15E9) then
   (
    faultControl.SetValue(1000000000000000);
   )else(
    faultControl.SetValue(0.00000000000001);
   );

   --This is to simulate manual command by control center operator
   --When physical afult is fixed, the control center should be informed
   --and operator would initiate recovery mechanism
   if (time > 15E9) then
   (
    recovery();
   );

   --Check if any switch entered emergency state
  if(isFaulty()) then (
   recalculateConfigurations();
  );
```

```
   --smart grid always choose the best available configuration
  optimise();
 );


 public isFaulty : () ==> bool
 isFaulty() ==
 (
  if switch1.isInEmergency() then return true;
  if switch2.isInEmergency() then return true;
  if switch3.isInEmergency() then return true;
  return false;
 );



 public recalculateConfigurations : () ==> ()
 recalculateConfigurations() ==
 (
  priorityMap.generateConfigurations([switch1.isInEmergency(),switch2.isInEmergency(),switch3.
      isInEmergency()]);

 );


 public optimise : () ==> ()
 optimise() ==
 (
  --only change configuration if allowed by the dependability cage
  if(dependabilityCage.dependabilityControl(priorityMap.getBestConfig())) then (
   dcl bestConf : seq of bool :=  priorityMap.getBestConfig().getConfValue();
   if(bestConf(1)) then (switch1.closeSwitch()) else (switch1.openSwitch());
   if(bestConf(2)) then (switch2.closeSwitch()) else (switch2.openSwitch());
   if(bestConf(3)) then (switch3.closeSwitch()) else (switch3.openSwitch());
  );
 );


 --This is when faults are fixed
 --This is done manually as the engineer need to report
 --to control center when the physical fault has been fixed

 public recovery : () ==> ()
 recovery() ==
 (
   switch1.reboot();
   switch2.reboot();
   switch3.reboot();
   priorityMap.generateConfigurations();
 );

end Controller
```

4 DependabilityCage

```
class DependabilityCage

instance variables
private testedBehaviour : set of Configuration;
private mode : token := mk_token(<IGNORE>);
```

```
--private mode : token := <CAUTIOUS>;

operations

public DependabilityCage : () ==> DependabilityCage
DependabilityCage() == (
 testedBehaviour := {
  new Configuration(true,true,false),
  new Configuration(false,false,true)
 };
);


public dependabilityControl : Configuration ==> bool
dependabilityControl(conf) == (
 if(mode =  mk_token(<IGNORE>)) then
 (
  --Naive approach, always allow all behaviour
  --including untested behaviour
  return true;
 )else if (mode =  mk_token(<CAUTIOUS>)) then
 (
  --Extreme approach, completely stop untested behaviour
  return conf in set testedBehaviour;
 );
 return false;
);

end DependabilityCage
```

5 HardwareInterface

```
class HardwareInterface

-- INPUT VARIABLES
instance variables

  -- @ interface: type = input, name="current1In";
  public current1In : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="current2In";
  public current2In : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="current3In";
  public current3In : RealPort := new RealPort(0.0);


-- OUTPUT VARIABLES
instance variables

  -- @ interface: type = output, name="SS1Out";
  public SS1Out : RealPort := new RealPort(0.0);

  -- @ interface: type = output, name="SS2Out";
  public SS2Out : RealPort := new RealPort(0.0);

  -- @ interface: type = output, name="SS3Out";
  public SS3Out : RealPort := new RealPort(0.0);

  -- @ interface: type = output, name="faultOut";
  public faultOut : RealPort := new RealPort(0.0);

end HardwareInterface
```

6 PriorityMap

```
class PriorityMap

instance variables

protected configurationPriority : map Configuration to int;

protected sectionPriority : seq of int;
--industrial area
protected section1prio : int := 10;
--4 residential homes
protected section2prio : int := 4;
--hospital
protected section3prio : int := 20;
--6 residential homes
protected section4prio : int := 6;

private priorityCount: int;
private numberOfSwitch : nat := 3;

private bestConfig: Configuration;

operations

-- constructor for PWM

public PriorityMap : () ==> PriorityMap
PriorityMap() == (
 sectionPriority := [section1prio, section2prio, section3prio, section4prio];

 generateConfigurations();
);

--take conf and calculate its priority based on the
--grid sections that will receive electricity
--from that configuration

public calculatePriority : Configuration ==> int
calculatePriority(conf) == (

 --Safety check, at least one switch must be open
 --note: false value means the switch is open
 if(conf.getConfValue() = [true,true,true]) then return -1;
 priorityCount := 0;

 let currentConfig = conf.getConfValue()
  in
 (
  --calculate electricity frow from gen 1
  dcl flag: bool := true;
  dcl count: nat := 1;
  while(flag and count < len currentConfig) do
   (
    if(currentConfig(count) = true) then (
     priorityCount := priorityCount + sectionPriority(count+1);
    )else(
     flag := false;
    );
    count := count + 1;
  );

  --calculate electricity frow from gen 2
  flag := true;
  count := len currentConfig;
  while(flag and count > 1) do
  (
    if(currentConfig(count) = true) then (
     priorityCount := priorityCount + sectionPriority(count-1);
    )else(
     flag := false;
    );
    count := count - 1;
  );
  );

  return  priorityCount;

);


public generateConfigurations : () ==> ()
generateConfigurations() == (

 dcl s1 : bool;
 dcl s2 : bool;
 dcl s3 : bool;

 dcl tempConf : Configuration := new Configuration(false,false,false);
 --dcl returnPrio : map Configuration to int;
 configurationPriority := { tempConf |-> calculatePriority(tempConf)};

 for n = 1 to (2 ** numberOfSwitch) - 1 do
 (
  s1 := (floor(n /4) mod 2) = 1;
  s2 := (floor(n /2) mod 2) = 1;
  s3 := (n mod 2) = 1;

  tempConf := new Configuration(s1,s2,s3);
  configurationPriority := configurationPriority munion { tempConf |-> calculatePriority(tempConf
      )};
 );
 setBestConfig();

);

public generateConfigurations : seq of bool ==> ()
generateConfigurations(emergencySwitch) == (

 dcl s1 : bool;
 dcl s2 : bool;
 dcl s3 : bool;
 dcl isInvalid : bool;

 dcl tempConf : Configuration := new Configuration(false,false,false);
 configurationPriority := { tempConf |-> calculatePriority(tempConf)};

 for n = 1 to (2 ** numberOfSwitch) - 1 do
 (
  s1 := (floor(n /4) mod 2) = 1;
  s2 := (floor(n /2) mod 2) = 1;
  s3 := (n mod 2) = 1;

  --Dont add to the calculation if it the config uses
  --switch that is in emergency
  if(s1 and not emergencySwitch(1)) then isInvalid := true;
  if(s2 and not emergencySwitch(2)) then isInvalid := true;
  if(s3 and not emergencySwitch(3)) then isInvalid := true;

  if(not isInvalid) then (
   tempConf := new Configuration(s1,s2,s3);
   configurationPriority := configurationPriority munion { tempConf |-> calculatePriority(
       tempConf)};
  )

 );
 setBestConfig();

);


public getPriority : Configuration ==> int
getPriority(conf) == (
 return configurationPriority(conf);
);


public setBestConfig : () ==> ()
setBestConfig() == (
 dcl tempConf : Configuration;
 dcl priority : int := 0;

 for all conf in set dom configurationPriority do (
   if(configurationPriority(conf) > priority) then (
    tempConf := conf;
    priority := configurationPriority(conf);
   );
 );

 bestConfig := tempConf;
);


public getBestConfig : () ==> Configuration
getBestConfig() == (
 return bestConfig;
);

end PriorityMap
```

7 Sensor

```
class Sensor

instance variables

-- this value will be set through the co-simulation
protected port: RealPort;

-- local copy of the shared variable
protected local_val: real := 0;
```

```
operations

-- constructor for Sensor

public Sensor: RealPort ==> Sensor
Sensor(p) == (
  port := p;
  local_val := p.getValue();
);

-- get sensor value

public GetValue: () ==> real
GetValue() == return local_val;

-- write local variable to shared variable

public Sync: () ==> ()
Sync() == local_val := port.getValue();

end Sensor
```

8 Switch

```
class Switch

instance variables
 resistance  : real := 0.00000001;
 currents : seq of real := [1,1,0];
 emergencyState : bool := false;
 maxCurrent : real;

 -- this value will be set through the co-simulation
 protected port: RealPort;

 -- local copy of the shared variable
 protected local_val: real := 0;


values
 private open : real = 100000000.0;
 private close : real = 0.00000001;

operations


 public Switch : RealPort ==> Switch
 Switch(p) == (
  port := p;
 );


 public Sync: () ==> ()
 Sync() == port.setValue(resistance);


 public openSwitch : () ==> ()
 openSwitch() ==
 (
  resistance := open;
 );


 public closeSwitch : () ==> ()
 closeSwitch() ==
 (
  resistance := close;
 );


 public getState : () ==> real
 getState() ==
 (
  return resistance;
 );


 public setCurrent : real ==> ()
 setCurrent(a) ==
 (
  currents := tl currents ^ [a];
  checkState();
 );


 public isClosed : () ==> bool
 isClosed() ==
 (
  if resistance < 1 then return true else return false;
 );


 public checkState : () ==> ()
 checkState() ==
 (
  if (getCurrent() < 1 or getCurrent() > 100000) and
  isClosed() then (
   emergencyState := true;
   openSwitch();
  );
 );


 public reboot : () ==> ()
 reboot() ==
 (
  emergencyState := false;
 );


 public isInEmergency : () ==> bool
 isInEmergency() ==
 (
  return emergencyState;
 );


 public getCurrent : () ==> real
 getCurrent() ==
 (
  maxCurrent := currents(1);
  if currents(2) > maxCurrent then maxCurrent := currents(2);
  if currents(3) > maxCurrent then maxCurrent := currents(3);
  return maxCurrent;
 );


 public isEmergency : () ==> real
 isEmergency() ==
 (
  if emergencyState then return 1 else return 0;
 );


end Switch
```

9 System

```
system System

instance variables

-- Hardware interface variable required by FMU Import/Export
public static hwi: HardwareInterface := new HardwareInterface();

-- controller
public static controller : [Controller] := nil;
public static mainthread: [Thread] := nil;

-- architecture
cpu : CPU := new CPU(<FP>, 1E6);


operations


public System : () ==> System
System () ==
 (

  let current1 = new Sensor(hwi.current1In),
    current2 = new Sensor(hwi.current2In),
    current3 = new Sensor(hwi.current3In),
    fault = new Actuator(hwi.faultOut),
    SS1 = new Switch(hwi.SS1Out),
    SS2 = new Switch(hwi.SS2Out),
    SS3 = new Switch(hwi.SS3Out)
  in
 (
  controller := new Controller(current1, current2, current3, fault , SS1 , SS2 , SS3);

   mainthread := new Thread(25, controller,current1, current2, current3, fault , SS1 , SS2 , SS3
       );
  );
  -- deploy the controller
  cpu.deploy(mainthread);
 );

end System
```

10 Thread

```
class Thread

instance variables

-- thread period
private period: nat := 1E9;

-- sensors
public static curr1: [Sensor] := nil;
public static curr2: [Sensor] := nil;
public static curr3: [Sensor] := nil;

-- actuators
public static fault: [Actuator] := nil;
public static switch1: [Switch] := nil;
public static switch2: [Switch] := nil;
public static switch3: [Switch] := nil;

private controller: [Controller] := nil;

operations

-- constructor for Thread

public Thread: nat * Controller * Sensor * Sensor * Sensor * Actuator * Switch * Switch * Switch
     ==> Thread
Thread(f,c,c1,c2,c3,faultData,s1,s2,s3) == (
 period := frequency_to_period(f);
 curr1 := c1;
 curr2 := c2;
  curr3 := c3;
 fault := faultData;
 switch1 := s1;
 switch2 := s2;
 switch3 := s3;
 controller := c
);

-- control loop

Step: () ==> ()
Step() == (
  curr1.Sync();
  curr2.Sync();
  curr3.Sync();
  fault.Sync();
  switch1.Sync();
  switch2.Sync();
  switch3.Sync();

  controller.Step();
);

-- run as a periodic thread
thread periodic(period, 0 ,0, 0)(Step);

functions

-- convert frequency to period in nanoseconds

private frequency_to_period: real -> nat
```

```
frequency_to_period(f) == floor 1E9/f

end Thread
```

11 World

```
class World
operations

-- run a simulation

public run : () ==> ()
run() ==
 (start(System`mainthread);
  block();
 );

-- wait for simulation to finish

block : () ==> ()
block() == skip;

sync per block => false;

end World
```

B.2 20-sim Models

B.2.1 SmartGrid.emx

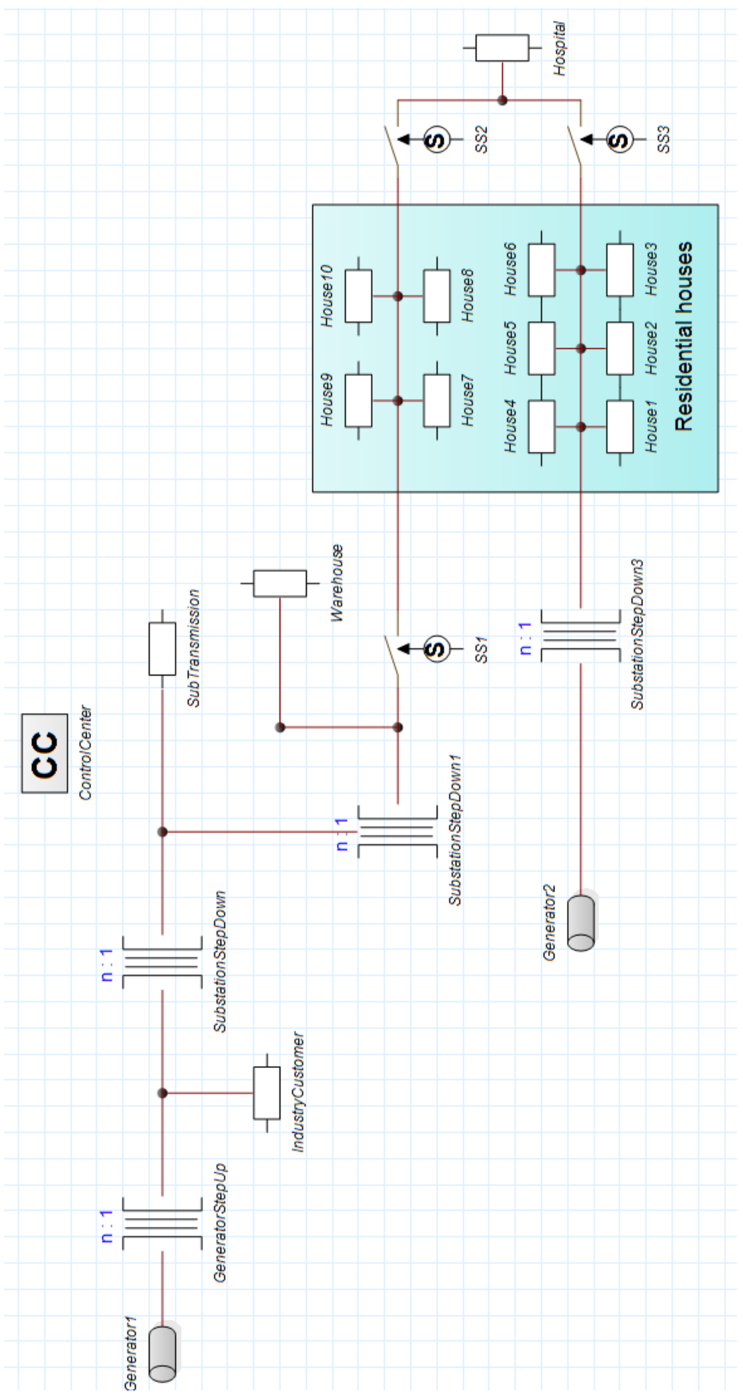


Figure B.1: Top level model

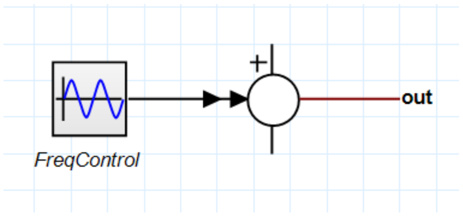


Figure B.2: Generator

B.2.2 Generator type (for Generator1 and Generator2)

B.2.3 Component values

Name	**Type**	**Parameter**	**Values**	**Details**
FreqControl	WaveGenerator	Amplitude	25000	Represent the peak Voltage level to be generated by the generators
FreqControl	WaveGenerator	Omega	314.159265 rad/s	Or the equivalent to 50Hz (50Hz x 2π = 314.159265 rad/s)
GeneratorStepUp	Transformer	Ratio	11	Step up 25kV to 275kV
SubstationStepDown	Transformer	Ratio	1/25	Step down 275kV to 11kV
SubstationStepDown1	Transformer	Ratio	230/11000	Step down 11kV to 230V
SubstationStepDown3	Transformer	Ratio	230/25000	Step down 25kV to 230V
IndustryCustomer	Load	Resistance	270089.286Ω	
SubTransmission	Load	Resistance	121000Ω	
Warehouse	Load	Resistance	0.176Ω	
Hospital	Load	Resistance	0.176Ω	
Houses	Load	Resistance	5.29Ω	

Table B.1: Values of each component in the model

Appendix: SAR UAV - VDM-RT models

C

This appendix details the VDM-RT models used in Chapter 6.3 in the SAR UAVs case study. The 20-Sim model is not included in this thesis as the model is left unmodified from the base model. The base model is obtained from the INTO-CPS example project available for download from the INTO-CPS COE. Documentation on the base model of the UAV swarm can be found in INTO-CPS examples compendium 3 [272].

C.1 UAVGlobalController

UAVGlobalController

November 5, 2019

Contents

1 Command

```
class Command

instance variables

-- this value will be set through the co-simulation
protected port: RealPort;

-- local copy of the shared variable
protected local_val: real := 0;

operations

-- constructor for PWM

public Command : RealPort ==> Command
Command(p) == (
 port := p;
);

-- set actuator value

public SetValue: real ==> ()
SetValue(v) ==  local_val := v;


public Read: () ==> real
Read() == return local_val;
```

```
-- write local variable to shared variable

public Sync: () ==> ()
Sync() == port.setValue(local_val);

end Command
```

2 Controller

```
class Controller

instance variables

-- command outputs
public uav1TargetX: [Command] := nil;
public uav1TargetY: [Command] := nil;
public uav1TargetZ: [Command] := nil;
public uav2TargetX: [Command] := nil;
public uav2TargetY: [Command] := nil;
public uav2TargetZ: [Command] := nil;
public uav3TargetX: [Command] := nil;
public uav3TargetY: [Command] := nil;
public uav3TargetZ: [Command] := nil;
public uav4TargetX: [Command] := nil;
public uav4TargetY: [Command] := nil;
public uav4TargetZ: [Command] := nil;
public uav5TargetX: [Command] := nil;
public uav5TargetY: [Command] := nil;
public uav5TargetZ: [Command] := nil;
public rad: int := 15;

public uav1ToiX: [Sensor] := nil;
public uav1ToiY: [Sensor] := nil;
public uav2ToiX: [Sensor] := nil;
public uav2ToiY: [Sensor] := nil;

public uav3ToiX: [Sensor] := nil;
public uav3ToiY: [Sensor] := nil;
public uav4ToiX: [Sensor] := nil;
public uav4ToiY: [Sensor] := nil;
public uav5ToiX: [Sensor] := nil;
public uav5ToiY: [Sensor] := nil;

operations

 -- constructor for Controller
 public Controller : Command * Command * Command * Command *
     Command * Command* Command *
     Command * Command* Command *
     Command * Command* Command *
     Command * Command
     * Sensor * Sensor
     * Sensor * Sensor

     * Sensor * Sensor
     * Sensor * Sensor
     * Sensor * Sensor ==> Controller
 Controller (uav1X, uav1Y, uav1Z, uav2X, uav2Y, uav2Z, uav3X, uav3Y, uav3Z, uav4X, uav4Y, uav4Z,
     uav5X, uav5Y, uav5Z,
     UAV1ToiX,UAV1ToiY,
     UAV2ToiX,UAV2ToiY,
     UAV3ToiX,UAV3ToiY,
     UAV4ToiX,UAV4ToiY,
     UAV5ToiX,UAV5ToiY) == (

 uav1TargetX := uav1X;
 uav1TargetY := uav1Y;
 uav1TargetZ := uav1Z;
 uav2TargetX := uav2X;
 uav2TargetY := uav2Y;
 uav3TargetZ := uav3Z;
 uav3TargetX := uav3X;
 uav3TargetY := uav3Y;
 uav2TargetZ := uav2Z;
 uav4TargetX := uav4X;
 uav4TargetY := uav4Y;
 uav4TargetZ := uav4Z;
 uav5TargetX := uav5X;
 uav5TargetY := uav5Y;
 uav5TargetZ := uav5Z;

 uav1ToiX := UAV1ToiX;
 uav1ToiY := UAV1ToiY;
 uav2ToiX := UAV2ToiX;
 uav2ToiY := UAV2ToiY;
 uav3ToiX := UAV3ToiX;
 uav3ToiY := UAV3ToiY;
 uav4ToiX := UAV4ToiX;
 uav4ToiY := UAV4ToiY;
 uav5ToiX := UAV5ToiX;
 uav5ToiY := UAV5ToiY;

 );

 -- MAIN CONTROL LOOP
 public Step : () ==> ()
 Step() ==
 (
  setMissionGround();

  sendRescueIfFound(uav1ToiX.GetValue(),uav1ToiY.GetValue());
  sendRescueIfFound(uav2ToiX.GetValue(),uav2ToiY.GetValue());
  sendRescueIfFound(uav3ToiX.GetValue(),uav3ToiY.GetValue());
  sendRescueIfFound(uav4ToiX.GetValue(),uav4ToiY.GetValue());

 );

 public setMissionGround: () ==> ()
 setMissionGround() == (
  let positiveBound = (5 + rad), negativeBound = (5 - rad) in
  (
   uav1TargetX.SetValue(positiveBound);
   uav1TargetY.SetValue(positiveBound);
   uav1TargetZ.SetValue(4.0);

   uav2TargetX.SetValue(negativeBound);
   uav2TargetY.SetValue(negativeBound);
   uav2TargetZ.SetValue(4.0);


   uav3TargetX.SetValue(negativeBound);
   uav3TargetY.SetValue(positiveBound);
   uav3TargetZ.SetValue(4.0);
```

```
   uav4TargetX.SetValue(positiveBound);
   uav4TargetY.SetValue(negativeBound);
   uav4TargetZ.SetValue(4.0);


  );
 );

 public idleUAV: () ==> ()
 idleUAV() == (
  uav5TargetX.SetValue(5.0);
  uav5TargetY.SetValue(5.0);
  uav5TargetZ.SetValue(4.0);
 );

 public sendRescueIfFound: real * real  ==> ()
  sendRescueIfFound(x,y) == (
   if(x <> 0 or y <> 0) then (
    uav5TargetX.SetValue(x);
    uav5TargetY.SetValue(y);
   )else(
    idleUAV();
   );
 );


end Controller
```

3 HardwareInterface

```
class HardwareInterface

  -- INPUT VARIABLES
instance variables

  -- @ interface: type = input, name="uav1ToiX";
  public uav1ToiX : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="uav1ToiY";
  public uav1ToiY : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="uav2ToiX";
  public uav2ToiX : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="uav2ToiY";
  public uav2ToiY : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="uav3ToiX";
  public uav3ToiX : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="uav3ToiY";
  public uav3ToiY : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="uav4ToiX";
  public uav4ToiX : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="uav4ToiY";
  public uav4ToiY : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="uav5ToiX";
  public uav5ToiX : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="uav5ToiY";
  public uav5ToiY : RealPort := new RealPort(0.0);

-- OUTPUT VARIABLES
instance variables

  -- @ interface: type = output, name="uav1TargetX";
  public uav1TargetX : RealPort := new RealPort(0.0);
  -- @ interface: type = output, name="uav1TargetY";
  public uav1TargetY : RealPort := new RealPort(0.0);
  -- @ interface: type = output, name="uav1TargetZ";
  public uav1TargetZ : RealPort := new RealPort(0.0);

    -- @ interface: type = output, name="uav2TargetX";
  public uav2TargetX : RealPort := new RealPort(0.0);
  -- @ interface: type = output, name="uav2TargetY";
  public uav2TargetY : RealPort := new RealPort(0.0);
  -- @ interface: type = output, name="uav2TargetZ";
  public uav2TargetZ : RealPort := new RealPort(0.0);

    -- @ interface: type = output, name="uav3TargetX";
  public uav3TargetX : RealPort := new RealPort(0.0);
  -- @ interface: type = output, name="uav3TargetY";
  public uav3TargetY : RealPort := new RealPort(0.0);
  -- @ interface: type = output, name="uav3TargetZ";
  public uav3TargetZ : RealPort := new RealPort(0.0);



    -- @ interface: type = output, name="uav4TargetX";
  public uav4TargetX : RealPort := new RealPort(0.0);
  -- @ interface: type = output, name="uav4TargetY";
  public uav4TargetY : RealPort := new RealPort(0.0);
  -- @ interface: type = output, name="uav4TargetZ";
  public uav4TargetZ : RealPort := new RealPort(0.0);



    -- @ interface: type = output, name="uav5TargetX";
  public uav5TargetX : RealPort := new RealPort(0.0);
  -- @ interface: type = output, name="uav5TargetY";
  public uav5TargetY : RealPort := new RealPort(0.0);
  -- @ interface: type = output, name="uav5TargetZ";
  public uav5TargetZ : RealPort := new RealPort(0.0);





end HardwareInterface
```

4 Sensor

```
class Sensor

instance variables

-- this value will be set through the co-simulation
```

```
protected port: RealPort;

-- local copy of the shared variable
protected local_val: real := 0;

operations

-- constructor for Sensor

public Sensor: RealPort ==> Sensor
Sensor(p) == (
  port := p;
  local_val := p.getValue();
);

-- get sensor value

public GetValue: () ==> real
GetValue() == return local_val;

-- write local variable to shared variable

public Sync: () ==> ()
Sync() == local_val := port.getValue();

end Sensor
```

5 System

```
system System

instance variables

  -- Hardware interface variable required by FMU Import/Export
  public static hwi: HardwareInterface := new HardwareInterface();

  -- controller
  public static controller : [Controller] := nil;
  public static mainthread: [Thread] := nil;

  -- architecture
  cpu : CPU := new CPU(<FP>, 1E6);


operations


  public System : () ==> System
  System () ==
  (--settings := World'settings;

    let uav1TargetX = new Command(hwi.uav1TargetX),
        uav1TargetY = new Command(hwi.uav1TargetY),
        uav1TargetZ = new Command(hwi.uav1TargetZ),
        uav2TargetX = new Command(hwi.uav2TargetX),
        uav2TargetY = new Command(hwi.uav2TargetY),
        uav2TargetZ = new Command(hwi.uav2TargetZ),
        uav3TargetX = new Command(hwi.uav3TargetX),
        uav3TargetY = new Command(hwi.uav3TargetY),
        uav3TargetZ = new Command(hwi.uav3TargetZ),
        uav4TargetX = new Command(hwi.uav4TargetX),
        uav4TargetY = new Command(hwi.uav4TargetY),
        uav4TargetZ = new Command(hwi.uav4TargetZ),
        uav5TargetX = new Command(hwi.uav5TargetX),
        uav5TargetY = new Command(hwi.uav5TargetY),
        uav5TargetZ = new Command(hwi.uav5TargetZ),
     uav1ToiX =  new Sensor(hwi.uav1ToiX),
     uav1ToiY =  new Sensor(hwi.uav1ToiY),
     uav2ToiX =  new Sensor(hwi.uav2ToiX),
     uav2ToiY =  new Sensor(hwi.uav2ToiY),
     uav3ToiX =  new Sensor(hwi.uav3ToiX),
     uav3ToiY =  new Sensor(hwi.uav3ToiY),
     uav4ToiX =  new Sensor(hwi.uav4ToiX),
     uav4ToiY =  new Sensor(hwi.uav4ToiY),
     uav5ToiX =  new Sensor(hwi.uav5ToiX),
     uav5ToiY =  new Sensor(hwi.uav5ToiY)
  in
 (
  controller := new Controller(uav1TargetX, uav1TargetY, uav1TargetZ,
               uav2TargetX ,uav2TargetY, uav2TargetZ,
               uav3TargetX ,uav3TargetY, uav3TargetZ,
               uav4TargetX ,uav4TargetY, uav4TargetZ,
               uav5TargetX ,uav5TargetY, uav5TargetZ,
               uav1ToiX,uav1ToiY,
               uav2ToiX,uav2ToiY,
               uav3ToiX,uav3ToiY,
               uav4ToiX,uav4ToiY,
               uav5ToiX,uav5ToiY
               );

   mainthread := new Thread(25, controller, uav1TargetX, uav1TargetY, uav1TargetZ,
               uav2TargetX ,uav2TargetY, uav2TargetZ,
               uav3TargetX ,uav3TargetY, uav3TargetZ,
               uav4TargetX ,uav4TargetY, uav4TargetZ,
               uav5TargetX ,uav5TargetY, uav5TargetZ,
               uav1ToiX,uav1ToiY,
               uav2ToiX,uav2ToiY,
               uav3ToiX,uav3ToiY,
               uav4ToiX,uav4ToiY,
               uav5ToiX,uav5ToiY
               );
  );
  -- deploy the controller
  cpu.deploy(mainthread);
 );

end System
```

6 Thread

```
class Thread

instance variables

-- thread period
private period: nat := 1E9;

-- command outputs
public static uav1TargetX: [Command] := nil;
public static uav1TargetY: [Command] := nil;
public static uav1TargetZ: [Command] := nil;
public static uav2TargetX: [Command] := nil;
public static uav2TargetY: [Command] := nil;
public static uav2TargetZ: [Command] := nil;
public static uav3TargetX: [Command] := nil;
public static uav3TargetY: [Command] := nil;
public static uav3TargetZ: [Command] := nil;
public static uav4TargetX: [Command] := nil;
public static uav4TargetY: [Command] := nil;
public static uav4TargetZ: [Command] := nil;
public static uav5TargetX: [Command] := nil;
public static uav5TargetY: [Command] := nil;
public static uav5TargetZ: [Command] := nil;


public static uav1ToiX: [Sensor] := nil;
public static uav1ToiY: [Sensor] := nil;
public static uav2ToiX: [Sensor] := nil;
public static uav2ToiY: [Sensor] := nil;
public static uav3ToiX: [Sensor] := nil;
public static uav3ToiY: [Sensor] := nil;
public static uav4ToiX: [Sensor] := nil;
public static uav4ToiY: [Sensor] := nil;
public static uav5ToiX: [Sensor] := nil;
public static uav5ToiY: [Sensor] := nil;

private controller: [Controller] := nil;


operations

-- constructor for Thread
public Thread: nat * Controller * Command * Command * Command * Command *
     Command * Command* Command *
     Command * Command* Command *
     Command * Command* Command *
     Command * Command
     * Sensor * Sensor
     * Sensor * Sensor
     * Sensor * Sensor
     * Sensor * Sensor
     * Sensor * Sensor ==> Thread
Thread(f,c, uav1X, uav1Y, uav1Z, uav2X, uav2Y, uav2Z, uav3X, uav3Y, uav3Z, uav4X, uav4Y, uav4Z,
    uav5X, uav5Y, uav5Z,
     UAV1ToiX,UAV1ToiY,
     UAV2ToiX,UAV2ToiY,
     UAV3ToiX,UAV3ToiY,
     UAV4ToiX,UAV4ToiY,
     UAV5ToiX,UAV5ToiY) == (

 period := frequency_to_period(f);
 uav1TargetX := uav1X;
 uav1TargetY := uav1Y;
 uav1TargetZ := uav1Z;
 uav2TargetX := uav2X;
 uav2TargetY := uav2Y;
 uav2TargetZ := uav2Z;
 uav3TargetX := uav3X;
 uav3TargetY := uav3Y;
 uav3TargetZ := uav3Z;
 uav4TargetX := uav4X;
 uav4TargetY := uav4Y;
 uav4TargetZ := uav4Z;
 uav5TargetX := uav5X;
 uav5TargetY := uav5Y;
 uav5TargetZ := uav5Z;

 uav1ToiX := UAV1ToiX;
 uav1ToiY := UAV1ToiY;
 uav2ToiX := UAV2ToiX;
 uav2ToiY := UAV2ToiY;
 uav3ToiX := UAV3ToiX;
 uav3ToiY := UAV3ToiY;
 uav4ToiX := UAV4ToiX;
 uav4ToiY := UAV4ToiY;
 uav5ToiX := UAV5ToiX;
 uav5ToiY := UAV5ToiY;
 --uav1ToiY,

 controller := c
);

-- control loop
Step: () ==> ()
Step() == (
  uav1TargetX.Sync();
  uav1TargetY.Sync();
  uav1TargetZ.Sync();
  uav2TargetX.Sync();
  uav2TargetY.Sync();
  uav2TargetZ.Sync();
  uav3TargetX.Sync();
  uav3TargetY.Sync();
  uav3TargetZ.Sync();
  uav4TargetX.Sync();
  uav4TargetY.Sync();
  uav4TargetZ.Sync();
  uav5TargetX.Sync();
  uav5TargetY.Sync();
  uav5TargetZ.Sync();

  uav1ToiX.Sync();
  uav1ToiY.Sync();
  uav2ToiX.Sync();
  uav2ToiY.Sync();
  uav3ToiX.Sync();
  uav3ToiY.Sync();
  uav4ToiX.Sync();
  uav4ToiY.Sync();
  uav5ToiX.Sync();
  uav5ToiY.Sync();

  controller.Step();
);

-- run as a periodic thread
thread periodic(period, 0 ,0, 0)(Step);

functions

-- convert frequency to period in nanoseconds
private frequency_to_period: real -> nat
frequency_to_period(f) == floor 1E9/f

end Thread
```

7 World

```
class World
operations

-- run a simulation

public run : () ==> ()
run() ==
 (start(System`mainthread);
  block();
 );

-- wait for simulation to finish

block : () ==> ()
block() == skip;

sync per block => false;

end World
```

C.2 UAV_rescue

UAV_rescue

November 5, 2019

Contents

1 Actuator

```
class Actuator

instance variables

-- this value will be set through the co-simulation
protected port: RealPort;

-- local copy of the shared variable
protected local_val: real := 0;

operations

-- constructor for PWM

public Actuator : RealPort ==> Actuator
Actuator(p) == (
 port := p;
);
```

```
-- set actuator value

public SetValue: real ==> ()
SetValue(v) ==  local_val := v;


public Read: () ==> real
Read() == return local_val;

-- write local variable to shared variable

public Sync: () ==> ()
Sync() == port.setValue(local_val);

end Actuator
```

2 Command

```
class Command

instance variables

-- this value will be set through the co-simulation
protected port: RealPort;

-- local copy of the shared variable
protected local_val: real := 0;

operations

-- constructor for Command

public Command: RealPort ==> Command
Command(p) == (
  port := p;
  local_val := port.getValue();
);

-- get Command value

public GetValue: () ==> real
GetValue() == return local_val;

-- write local variable to shared variable
public Sync: () ==> ()
Sync() == local_val := port.getValue();


end Command
```

3 Controller

```
class Controller
```

```
types

instance variables

 Altitude :real:=1.0;
 --10% angle then down to 1%
 -- RollAng = +4 * Sqrt-1(0.1 / Sqrt(2)) - pi
 -- PitchAng Inverse above.
 RollAng  :real:=0.2; -- 3.0
 PitchAng :real:=0; -- 2.85
 CarlAngleTemp :real := 0.2;
 ThrottleHigh :real:=0.63;
 ThrottleLow  :real:=0.37;
 X :real:=0.0;
 Y :real:=0.0;

 throttleIpart : real := 0;
 throttlePpart : real := 0;
 throttleDpart : real := 0;

 xIpart : real := 0;
 xPpart : real := 0;
 xDpart : real := 0;
 desiredPitch : real := 0;


 yIpart : real := 0;
 yPpart : real := 0;
 yDpart : real := 0;
 desiredRoll : real := 0;


 switchTime : real := 5E9;

  tarX: Command;
  tarY: Command;
  tarZ: Command;

 velX: Sensor;
 velY: Sensor;
 velZ: Sensor;
 posX: Sensor;

 posY: Sensor;
 posZ: Sensor;
 yaw: Sensor;

 pitchOut: Actuator;
 rollOut: Actuator;
 yawOut: Actuator;
 throttleOut: Actuator;

 toiX: Actuator;
 toiY: Actuator;

 targetMap: Map;
 flag1 : bool := true;
 flag2 : bool := true;
 flag3 : bool := true;
 flag4 : bool := true;

 targetX : real := 5;
 targetY : real := 5;
 targetZ : real := 0.5;
 precision : real := 0.3;
 obstacles : set of seq of real;


operations

 -- constructor for Controller
 public Controller : Command * Command * Command * Sensor * Sensor * Sensor * Sensor *
     Sensor * Sensor * Sensor * Actuator * Actuator * Actuator * Actuator
     * Actuator * Actuator ==> Controller
 Controller (TarX, TarY, TarZ,VelX,VelY,VelZ,PosX,PosY,PosZ,Yaw,Pitchout,Rollout,Yawout,
     Throttleout, toix, toiy) ==
  (
    tarX := TarX;
    tarY := TarY;
    tarZ := TarZ;
  velX := VelX;
  velY := VelY;
  velZ := VelZ;
  posX := PosX;
  posY := PosY;
  posZ := PosZ;
  yaw := Yaw;

  pitchOut := Pitchout;
  rollOut := Rollout;
  yawOut := Yawout;
  throttleOut := Throttleout;

  toiX:= toix;
  toiY:= toiy;

  targetMap := new Map();
  obstacles := new Environment().getObstacles();
  );

 public GetThrottleOut: () ==> Actuator
 GetThrottleOut() == return throttleOut;


 public moveUAV: () ==> ()
 moveUAV() ==

 (
   setThrottleNew();
  setPitchNew();
  setRollNew();
  SetAltitude(2);
 );

 -- MAIN CONTROL LOOP
 public Step : () ==> ()
 Step() ==
 (
  targetX := tarX.GetValue();
  targetY := tarY.GetValue();

   moveUAV();

 );


 public setThrottleNew: () ==> ()
 setThrottleNew () == (
   throttleIpart := throttleIpart + ((targetX-posZ.GetValue()) * 0.03);
    throttlePpart := (targetZ - posZ.GetValue()) * 1.2;
    if velZ.GetValue() > 0 then(
     throttleDpart := velZ.GetValue() * 0.5
    )
    else(
     throttleDpart := velZ.GetValue() * 1.0
    );
    GetThrottleOut().SetValue(throttleIpart + throttlePpart - throttleDpart)
 );



 public setPitchNew: () ==>  ()
 setPitchNew() == (
  --yIpart := yIpart + ((posY.GetValue()-targetY) * 0.00);
    yPpart := ( posY.GetValue()- targetY) * 0.28;
    yDpart := velY.GetValue() * -0.2;
    desiredPitch := yIpart + yPpart - yDpart;
    if desiredPitch > 0.35 then
     desiredPitch := 0.35;
    if desiredPitch < -0.35 then
     desiredPitch := -0.35;
   pitchOut.SetValue(desiredPitch)
 );

 public setRollNew: () ==>  ()
 setRollNew() == (
  --xIpart := xIpart + (targetX - (posX.GetValue()) * 0.00);
    xPpart := (targetX - posX.GetValue()) * 0.28;
    xDpart := velX.GetValue() * 0.2;
    desiredRoll := xIpart + xPpart - xDpart;
    if desiredRoll > 0.35 then
     desiredRoll := 0.35;
    if desiredRoll < -0.35 then
     desiredRoll := -0.35;
   rollOut.SetValue(desiredRoll)
 );


 public Upwards: real ==> ()
 Upwards(throttle) == (
    GetThrottleOut().SetValue(throttle);
 );

 public MoveX: real ==> ()
 MoveX(x) == (
  if x <> 0.0 then
   if x - posX.GetValue()  < x*0.01 then
    RollAng := CarlAngleTemp; --3.1;
  if x <> 0.0 then
   if x > posX.GetValue() then
    if velX.GetValue() < abs(posX.GetValue()-x)+0.01 then
     rollOut.SetValue(CarlAngleTemp)--(RollAng)
    else if posX.GetValue() > (x) or velX.GetValue() > (posX.GetValue()-x)  then
     rollOut.SetValue(-CarlAngleTemp)
    else rollOut.SetValue(CarlAngleTemp)
     else if velX.GetValue() > (x - posX.GetValue())+0.01 then
      rollOut.SetValue(-CarlAngleTemp)
  else if posX.GetValue() < (x) or velX.GetValue() < (x - posX.GetValue())  then
   rollOut.SetValue(CarlAngleTemp)
  else rollOut.SetValue(-CarlAngleTemp);
 );

 public MoveY: real ==> ()
 MoveY(y) == (
  if y <> 0.0 then
   if y - posY.GetValue() < y*0.01 then
    PitchAng := CarlAngleTemp;
  if y <> 0.0 then
   if y < posY.GetValue() then
    if velY.GetValue() > abs(posY.GetValue()-y)+0.01 then
     pitchOut.SetValue(CarlAngleTemp)
    else if posY.GetValue() > (y) or velY.GetValue() > (posY.GetValue()-y) then
     pitchOut.SetValue(-CarlAngleTemp)
    else pitchOut.SetValue(CarlAngleTemp)
     else if velY.GetValue() > (y - posY.GetValue())+0.01 then
      pitchOut.SetValue(-CarlAngleTemp)
   else if posY.GetValue() < (y) or velY.GetValue() < (y - posY.GetValue())  then
    pitchOut.SetValue(CarlAngleTemp)
   else pitchOut.SetValue(-CarlAngleTemp);

 );

 public SetAltitude: real ==> ()
 SetAltitude(altitude) ==(
  if altitude > posZ.GetValue() then
   if velZ.GetValue() < abs(posZ.GetValue()-altitude)+0.025 then
    Upwards(ThrottleHigh)
   else if posZ.GetValue() > (altitude) or velZ.GetValue() > (posZ.GetValue()-altitude) then
    Upwards(ThrottleLow)
   else Upwards(ThrottleHigh)
     else if velZ.GetValue() > (altitude - posZ.GetValue())+0.025 then
      Upwards(ThrottleLow)
   else if posZ.GetValue() < (altitude) or velZ.GetValue() < (altitude - posZ.GetValue())
   then Upwards(ThrottleHigh)
   else Upwards(ThrottleLow);
);


end Controller
```

4 Coor

```
class Coor

instance variables

-- local copy of the shared variable
protected x : real := 0;
protected y : real := 0;

operations

-- constructor for Command

public Coor: real * real ==> Coor
Coor(xAxis,yAxis) == (
 x := xAxis;
 y := yAxis;
);


public getX: () ==> real
getX() == return x;
```

```
public getY: () ==> real
getY() == return y;


public isEqual: Coor ==> bool
 isEqual(c) == (
  return c.getY() = y and c.getX() = x;
 );

end Coor
```

5 Environment

```
class Environment

instance variables

-- local copy of the shared variable
protected obtacles : set of (seq of real) := {};
protected numOfObstables : int;
protected x : real;
protected y : real;

operations

-- constructor for Command

public Environment: int * int ==> Environment
Environment(n,negativeOffset) == (
 --Create at least 1 obstacles
 numOfObstables := MATH`rand(n) + 1;

 obtacles := {};
 for i = 0 to numOfObstables do
 (
   x := MATH`rand(n) - negativeOffset;
   y := MATH`rand(n) - negativeOffset;
   obtacles := obtacles union {[ x, y ]};
 );

);

public getObstacles: () ==> set of (seq of real)

getObstacles() == return obtacles;

end Environment
```

6 HardwareInterface

```
class HardwareInterface

-- INPUT VARIABLES
instance variables
```

```
  -- @ interface: type = input, name="targetX";
  public targetX : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="targetY";
  public targetY : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="targetZ";
  public targetZ : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="posXIn";
  public posXIn : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="posYIn";
  public posYIn : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="posZIn";
  public posZIn : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="yawIn";
  public yawIn : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="velXIn";
  public velXIn : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="velYIn";
  public velYIn : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="velZIn";
  public velZIn : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="batteryChargeIn";
  public batteryChargeIn : RealPort := new RealPort(0.0);

-- OUTPUT VARIABLES
instance variables

  -- @ interface: type = output, name="pitchOut";
  public pitchOut : RealPort := new RealPort(0.0);

  -- @ interface: type = output, name="rollOut";
  public rollOut : RealPort := new RealPort(0.0);

  -- @ interface: type = output, name="yawOut";
  public yawOut : RealPort := new RealPort(0.0);

  -- @ interface: type = output, name="throttleOut";
  public throttleOut : RealPort := new RealPort(0.0);

  -- @ interface: type = output, name="toiXout";
  public toiXout : RealPort := new RealPort(0.0);

  -- @ interface: type = output, name="toiYout";
  public toiYout : RealPort := new RealPort(0.0);

end HardwareInterface
```

7 Map

```
class Map
```

```
instance variables

-- this value will be set through the co-simulation
protected port: RealPort;

-- local copy of the shared variable
private path: seq of Coor;
private hasTarget: bool;

--Variable for map planning
protected endX: real := 10;
protected endY: real := 10;

private startX: real := 5.0;
private startY: real := 5.0;

private currX: real;
private currY: real;

private xlength: real;
private scale: real := 2;

--Storing the x-axis path
private startToEnd : seq of real := [];
private endToStart : seq of real := [];

--Flag for x-axis flight direction
private xAxisFlag : bool;

--flag for when survivor are found
private idleMode: bool := false;

operations

-- constructor for Command

public Map: () ==> Map
Map() == (
  hasTarget := false;
  path := [new Coor(startX,startY)];
);

public setTargetNew: real * real ==> ()
setTargetNew(x,y) == (

 --If target is the same as current spot, no need to do path planning.
 if(x = targetX() and y = targetY()) then
 (

  hasTarget := true;
 );

 --Ignore input if x and y is 0 because of initial value from cosimulation
 --Might need to change this to be time based
 if( not hasTarget and (x <> 0 or y <> 0) and not idleMode) then (

  hasTarget := true;
  path := [];

  endY := y;
  endX := x;
  --https://playcode.io/463216

  currX := startX;
```

```
   currY := startY;

  xlength := abs(startX - endX);
  for i = 0 to (xlength - 1) by scale do
   (

    if(startX < endX) then (
      startToEnd := startToEnd ^ [startX + i + 1];
      endToStart := endToStart ^ [endX - i];

    )else(

      startToEnd := startToEnd ^ [startX - i - 1];
      endToStart := endToStart ^ [endX + i];
    );
   );

   xAxisFlag := true;
   if(currY < endY) then
   (
    if(currY = startY) then currY := currY + (scale/2);
    while(currY < endY) do
    (
    assignPath()
    );
   )else
   (
    if(currY = startY) then currY := currY - (scale/2);
    while(currY > endY) do
    (
    assignPath();
    );
   );

 );

);

 public assignPath : () ==> ()
  assignPath () ==
  (
   for i = 1 to len startToEnd do (
      if(xAxisFlag) then (
        currX := startToEnd(i);

        path := path ^ [new Coor(currX,currY)];
      ) else (
        currX := endToStart(i);
        path := path ^ [new Coor(currX,currY)];
      );
   );
    xAxisFlag := not xAxisFlag;
    currY := currY + (((endY - currY) / abs(endY - currY)) * scale);
  );

public getCurrentTarget: ()  ==> Coor
getCurrentTarget() == (
 return hd path;
);

public targetX: ()  ==> real

targetX() == (
 return (hd path).getX();
);
```

```
public targetY: ()  ==> real

targetY() == (
 return (hd path).getY();
);

public nextTargetX: ()  ==> real

nextTargetX() == (
 return (hd(tl path)).getX();
);

public nextTargetY: ()  ==> real

nextTargetY() == (
 return (hd(tl path)).getY();
);

public currentTargetReached: ()  ==> ()

currentTargetReached() == (
 --if there is more coordiantes on the plan, go to next one
 if(len path > 1) then
 (
  path := tl path;

 )
 else
 (
  hasTarget := false;
 )
);

public survivorFound: real * real  ==> ()
survivorFound(x,y) == (
 path := [new Coor(x,y)];
 idleMode := true;
);



end Map
```

8 Sensor

```
class Sensor

instance variables

-- this value will be set through the co-simulation
protected port: RealPort;

-- local copy of the shared variable
protected local_val: real := 0;

operations

-- constructor for Sensor
```

```
public Sensor: RealPort ==> Sensor
Sensor(p) == (
  port := p;
  local_val := p.getValue();
);

-- get sensor value

public GetValue: () ==> real
GetValue() == return local_val;

-- write local variable to shared variable

public Sync: () ==> ()
Sync() == local_val := port.getValue();

end Sensor
```

9 System

```
system System

instance variables

-- Hardware interface variable required by FMU Import/Export
public static hwi: HardwareInterface := new HardwareInterface();

-- controller
public static controller : [Controller] := nil;
public static mainthread: [Thread] := nil;

-- architecture
cpu : CPU := new CPU(<FP>, 1E6);


operations


public System : () ==> System
System () ==
 (--settings := World'settings;

  let tarX =  new Command(hwi.targetX),
      tarY =  new Command(hwi.targetY),
    tarZ =  new Command(hwi.targetZ),
    velX =  new Sensor(hwi.velXIn),
      velY =  new Sensor(hwi.velYIn),
    velZ =  new Sensor(hwi.velZIn),
    posX =  new Sensor(hwi.posXIn),
    posY =  new Sensor(hwi.posYIn),
    posZ =  new Sensor(hwi.posZIn),
    yaw =  new Sensor(hwi.yawIn),
    --batteryCharge =  new Sensor(hwi.batteryChargeIn),
    pitchOut = new Actuator(hwi.pitchOut),
    rollOut = new Actuator(hwi.rollOut),
    yawOut = new Actuator(hwi.yawOut),
    throttleOut = new Actuator(hwi.throttleOut),
    toiX = new Actuator(hwi.toiXout),
    toiY = new Actuator(hwi.toiYout)
  in
 (
  controller := new Controller(tarX, tarY, tarZ, velX , velY , velZ ,
            posX , posY , posZ ,
            yaw ,
            pitchOut , rollOut , yawOut , throttleOut,
            toiX, toiY);

   mainthread := new Thread(25, controller, tarX, tarY, tarZ, velX , velY , velZ ,
            posX , posY , posZ ,
            yaw ,
            pitchOut , rollOut , yawOut , throttleOut,
            toiX, toiY);
  );
  -- deploy the controller
  cpu.deploy(mainthread);
 );

end System
```

10 Thread

```
class Thread

instance variables

-- thread period
private period: nat := 1E9;

-- sensors and actuators, controller
-- commands
public static tarX: [Command] := nil;
public static tarY: [Command] := nil;
public static tarZ: [Command] := nil;

-- sensors
public static velX: [Sensor] := nil;
public static velY: [Sensor] := nil;
public static velZ: [Sensor] := nil;
public static posX: [Sensor] := nil;
public static posY: [Sensor] := nil;
public static posZ: [Sensor] := nil;
public static yaw: [Sensor] := nil;

-- actuators

public static pitchOut: [Actuator] := nil;
public static rollOut: [Actuator] := nil;
public static yawOut: [Actuator] := nil;
public static throttleOut: [Actuator] := nil;

public static toiX: [Actuator] := nil;
public static toiY: [Actuator] := nil;

private controller: [Controller] := nil;

operations

-- constructor for Thread
public Thread: nat * Controller * Command * Command * Command * Sensor * Sensor * Sensor * Sensor
     *
     Sensor * Sensor * Sensor *
     Actuator * Actuator * Actuator * Actuator
     * Actuator * Actuator  ==> Thread
Thread(f,c,TarX, TarY, TarZ,VelX,VelY,VelZ,PosX,PosY,PosZ,Yaw,Pitchout,Rollout,Yawout,Throttleout
    ,toix,toiy) == (
 period := frequency_to_period(f);
 tarX := TarX;
 tarY := TarY;
  tarZ  := TarZ;
 velX := VelX;
 velY := VelY;
 velZ := VelZ;
 posX := PosX;
 posY := PosY;
 posZ := PosZ;
 yaw := Yaw;

 pitchOut := Pitchout;
 rollOut := Rollout;
 yawOut := Yawout;

 throttleOut := Throttleout;

 toiX:= toix;
 toiY:= toiy;

 controller := c
);

-- control loop
Step: () ==> ()
Step() == (
  tarX.Sync();
  tarY.Sync();
  tarZ.Sync();
  velX.Sync();
  velY.Sync();
  velZ.Sync();
  posX.Sync();
  posY.Sync();
  posZ.Sync();
  yaw.Sync();
  pitchOut.Sync();
  rollOut.Sync();
  yawOut.Sync();
  throttleOut.Sync();
  toiX.Sync();
  toiY.Sync();

  controller.Step();
);

-- run as a periodic thread
thread periodic(period, 0 ,0, 0)(Step);

functions

-- convert frequency to period in nanoseconds
private frequency_to_period: real -> nat
frequency_to_period(f) == floor 1E9/f

end Thread
```

11 World

```
class World
operations

-- run a simulation

public run : () ==> ()
run() ==
 (start(System`mainthread);
  block();
 );

-- wait for simulation to finish

block : () ==> ()
block() == skip;

sync per block => false;

values

public static q01Xstart : real = 0.0;
public static q01Ystart : real = 0.0;
public static q01Zstart : real = 0.0;
public static q01Thetastart : real = 0.0;

end World
```

C.3 UAV_ConRep

UAVController

November 10, 2019

Contents

1 Actuator

```
class Actuator

instance variables

-- this value will be set through the co-simulation
protected port: RealPort;

-- local copy of the shared variable
protected local_val: real := 0;

operations

-- constructor for PWM

public Actuator : RealPort ==> Actuator
```

```
Actuator(p) == (
 port := p;
);

-- set actuator value

public SetValue: real ==> ()
SetValue(v) ==  local_val := v;


public Read: () ==> real
Read() == return local_val;

-- write local variable to shared variable

public Sync: () ==> ()
Sync() == port.setValue(local_val);

end Actuator
```

2 Command

```
class Command

instance variables

-- this value will be set through the co-simulation
protected port: RealPort;

-- local copy of the shared variable
protected local_val: real := 0;

operations

-- constructor for Command

public Command: RealPort ==> Command
Command(p) == (
  port := p;
  local_val := port.getValue();
);

-- get Command value

public GetValue: () ==> real
GetValue() == return local_val;

-- write local variable to shared variable

public Sync: () ==> ()
Sync() == local_val := port.getValue();

end Command
```

3 Controller

```
class Controller

types

instance variables

 Altitude :real:=1.0;
 --10% angle then down to 1%
 -- RollAng = +4 * Sqrt-1(0.1 / Sqrt(2)) - pi
 -- PitchAng Inverse above.
 RollAng  :real:=0.2; -- 3.0
 PitchAng :real:=0; -- 2.85
 CarlAngleTemp :real := 0.2;
 ThrottleHigh :real:=0.63;
 ThrottleLow  :real:=0.37;
 X :real:=0.0;
 Y :real:=0.0;

 throttleIpart : real := 0;
 throttlePpart : real := 0;
 throttleDpart : real := 0;

 xIpart : real := 0;
 xPpart : real := 0;
 xDpart : real := 0;
 desiredPitch : real := 0;

 yIpart : real := 0;
 yPpart : real := 0;
 yDpart : real := 0;
 desiredRoll : real := 0;


 switchTime : real := 5E9;

  tarX: Command;

  tarY: Command;
  tarZ: Command;

 velX: Sensor;
 velY: Sensor;
 velZ: Sensor;
 posX: Sensor;
 posY: Sensor;
 posZ: Sensor;
 yaw: Sensor;

 pitchOut: Actuator;
 rollOut: Actuator;
 yawOut: Actuator;
 throttleOut: Actuator;

 toiX: Actuator;
 toiY: Actuator;

 targetMap: Map;
 flag1 : bool := true;
 flag2 : bool := true;
 flag3 : bool := true;
 flag4 : bool := true;

 targetX : real := 5;
 targetY : real := 5;
 targetZ : real := 0.5;

 precision : real := 0.3;
 obstacles : set of seq of real;
 survivors: Survivor;


 collisionAvoidanceMode : bool := false;

operations

 -- constructor for Controller
 public Controller : Command * Command * Command * Sensor * Sensor * Sensor * Sensor *
     Sensor * Sensor * Sensor * Actuator * Actuator * Actuator * Actuator
     * Actuator * Actuator ==> Controller
 Controller (TarX, TarY, TarZ,VelX,VelY,VelZ,PosX,PosY,PosZ,Yaw,Pitchout,Rollout,Yawout,
     Throttleout, toix, toiy) ==
  (
    tarX := TarX;
    tarY := TarY;
    tarZ := TarZ;

  velX := VelX;
  velY := VelY;
  velZ := VelZ;
  posX := PosX;
  posY := PosY;
  posZ := PosZ;
  yaw := Yaw;

  pitchOut := Pitchout;
  rollOut := Rollout;
  yawOut := Yawout;
  throttleOut := Throttleout;

  toiX:= toix;
  toiY:= toiy;

  targetMap := new Map();
  obstacles := new Environment().getObstacles();
  survivors := new Survivor();

  );

 public GetThrottleOut: () ==> Actuator

 GetThrottleOut() == return throttleOut;

 public moveUAV: () ==> ()
 moveUAV() ==
 (
   setThrottleNew();
  setPitchNew();
  setRollNew();
  SetAltitude(2);
 );

 -- MAIN CONTROL LOOP
 public Step : () ==> ()
 Step() ==
 (
  targetMap.setTargetNew(tarX.GetValue(),tarY.GetValue());

  collisionAvoidance();
  if(survivors.isSurvivorFound(posX.GetValue(),posY.GetValue())) then(
   --send survivor location to control centre
    toiX.SetValue(posX.GetValue());
    toiY.SetValue(posY.GetValue());
    -- return base
    targetMap.survivorFound(0,0);
  );

  if(

   posX.GetValue() > (targetX - precision) and
   posX.GetValue() < (targetX + precision) and
   posY.GetValue() > (targetY - precision) and
   posY.GetValue() < (targetY + precision)
   ) then targetMap.currentTargetReached();


   moveUAV();
 );

 public collisionAvoidance: () ==> ()
 collisionAvoidance () == (
   if(collisionAvoidanceMode and [targetMap.targetX(), targetMap.targetY()] in set obstacles)
       then
  (
    --Maintain x movement
    targetX := targetMap.targetX();
    if(targetMap.targetY() > targetMap.nextTargetY()) then
    (
     --If the target area after the obstacle is towards negative y
     targetY := targetMap.targetY()-1;
    )else
    (
     --otherwise move to positive y to evade obstacle

     targetY := targetMap.targetY()+1;
    )
  )else
  (
    --no obstacle, so move to next location from the planned path
    targetX := targetMap.targetX();
    targetY := targetMap.targetY();
  );
 );

 public setThrottleNew: () ==> ()
 setThrottleNew () == (
   throttleIpart := throttleIpart + ((targetX-posZ.GetValue()) * 0.03);

    throttlePpart := (targetZ - posZ.GetValue()) * 1.2;
    if velZ.GetValue() > 0 then(
     throttleDpart := velZ.GetValue() * 0.5
    )
    else(
     throttleDpart := velZ.GetValue() * 1.0
    );
    GetThrottleOut().SetValue(throttleIpart + throttlePpart - throttleDpart)
 );

  public setPitchNew: () ==>  ()
 setPitchNew() == (
  --yIpart := yIpart + ((posY.GetValue()-targetY) * 0.00);

    yPpart := ( posY.GetValue()- targetY) * 0.28;
    yDpart := velY.GetValue() * -0.2;
    desiredPitch := yIpart + yPpart - yDpart;
    if desiredPitch > 0.35 then
     desiredPitch := 0.35;
    if desiredPitch < -0.35 then
     desiredPitch := -0.35;
   pitchOut.SetValue(desiredPitch)
 );

 public setRollNew: () ==>  ()
 setRollNew() == (
  --xIpart := xIpart + (targetX - (posX.GetValue()) * 0.00);
    xPpart := (targetX - posX.GetValue()) * 0.28;

    xDpart := velX.GetValue() * 0.2;
    desiredRoll := xIpart + xPpart - xDpart;
    if desiredRoll > 0.35 then
     desiredRoll := 0.35;
    if desiredRoll < -0.35 then

     desiredRoll := -0.35;
   rollOut.SetValue(desiredRoll)
 );


 public Upwards: real ==> ()
 Upwards(throttle) == (
    GetThrottleOut().SetValue(throttle);
 );

 public MoveX: real ==> ()
 MoveX(x) == (
  if x <> 0.0 then
   if x - posX.GetValue()  < x*0.01 then
    RollAng := CarlAngleTemp; --3.1;
  if x <> 0.0 then
   if x > posX.GetValue() then
    if velX.GetValue() < abs(posX.GetValue()-x)+0.01 then
     rollOut.SetValue(CarlAngleTemp)--(RollAng)

    else if posX.GetValue() > (x) or velX.GetValue() > (posX.GetValue()-x)  then
     rollOut.SetValue(-CarlAngleTemp)
    else rollOut.SetValue(CarlAngleTemp)
     else if velX.GetValue() > (x - posX.GetValue())+0.01 then
      rollOut.SetValue(-CarlAngleTemp)
  else if posX.GetValue() < (x) or velX.GetValue() < (x - posX.GetValue())  then
   rollOut.SetValue(CarlAngleTemp)
  else rollOut.SetValue(-CarlAngleTemp);
 );

 public MoveY: real ==> ()
 MoveY(y) == (
  if y <> 0.0 then
   if y - posY.GetValue() < y*0.01 then
    PitchAng := CarlAngleTemp;
  if y <> 0.0 then
   if y < posY.GetValue() then
    if velY.GetValue() > abs(posY.GetValue()-y)+0.01 then
     pitchOut.SetValue(CarlAngleTemp)

    else if posY.GetValue() > (y) or velY.GetValue() > (posY.GetValue()-y) then
     pitchOut.SetValue(-CarlAngleTemp)
    else pitchOut.SetValue(CarlAngleTemp)
     else if velY.GetValue() > (y - posY.GetValue())+0.01 then
      pitchOut.SetValue(-CarlAngleTemp)
   else if posY.GetValue() < (y) or velY.GetValue() < (y - posY.GetValue())  then
```

```
    pitchOut.SetValue(CarlAngleTemp)
   else pitchOut.SetValue(-CarlAngleTemp);
 );

 public SetAltitude: real ==> ()
 SetAltitude(altitude) ==(
  if altitude > posZ.GetValue() then
   if velZ.GetValue() < abs(posZ.GetValue()-altitude)+0.025 then
    Upwards(ThrottleHigh)
   else if posZ.GetValue() > (altitude) or velZ.GetValue() > (posZ.GetValue()-altitude) then
    Upwards(ThrottleLow)
   else Upwards(ThrottleHigh)
     else if velZ.GetValue() > (altitude - posZ.GetValue())+0.025 then
      Upwards(ThrottleLow)
   else if posZ.GetValue() < (altitude) or velZ.GetValue() < (altitude - posZ.GetValue())
   then Upwards(ThrottleHigh)
   else Upwards(ThrottleLow);
);


end Controller
```

4 Coor

```
class Coor

instance variables

-- local copy of the shared variable
protected x : real := 0;
protected y : real := 0;

operations

-- constructor for Command

public Coor: real * real ==> Coor
Coor(xAxis,yAxis) == (
 x := xAxis;
 y := yAxis;
);


public getX: () ==> real
getX() == return x;


public getY: () ==> real
getY() == return y;


public isEqual: Coor ==> bool
 isEqual(c) == (
  return c.getY() = y and c.getX() = x;
 );


public isWithinRange : real * real * real ==> bool
 isWithinRange(inputX,inputY,range) == (
  if(
   (inputX >= x-range) and
   (inputX <= x+range) and
   (inputY >= y-range) and
   (inputY <= y+range)
  ) then return true;
  return false;
 );

end Coor
```

5 Environment

```
class Environment

instance variables

-- local copy of the shared variable
protected obtacles : set of (seq of real);
protected numOfObstables : int;
protected x : real;
protected y : real;

operations

-- constructor for Command

public Environment: int * int ==> Environment
Environment(n,negativeOffset) == (
 --Create at least 1 obstacles
 numOfObstables := MATH`rand(n) + 1;

 obtacles := {};
 for i = 0 to numOfObstables do
 (
   x := MATH`rand(n) - negativeOffset;
   y := MATH`rand(n) - negativeOffset;
   obtacles := obtacles union {[ x, y ]};
 );

);


public getObstacles: () ==> set of (seq of real)
getObstacles() == return obtacles;

end Environment
```

6 HardwareInterface

```
class HardwareInterface

-- INPUT VARIABLES
instance variables

  -- @ interface: type = input, name="targetX";
  public targetX : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="targetY";
  public targetY : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="targetZ";
  public targetZ : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="posXIn";
  public posXIn : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="posYIn";
  public posYIn : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="posZIn";
  public posZIn : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="yawIn";
  public yawIn : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="velXIn";
  public velXIn : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="velYIn";
  public velYIn : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="velZIn";
  public velZIn : RealPort := new RealPort(0.0);

  -- @ interface: type = input, name="batteryChargeIn";
  public batteryChargeIn : RealPort := new RealPort(0.0);

-- OUTPUT VARIABLES
instance variables

  -- @ interface: type = output, name="pitchOut";
  public pitchOut : RealPort := new RealPort(0.0);

  -- @ interface: type = output, name="rollOut";
  public rollOut : RealPort := new RealPort(0.0);

  -- @ interface: type = output, name="yawOut";
  public yawOut : RealPort := new RealPort(0.0);

  -- @ interface: type = output, name="throttleOut";
  public throttleOut : RealPort := new RealPort(0.0);

  -- @ interface: type = output, name="toiXout";
  public toiXout : RealPort := new RealPort(0.0);

  -- @ interface: type = output, name="toiYout";
  public toiYout : RealPort := new RealPort(0.0);

end HardwareInterface
```

7 Map

```
class Map

instance variables
```

```
-- this value will be set through the co-simulation
protected port: RealPort;

-- local copy of the shared variable
private path: seq of Coor;
private hasTarget: bool;

--Variable for map planning
protected endX: real := 10;
protected endY: real := 10;

private startX: real := 5.0;
private startY: real := 5.0;

private currX: real;
private currY: real;

private xlength: real;
private scale: real := 2;

--Storing the x-axis path
private startToEnd : seq of real := [];
private endToStart : seq of real := [];

--Flag for x-axis flight direction
private xAxisFlag : bool;

--flag for when survivor are found
private idleMode: bool := false;

operations

-- constructor for Command

public Map: () ==> Map
Map() == (
  hasTarget := false;
  path := [new Coor(startX,startY)];
);

--public Map: real * real ==> Map
--Map(x,y) == (
--
--  path := [new Coor(x,y)];
--);


public setTargetNew: real * real ==> ()
setTargetNew(x,y) == (

 --If target is the same as current spot, no need to do path planning.
 if(x = targetX() and y = targetY()) then
 (
  hasTarget := true;
 );

 --Ignore input if x and y is 0 because of initial value from cosimulation
 --Might need to change this to be time based
 if( not hasTarget and (x <> 0 or y <> 0) and not idleMode) then (

  hasTarget := true;
  path := [];

  endY := y;
  endX := x;
  --https://playcode.io/463216

  currX := startX;
   currY := startY;

  xlength := abs(startX - endX);
  for i = 0 to (xlength - 1) by scale do
   (

    if(startX < endX) then (
      startToEnd := startToEnd ^ [startX + i + 1];
      endToStart := endToStart ^ [endX - i];

    )else(

      startToEnd := startToEnd ^ [startX - i - 1];
      endToStart := endToStart ^ [endX + i];
    );
   );

   xAxisFlag := true;
   if(currY < endY) then
   (
    if(currY = startY) then currY := currY + (scale/2);
    while(currY < endY) do
     (
    assignPath()
     );
   )else
   (
    if(currY = startY) then currY := currY - (scale/2);
    while(currY > endY) do
     (
    assignPath();
     );
   );

 );

);


 public assignPath : () ==> ()
  assignPath () ==
  (
   for i = 1 to len startToEnd do (
      if(xAxisFlag) then (
        currX := startToEnd(i);
        path := path ^ [new Coor(currX,currY)];
      ) else (
        currX := endToStart(i);
        path := path ^ [new Coor(currX,currY)];
      );
   );
    xAxisFlag := not xAxisFlag;
    currY := currY + (((endY - currY) / abs(endY - currY)) * scale);
  );


public getCurrentTarget: ()  ==> Coor
getCurrentTarget() == (
 return hd path;
);
```

```
public targetX: ()  ==> real
targetX() == (
 return (hd path).getX();
);


public targetY: ()  ==> real
targetY() == (
 return (hd path).getY();
);


public nextTargetX: ()  ==> real
nextTargetX() == (
 return (hd(tl path)).getX();
);


public nextTargetY: ()  ==> real
nextTargetY() == (
 return (hd(tl path)).getY();
);


public currentTargetReached: ()  ==> ()
currentTargetReached() == (
 --if there is more coordiantes on the plan, go to next one
 if(len path > 1) then
 (
  path := tl path;
 )
 else
 (
  hasTarget := false;
 )
);


public survivorFound: real * real  ==> ()
survivorFound(x,y) == (
 path := [new Coor(x,y)];
 idleMode := true;
);


end Map
```

8 Sensor

```
class Sensor

instance variables

-- this value will be set through the co-simulation
protected port: RealPort;

-- local copy of the shared variable
protected local_val: real := 0;
```

```
operations

-- constructor for Sensor

public Sensor: RealPort ==> Sensor
Sensor(p) == (
  port := p;
  local_val := p.getValue();
);

-- get sensor value

public GetValue: () ==> real
GetValue() == return local_val;

-- write local variable to shared variable

public Sync: () ==> ()
Sync() == local_val := port.getValue();

end Sensor
```

9 Survivor

```
class Survivor

instance variables

private survivors: set of Coor;
private range: real := 0.5;

operations

-- constructor for Command

public Survivor: () ==> Survivor
Survivor() == (
  survivors := {new Coor(10,8)}
);


public isSurvivorFound: real * real ==> bool
isSurvivorFound(currX,currY) == (
  dcl personFound : bool := false;
 for all person in set survivors do (
  if(person.isWithinRange(currX,currY,range)) then personFound:= true;
 );
 return personFound;
);

end Survivor
```

10 System

```
system System

instance variables

-- Hardware interface variable required by FMU Import/Export
public static hwi: HardwareInterface := new HardwareInterface();

-- controller
public static controller : [Controller] := nil;
public static mainthread: [Thread] := nil;

-- architecture
cpu : CPU := new CPU(<FP>, 1E6);


operations


public System : () ==> System
System () ==
 (--settings := World'settings;

  let tarX =  new Command(hwi.targetX),
      tarY =  new Command(hwi.targetY),
    tarZ =  new Command(hwi.targetZ),
    velX =  new Sensor(hwi.velXIn),
      velY =  new Sensor(hwi.velYIn),
    velZ =  new Sensor(hwi.velZIn),
    posX =  new Sensor(hwi.posXIn),
    posY =  new Sensor(hwi.posYIn),
    posZ =  new Sensor(hwi.posZIn),
    yaw =  new Sensor(hwi.yawIn),
    --batteryCharge =  new Sensor(hwi.batteryChargeIn),
    pitchOut = new Actuator(hwi.pitchOut),
    rollOut = new Actuator(hwi.rollOut),
    yawOut = new Actuator(hwi.yawOut),
    throttleOut = new Actuator(hwi.throttleOut),
    toiX = new Actuator(hwi.toiXout),
    toiY = new Actuator(hwi.toiYout)
  in
 (
  controller := new Controller(tarX, tarY, tarZ, velX , velY , velZ ,
            posX , posY , posZ ,
            yaw ,
            pitchOut , rollOut , yawOut , throttleOut,
            toiX, toiY);

   mainthread := new Thread(25, controller, tarX, tarY, tarZ, velX , velY , velZ ,
            posX , posY , posZ ,
            yaw ,
            pitchOut , rollOut , yawOut , throttleOut,
            toiX, toiY);
  );
  -- deploy the controller
  cpu.deploy(mainthread);
 );

end System
```

11 Thread

```
class Thread

instance variables

-- thread period
private period: nat := 1E9;

-- sensors and actuators, controller
-- commands
public static tarX: [Command] := nil;
public static tarY: [Command] := nil;
public static tarZ: [Command] := nil;

-- sensors
public static velX: [Sensor] := nil;
public static velY: [Sensor] := nil;
public static velZ: [Sensor] := nil;
public static posX: [Sensor] := nil;
public static posY: [Sensor] := nil;
public static posZ: [Sensor] := nil;
public static yaw: [Sensor] := nil;

-- actuators
public static pitchOut: [Actuator] := nil;
public static rollOut: [Actuator] := nil;
public static yawOut: [Actuator] := nil;
public static throttleOut: [Actuator] := nil;

public static toiX: [Actuator] := nil;
public static toiY: [Actuator] := nil;

private controller: [Controller] := nil;

operations

-- constructor for Thread

public Thread: nat * Controller * Command * Command * Command * Sensor * Sensor * Sensor * Sensor
     *
     Sensor * Sensor * Sensor *
     Actuator * Actuator * Actuator * Actuator
     * Actuator * Actuator  ==> Thread
Thread(f,c,TarX, TarY, TarZ,VelX,VelY,VelZ,PosX,PosY,PosZ,Yaw,Pitchout,Rollout,Yawout,Throttleout
    ,toix,toiy) == (
 period := frequency_to_period(f);
 tarX := TarX;
 tarY := TarY;
  tarZ := TarZ;
 velX := VelX;
 velY := VelY;
 velZ := VelZ;
 posX := PosX;
 posY := PosY;
 posZ := PosZ;
 yaw := Yaw;

 pitchOut := Pitchout;
 rollOut := Rollout;
 yawOut := Yawout;
 throttleOut := Throttleout;

 toiX:= toix;
 toiY:= toiy;
 controller := c
);

-- control loop

Step: () ==> ()
Step() == (
  tarX.Sync();
  tarY.Sync();
  tarZ.Sync();
  velX.Sync();
  velY.Sync();
  velZ.Sync();
  posX.Sync();
  posY.Sync();
  posZ.Sync();
  yaw.Sync();
  pitchOut.Sync();
  rollOut.Sync();
  yawOut.Sync();
  throttleOut.Sync();
  toiX.Sync();
  toiY.Sync();

  controller.Step();
);

-- run as a periodic thread
thread periodic(period, 0 ,0, 0)(Step);

functions

-- convert frequency to period in nanoseconds

private frequency_to_period: real -> nat
frequency_to_period(f) == floor 1E9/f

end Thread
```

12 World

```
class World
operations

-- run a simulation

public run : () ==> ()
run() ==
 (start(System`mainthread);
  block();
 );

-- wait for simulation to finish

block : () ==> ()
block() == skip;

sync per block => false;

values
```

```
public static q01Xstart : real = 0.0;
public static q01Ystart : real = 0.0;
public static q01Zstart : real = 0.0;
public static q01Thetastart : real = 0.0;

end World
```

D Appendix: VDM-RT primer

This appendix provide a primer to the VDM-RT language, which is used in the case studies to create the DE-models for co-simulation. The primer provided here is based on work by Fitzgerald et al. [271]. VDM-RT is an extended form of the VDM++ language by adding features for describing timed computations and threads. VDM++ is an extended form of the VDM-SL which adds classical object-orientation features.

In VDM-SL, system descriptions specify states in terms of typed variable constrained by logical predicates (in the form of data type invariants). Functionality is described either explicitly in applicative or imperative styles, or implicitly in terms of pre/post specifications. Descriptions of functionality yield proof obligations, e.g., for invariant preservation or satisfiability.

Basic Types

Basic types in VDM-RT includes Booleans (**`bool`**), Natural numbers (**`nat`**), Integers (**`int`**) and Real number (**`real`**). Table 1 shows the fundamental operators on these types. Throughout this summary, partial operators are denoted by $\overset{\sim}{\rightarrow}$. Note that the Boolean values **`true`** and **`false`** are not the same as numbers `1` and `0`.

Table 1: Operators on basic types; "T" stands for any type.

Operator	*Name*	*Type*
`not` `b`	Negation	**`bool`** $\rightarrow$ **`bool`**
`a` **`and`** `b`	Conjunction	**`bool`** * **`bool`** $\rightarrow$ **`bool`**
`a` **`or`** `b`	Disjunction	**`bool`** * **`bool`** $\rightarrow$ **`bool`**
`a => b`	Implication	**`bool`** * **`bool`** $\rightarrow$ **`bool`**
`a <=> b`	Biimplication	**`bool`** * **`bool`** $\rightarrow$ **`bool`**
`a = b`	Equality	T * T $\rightarrow$ **`bool`**
`a <> b`	Inequality	T * T $\rightarrow$ **`bool`**
`-x`	Unary minus	**`real`** $\rightarrow$ **`real`**
`abs` `x`	Absolute value	**`real`** $\rightarrow$ **`real`**
`floor` `x`	Floor	**`real`** $\rightarrow$ **`int`**
`x + y`	Addition	**`real`** * **`real`** $\rightarrow$ **`real`**
`x - y`	Difference	**`real`** * **`real`** $\rightarrow$ **`real`**
`x * y`	Product	**`real`** * **`real`** $\rightarrow$ **`real`**
`x / y`	Division	**`real`** * **`real`** $\overset{\sim}{\rightarrow}$ **`real`**
`x**y`	Power	**`real`** * **`real`** $\rightarrow$ **`real`**
`x` **`div`** `y`	Integer division	**`int`** * **`int`** $\overset{\sim}{\rightarrow}$ **`int`**
`x` **`rem`** `y`	Remainder	**`int`** * **`int`** $\overset{\sim}{\rightarrow}$ **`int`**
`x` **`mod`** `y`	Modulus	**`int`** * **`int`** $\overset{\sim}{\rightarrow}$ **`int`**
`x < y`	Less than	**`real`** * **`real`** $\rightarrow$ **`bool`**
`x > y`	Greater than	**`real`** * **`real`** $\rightarrow$ **`bool`**
`x <= y`	Less or equal	**`real`** * **`real`** $\rightarrow$ **`bool`**
`x >= y`	Greater or equal	**`real`** * **`real`** $\rightarrow$ **`bool`**

Collections: finite sets, maps, sequences

Sets are unordered finite collections of elements. They are denoted by braces and the empty set is written as {}. Table 2 shows operators on sets.

Sequences are finite ordered collections of elements. They are denoted by brackets and the empty sequence is denoted by `[]`. Table 3 shows operators on sequences.

Mappings are finite unordered collections of pairs of elements with a functional relationship between domain and range elements of specified types. Mappings are denoted by braces and with an arrow "`|->`" between domain and range values. The empty mapping is {`|->`}. Table 4 shows operators on mappings.

Structured Data: Records

A composite record type is defined in the manner indicated by the following example:

```
RecordT ::
```

Table 2: Summary of VDM++ set operators.

Operator	Name	Type
`e in set s1`	Membership	A * **set of** A → **bool**
`e not in set s1`	Not membership	A * **set of** A → **bool**
`s1 union s2`	Union	**set of** A * **set of** A → **set of** A
`s1 inter s2`	Intersection	**set of** A * **set of** A → **set of** A
`s1 \ s2`	Difference	**set of** A * **set of** A → **set of** A
`s1 subset s2`	Subset	**set of** A * **set of** A → **bool**
`s1 psubset s2`	Proper subset	**set of** A * **set of** A → **bool**
`s1 = s2`	Equality	**set of** A * **set of** A → **bool**
`s1 <> s2`	Inequality	**set of** A * **set of** A → **bool**
`card s1`	Cardinality	**set of** A → **nat**
`dunion ss`	Distributed union	**set of set of** A → **set of** A
`dinter ss`	Distributed intersection	**set of set of** A $\overset{\sim}{\to}$ **set of** A
`power s1`	Finite power set	**set of** A → **set of set of** A

Table 3: Summary of VDM++ sequence operators.

Operator	Name	Type
`hd l`	Head	**seq1 of** A → A
`tl l`	Tail	**seq1 of** A → **seq of** A
`len l`	Length	**seq of** A → **nat**
`elems l`	Elements	**seq of** A → **set of** A
`inds l`	Indexes	**seq of** A → **set of nat1**
`l1 ^ l2`	Concatenation	(**seq of** A) * (**seq of** A) → **seq of** A
`conc ll`	Distributed concatenation	**seq of seq of** A → **seq of** A
`l ++ m`	Sequence modification	**seq of** A * **map nat1 to** A $\overset{\sim}{\to}$ **seq of** A
`l(i)`	Sequence application	**seq of** A * **nat1** $\overset{\sim}{\to}$ A
`l1 = l2`	Equality	(**seq of** A) * (**seq of** A) → **bool**
`l1 <> l2`	Inequality	(**seq of** A) * (**seq of** A) → **bool**

```
  sel1 : T1
  sel2 : T2
```

This is a *record type* with two fields. All values belonging to the type contain a *tag* holding the name of the type, `RecordT`. The presence of a tag allows use of a constructor `mk_`*RecordT* for the tagged record type *RecordT*. Values belonging to this type could be written as follows:

```
mk_RecordT(t1,t2)
```

The field names `sel1` and `sel2` give rise to selector operators that are applied postfix using a "." notation.

Table 4: Summary of VDM++ mapping operators.

Operator	Name	Type
`dom m`	Domain	(**map** A **to** B) $\rightarrow$ **set of** A
`rng m`	Range	(**map** A **to** B) $\rightarrow$ **set of** B
`m1 munion m2`	Merge	(**map** A **to** B) * (**map** A **to** B) $\overset{\sim}{\rightarrow}$ **map** A **to** B
`m1 ++ m2`	Override	(**map** A **to** B) * (**map** A **to** B) $\rightarrow$ **map** A **to** B
`merge ms`	Distributed merge	**set of** (**map** A **to** B) $\overset{\sim}{\rightarrow}$ **map** A **to** B
`s <: m`	Domain restrict to	(**set of** A) * (**map** A **to** B) $\rightarrow$ **map** A **to** B
`s <-: m`	Domain restrict by	(**set of** A) * (**map** A **to** B) $\rightarrow$ **map** A **to** B
`m :> s`	Range restrict to	(**map** A **to** B) * (**set of** B) $\rightarrow$ **map** A **to** B
`m :-> s`	Range restrict by	(**map** A **to** B) * (**set of** B) $\rightarrow$ **map** A **to** B
`m(d)`	Map apply	(**map** A **to** B) * A $\overset{\sim}{\rightarrow}$ B
`m1 = m2`	Equality	(**map** A **to** B) * (**map** A **to** B) $\rightarrow$ `bool`
`m1 <> m2`	Inequality	(**map** A **to** B) * (**map** A **to** B) $\rightarrow$ `bool`
`inverse m`	Map inverse	**inmap** A **to** B $\rightarrow$ **inmap** B **to** A

Basic VDM-RT Expressions

Traditional if-then constructs can be used with an `elseif` keyword.

```
if predicate then Expression else Expression
```

Multiple alternatives can be denoted by a `cases` expression using pattern matching.

```
cases expression:
  pattern list 1 -> Expression 1,
  pattern list 2,
  pattern list 3 -> Expression 2,
  others                       -> Expression 3
end
```

Traversal over a set of values can be expressed as follows:

```
for all value in set setOfValues
do Expression
```

Local variable declarations take the following form:

```
dcl variable : type := Variable creation
```

Otherwise local temporary "constants" can be defined using a `let`-expression (again, with pattern matching):

```
let variable : type = Variable creation
in Expression
```

A **let**-be-such-that expression can be used to select an arbitrary value from a set satisfying a predicate, `pred`:

```
let variable in set setOfValues be st pred(variable)
in Expression
```

Comprehensions (Structure to Structure)

Comprehension expressions can be used for describing implicit ways for characterising the elements contained inside the collections (sets, sequences and mappings):

```
{element(var) | var in set setexpr & pred(var)}

[element(i) | i in set numsetexpr & pred(i)]

Typically:

[element(list(i)) | i in set inds list & pred(list(i))]

{dexpr(var) |-> rexpr(var) | var in set setexpr & pred(var)}
```

From Structure to Arbitrary Value Arbitrary element selection from a set can be expressed as:

```
Select: set of nat -> nat
Select(s) ==
  let e in set s
  in
    e
pre s <> {}
```

From Structure to Single Value Extraction of a single value from a structure is most naturally done using recursion:

```
SumSet: set of nat -> nat
SumSet(s) ==
  if s = {}
  then 0
  else let e in set s
       in
         e + SumSet(s\{e})
measure CardMeasure
```

The **measure** here is used to show the reader (and tools) how termination is ensured. Such a measure function must take the same parameters as the function and yield a natural number.

From Structure to single Boolean

For the special case where one needs to move from a set of values to a single boolean quantified expressions are most natural:

```
forall p in set setOfP & pred(p)

exists p in set setOfP & pred(p)

exists1 p in set setOfP & pred(p)
```

the **`exists1`** quantifier is true if and only if there is exactly one value from `setOfP` that satisfies the predicate `pred(p)`.

Threads and Synchronisation in VDM-RT

In a VDM-RT class it is possible to define a *thread*. These can be declared for example as:

```
thread
periodic(1000,10,200,0)(IncTime)
```

The four numeric parameters are, respectively, the period, jitter, delay, and offset. The final argument is the name of the operation to be called (in this example `IncTime`). Objects instantiated from a class with a *thread* part are called *active* objects. They nevertheless must be actively started using a **`start`** statement.

When multiple threads exist it may be necessary to synchronise them using *permission predicates*, for example:

```
per Push => length < maxsize;
per Pop => length > 0
```

Such permission predicates can also make use of history counters (**`#req`**, **`#act`** and **`#fin`**) for describing **`mutex`** constraints.

The System Class Concept in VDM-RT

A special **`system`** class is used to describe distributed systems which have static structures. This is done using `CPU`s connected by `BUS`ses. The constructor of the **`system`** class enables deployment of the statically declared top-level instances to the different `CPU`s.

Example of Classes

New instances of a class (called an object) can be produced using a **`new`** expression. Inside such a class it is possible to refer to itself using the **`self`** expression.

```
class Person

types

public String = seq of char;
public Sex = Male | Female
```

```
values

protected Name : seq of char = "Peter";

instance variables

public nationality : seq of char:="Danish";
comment             : String;
yearOfBirth         : int;
sex                 : Sex;
friends             : map String to Person

operations

public Person: int * Sex ==> Person
Person(pYear,pSex) ==
  (yearOfBirth := pYear;
   sex := pSex);

public GetAge : int ==> int
GetAge(year) == CalculateAge(year, yearOfBirth)
pre pre_CalculateYear(year, yearOfBirth);

functions

public CalculateAge : int * int -> int
CalculateAge (year,bornInYear) ==  year-bornInYear
pre year >= bornInYear
post RESULT + bornInYear = year;

Card: set of nat -> nat
Card(s) == card s;

thread
  while true do
    skip;

end Person
```

Subclasses can be defined as:

```
class Male is subclass of Person

...

end Male
```

And...

```
class Female is subclass of Person

...

end Female
```