Investigations in Ontology Development: The Use of Standard Office Tools and Arabic Script

Aisha Blfgeh

Submitted for the degree of Doctor of Philosophy in the School of Computing, Newcastle University
April 2023
Abstract

Ontologies are useful models for representing knowledge, especially in complex and dynamic domains such as biology or medicine. However, capturing such complex knowledge is essential to ensure the accuracy and reliability of any ontology. Therefore, building an ontology is a collaborative process which involves different communities who are required to work effectively in developing it; this includes domain specialists who understand the knowledge and ontology developers who identify the ontological representation of that specific knowledge. This collaboration requires a variety of instruments as each community has its own tooling and techniques.

In order to facilitate the interaction between domain specialists and ontologists, a document-centric workflow focused on the use of English and standard office tooling is designed. The aspiration here is to introduce a framework for building ontologies that allows both parties to use their own toolchains and for those toolchains to work together. Within biology and medicine, probably the most common tools for manipulating and sharing data are tools such as Microsoft Excel spreadsheets and Word documents. Unfortunately, these environments are far distant from the formal structured representation of the ontology development environments with which the ontologists work.

This research investigates approaches to applying standard office tools for developing and presenting ontologies, demonstrates how a narrative structure can be applied to an ontology, and finally, discusses the opportunities and challenges associated with internationalising an ontology development environment into Arabic. The goal is to bridge the gap, both in tooling and language, between ontological experts and biologists.
The results showed that it is possible to rethink ontology tooling and how various ontology representations are used in the development process. It has been demonstrated that office tooling, used naturally, can be integrated into the source code of an ontology, that it is possible to add narrative documentation to an ontology, and finally, to represent that narrative structure in Word, an application that most domain users are intimately familiar with. Furthermore, the user evaluation has shown that users can interact well with this form of representation. Taken together, these mechanisms suggest a plethora of future directions for developing ontologies and, more broadly, computational representations of complex knowledge domains.
ملخص الرسالة

الأطروحة هي نماذج مفيدة لتمثيل المعرفة خاصة في المجالات المعقدة والديناميكية مثل البيولوجيا أو الطب. ومع ذلك، تجميع وترتيب هذه المعرفة المعقدة ضروري لضمان الدقة والموثوقية لأي أطروحة. لذا، فإن عملية بناء الأطروحة هي عملية تعاونية تتضمن فئات مختلفة مطلوبة منها العمل بفعالية لتطوير هذه الأطروحة، هذا يشمل المتخصصين المجال الذين يفهمون المعرفة، ومطورين الأطروحة الذين يجدون ويفهمون التمثيل الأطروحجي لتلك المعرفة. ويطلب هذا التعاون أدوات وتقنيات مختلفة للعمل المشترك بين المجتمعات المختلفة فكل مجتمع الأدوات والتكنولوجيا الخاصة به.

فمن أجل تسهيل التفاعل بين المتخصصين في المجالات العلمي المراد تمثيله وعلماء الأطروحة، يقوم البحث الحالي بتصميم سير عمل يعتمد على المستندات ويرتكز حول استخدام اللغة العربية والأدوات المكتبية Microsoft Office. ويشمل إطار عمل لبناء الأطروحة بحيث يسمح لكلا الطرفين استخدام أدواتهم الخاصة.

داخل مجال البيولوجيا والطب، الأدوات الأكثر شيوعًا لمعالجة البيانات ومشاركتها هي أدوت مثل جداول بيانات Microsoft Excel، والمستندات النصية Word. وتسهّل النتائج هذه البيانات بعدة طرق، بعد أن تم تصميم المنظم الرسمي لبيانات التطوير الأطروحجي، والتي تعمل علماء الأطروحة بها.

هذه الرسالة، تبحث في الطرق المختلفة التي يمكن من استخدام الأدوات المكتبيةقياسية لتطوير وعرض الأطروحة. وكذلك، تبين كيف يمكن تطبيق الهيكل السردي على الأطروحة. وأخيرا، تناقش الاحتمالات والصعوبات المرتبطة بوجود أطروحة في اللغة العربية. وبهذه الطريقة، نأمل في سد الفجوة في مجال أدوات اللغة بين خبراء الأطروحة وعلماء الأحياء.

لقد أثبت أنه من الممكن إعادة التفكير في أدوات الأطروحة والدور الذي تمثله مختلف الأطروحة في عملية التطوير. لقد ظهر فعلا أن الأدوات المكتبية القياسية، المستخدمة بطبيعة الحال، يمكن دمجها في شفرة المصدر الخاصة بالأطروحة، وكذلك من الممكن إضافة وثائق سردية إلى الأطروحة، وعرض وتمثيل تلحك البنية السردية في Word.
البرنامج الذي تبين أن معظم مستخدمي المجال على دراية واسعة باستخدامه. وأظهر تقييم المستخدم الخاص بهذه الدراسة أنه يمكن للمستخدمين التفاعل بشكل جيد مع هذا النوع من التمثيل. إن هذه الآليات مجتمعة تقترح الكثير الطرق المختلفة التي يمكننا من خلالها تطوير الأنظمة ويشكل أعم التمثيل الحسابي لمجالات المعرفة المعقدة في المستقبل.
Declaration

I declare that this thesis is my own work, unless otherwise stated. No part of this thesis has previously been submitted for a degree or any other qualification at Newcastle University or any other institution.

Aisha Blfgeh

April 2023
Publications

Portions of the work within this thesis have been documented in the following publications:


²Available at [https://doi.org/10.1186/s13326-017-0159-4](https://doi.org/10.1186/s13326-017-0159-4)


\(^5\)Schedule available at https://www.keele.ac.uk/scm/newsandevents/events/ukontologynetwork/#ukon-2018-programme

\(^6\)Available at https://drive.google.com/file/d/1Cv46IELah_yCDf0LUke9Xi4qxxernW1L/view
Acknowledgements

In the name of God, the most Merciful, the most Compassionate.
I would like to express my sincere gratitude to my supervisor, Dr. Phillip Lord for his unlimited and dedicated support and guidance, with whom I would not have been able to complete this research.

Additionally, this endeavor would not have been possible without the generous assistance of my second supervisor, Dr. Ken Pierce, who provided invaluable feedback.

Moreover, I am very grateful to Dr. Jennifer Warrender, as I cannot forget her great effort and assistance with my thesis editing and formatting. Without her limitless support, I would not have been able to complete this thesis on time.

I would like to express my warm thanks to Dr. Catharien Hilkens for providing the Excel spreadsheet data which I used in my research, and the nice little meeting with her when she explained these data in the spreadsheet.

I would like to thank Dr. Nicola Lazzarini for helping me with the Italian translation during my research. Furthermore, I must express gratitude to Dr. Mohammad Halawani for the Cloud Ontology development and the nice conversations.

I am deeply grateful to all my friends, ICOS members, and colleagues for all their support and feedback which helped me to be productive and successful.

The School of Computing has to be remembered for all its guidance and enabling me to do my best. I will never forget the staff.

Most of all, I would like to express my extreme thanks to my dear husband, sons and daughter for all the support they have shown me throughout this journey. For my children,
thank you for your thoughts and consideration, and sorry for not being able to give you my full self while writing this thesis.

My parents, sisters and brothers, you are my greatest blessing. Huge gratitude for your prayers towards me! I am grateful for your existence and I cannot express my deep longing for our passionate gatherings.

Finally, I want to express a deep gratitude to myself for my enthusiasm and commitment towards the hard work, and the enjoyment of this PhD journey, with all the uplifting and stressful situations.
# Contents

1 Introduction ............................................ 1
   1.1 Ontologies in Biology and Medicine .................. 2
   1.2 Tools for Building Ontologies .......................... 3
   1.3 Research Problem ...................................... 4
      1.3.1 Research Questions .................................. 5
      1.3.2 Research Objectives ................................. 6
   1.4 Research Contributions .................................. 6
   1.5 Thesis Structure ......................................... 8

2 Background Research .................................... 11
   2.1 Conceptual Framework: Ontologies ....................... 13
      2.1.1 What is an Ontology? ................................. 13
         2.1.1.1 Definitions of Ontologies ....................... 15
      2.1.2 Why Ontologies? ..................................... 16
         2.1.2.1 Importance ....................................... 16
         2.1.2.2 Affordances ...................................... 17
         2.1.2.3 Challenges in Using and Developing ............... 21
      2.1.3 Examples of Ontologies .............................. 26
         2.1.3.1 Gene Ontology .................................. 26
2.1.3.2 ICD-11 .................................................. 30
2.1.3.3 SNOMED CT .......................................... 33

2.2 Technologies .............................................. 37
  2.2.1 Ontology Languages ................................. 38
    2.2.1.1 On the Use of Logic for Ontologies ......... 38
    2.2.1.2 The Web ........................................ 44
    2.2.1.3 Semantic Web .................................. 45
    2.2.1.4 OWL ............................................ 50
  2.2.2 Ontology Representation ............................ 56
    2.2.2.1 RDF/XML ....................................... 57
    2.2.2.2 Graphical User Interface (GUI) .............. 58
    2.2.2.3 Tawny-OWL ................................... 58

2.3 Ontology Engineering ..................................... 61
  2.3.1 Community Role in Ontology Development .......... 61
  2.3.2 Methodologies ....................................... 63
  2.3.3 Tools and Editors .................................... 64
    2.3.3.1 Ontolingua ..................................... 65
    2.3.3.2 OilEd ......................................... 65
    2.3.3.3 TopBraid Composer ............................. 65
    2.3.3.4 SWOOP 2.3 .................................... 66
    2.3.3.5 Protégé ....................................... 66
    2.3.3.6 Tawny-OWL ................................... 67

2.4 User Interaction with Ontologies ........................ 68
  2.4.1 User-centred Representation ........................ 68
    2.4.1.1 Representing Ontologies Outside of an Editor .. 68
    2.4.1.2 Ontology Documentation ........................ 70
    2.4.1.3 Literate Ontologies ............................ 72

2.5 Domain Specialization .................................... 74

2.6 Conclusions ............................................... 76
3 Technologies Employed in the Thesis

3.1 Introduction ................................................. 80
3.2 Clojure .................................................. 80
3.3 Tawny-OWL .............................................. 81
3.4 Emacs .................................................... 82
3.5 Literate Programming Approach ......................... 84
3.6 the UNICs of ORgaNizers (Org-mode) .................... 86
3.7 Pandoc .................................................... 87
3.8 Microsoft Excel and Microsoft Word ..................... 87
3.9 Summary ................................................ 88

4 Arabic Ontologies ............................................ 89

4.1 Introduction ................................................. 90
4.2 The Arabic language ....................................... 91
4.3 Identifier Types in Tawny-OWL ........................... 92
4.4 Tawny-OWL Polyglot Library .............................. 93
4.5 An Internationalized Source Code Environment ........ 94
4.6 Arabic Language Challenges ............................... 95
4.6.1 Text Direction ........................................ 97
4.6.2 Cursor Movement with Text .......................... 97
4.6.3 Integrated Development Environments (IDEs) ....... 99
4.6.4 Alphabetic Features .................................... 104
4.7 Conclusions ............................................... 105

5 Ontology Development Excel Workflow (ODEW) ......... 107

5.1 Introduction ................................................ 108
5.2 A Motivating Example ...................................... 110
5.2.1 The tolerogenic antigen-presenting cells (tolAPC) Catalog .... 111
5.2.2 Initial ODEW for the tolAPC Catalog ................ 112
5.3 The tolAPC Ontology .................................................. 114
  5.3.1 Building the tolAPC Ontology ............................... 114
  5.3.2 Modeling in the tolAPC Ontology ......................... 116
5.4 Generalization of ODEW .............................................. 119
  5.4.1 The Patterns in Bubo ......................................... 119
  5.4.2 Bubo Validation Using a Musical Spreadsheet .......... 123
5.5 Discussion ............................................................. 124
5.6 Conclusions ........................................................... 129

6 Ontologies’ Wordification ............................................... 131
  6.1 Introduction ....................................................... 132
  6.2 A Path to Using Microsoft Word in an Ontology Development ...... 134
    6.2.1 Generating Microsoft Word Documents .................. 135
    6.2.2 Implementation Use Case: Cloud Ontology ............. 136
      6.2.2.1 Lenticular Text and Org Files ....................... 137
      6.2.2.2 Pandoc and File Conversion ......................... 140
  6.3 Reflections on Ontologies Wordification .................... 141
  6.4 Summary .......................................................... 143

7 User Evaluation ........................................................ 145
  7.1 Introduction ..................................................... 146
  7.2 Arabic Version of Tawny-OWL ................................ 149
    7.2.1 Distribution Procedure .................................. 150
    7.2.2 Results of Arabic Questionnaire ....................... 150
    7.2.3 Characteristics of Respondents .......................... 150
    7.2.4 Marking Criteria ........................................... 152
    7.2.5 Accuracy and Errors in Answers .......................... 153
    7.2.6 System Usability Scale (SUS) ............................. 155
  7.3 Microsoft Excel .................................................. 157

- xvi -
7.4 Wordified Ontologies ................................................. 158
7.4.1 Stage 1: Empirical Test (Controlled) ......................... 159
  7.4.1.1 Recruitment Procedure .................................... 160
  7.4.1.2 Experiment Design and Protocol .......................... 160
7.4.2 Stage 2: Remote Test (Semi-Controlled) .......................... 161
  7.4.2.1 Recruitment Procedure .................................... 162
  7.4.2.2 Experiment Protocol ...................................... 162
7.5 Qualitative Data Analysis of Wordification Experiments .............. 163
7.6 Results of Wordification Experiments .................................. 163
  7.6.1 Stage 1 .......................................................... 163
    7.6.1.1 Reading the Ontology ................................... 163
    7.6.1.2 Finding Errors .......................................... 164
    7.6.1.3 Difficulties and Comments ............................... 165
  7.6.2 Stage 2 .......................................................... 166
    7.6.2.1 Experience Level of Participants ....................... 167
    7.6.2.2 Reading Ontologies ..................................... 167
    7.6.2.3 Understanding Ontologies and Finding Errors .......... 169
    7.6.2.4 Editing Ontologies ...................................... 172
    7.6.2.5 Overall Results ......................................... 173
7.7 Discussion and Conclusions ........................................... 175
7.8 Summary ..................................................................... 177

8 Discussion .................................................................... 179
  8.1 Introduction ............................................................ 180
  8.2 Development Approaches ............................................ 181
    8.2.1 Internationalization Environments .......................... 181
    8.2.2 Re-purposing Literate Programming .......................... 182
    8.2.3 Utilizing Office Tools and Ontology Building Environment 183
  8.3 Research Outcomes and Contributions ............................ 185

- xvii -
8.4 SWOT Analysis ......................................................... 190
  8.4.1 Analysis of Developing Ontology ............................ 191
    8.4.1.1 Strengths and Opportunities ............................ 191
    8.4.1.2 Weakness and Threats .................................. 192
  8.4.2 Analysis of Readability and Editability ..................... 193
    8.4.2.1 Strengths ................................................ 193
    8.4.2.2 Weakness and Opportunities ............................ 194
  8.4.3 Analysis of Evaluation ....................................... 194
    8.4.3.1 Strengths and Opportunities ............................ 194
    8.4.3.2 Weakness and Opportunities ............................ 195
    8.4.3.3 Threats ................................................ 195
8.5 Limitations and Restrictions ...................................... 196
  8.5.1 Internationalized Ontologies ................................. 196
  8.5.2 Document-centric Ontology Development ..................... 196
  8.5.3 Reusing Existing Ontology .................................. 197
  8.5.4 Evaluation Stage ............................................ 198
8.6 Future Work ....................................................... 198
8.7 Improving the Collaborative Ontology Engineering Landscape 200

A Evaluation:
  Supplementary Material ........................................... 201
    A.1 Stage 1 Scripts .............................................. 202
    A.2 Stage 2 Questions ............................................ 205
    A.3 Comments .................................................... 207
      1.3.1 Stage 1 Comments ....................................... 207
      1.3.2 Stage 2 Comments ....................................... 207

B Arabic Questionnaire .............................................. 211
    B.1 Arabic Questionnaire with English Translation and Answers 212
C Ontologies

C.1 Arabic Ontology .................................................. 220
C.2 Testing ontologies ................................................. 221
  3.2.1 Wordified Pizza Ontology ................................. 221
  3.2.2 Wordified Amino Acid Ontology ......................... 230
List of Figures

1.1 The interaction elements in the thesis. (The arrows represent all potential interactions, which will be discussed in detail in each chapter.) ................................................. 7

2.1 The flow of topics in Chapter 2. ................................................................................. 12
2.2 The evolution of knowledge representation. ............................................................... 14
2.3 Summary of ontology types according to the level of abstraction. ......................... 25
2.4 GO Elements. ............................................................................................................. 27
2.5 GO Overview. ............................................................................................................ 28
2.6 The structure of GO as a graph showing the relations between terms. ................. 29
2.7 ICD-11 components. ................................................................................................. 31
2.8 ICD-11 Browser. ........................................................................................................ 32
2.9 ICD-11 Coding Tool. ................................................................................................. 33
2.10 A high-level SNOMED Logical model. ..................................................................... 35
2.11 SNOMED Browser. ................................................................................................... 36
2.12 SNOMED Search. ...................................................................................................... 36
2.14 Early web model by Sir Tim Berners-Lee (Barners-Lee et al., 2001). .................... 45
2.15 Example of a Resource Description Framework (RDF) Graph model. ................. 48
2.16 Example of a RDF Graph model with added RDF Schema (RDFS) (by dotted lines). ....................................................................................................................... 50
2.17 Expressiveness levels of Web Ontology Language (OWL).......................... 53
2.18 A class Person and its equivalent class Human and disjoint subclasses, Man
and Woman................................................................. 54
2.19 An individual Chloe is a type of Person class since it is a Woman.............. 56
2.20 GUI screen of Protégé...................................................... 59
2.21 The role model for ontology users........................................... 62
2.22 Ontology development stages with user roles in each stage...................... 63
2.23 Ontology engineering methodologies........................................... 64
2.24 An OWLDoc representation inside Protégé environment.......................... 71
2.25 Two examples of literate ontologies............................................. 74
2.26 Intermediate representation and Cucumber text................................... 75
3.1 Emacs built-in documentation Screen.............................................. 83
4.1 The interactions in ontology development using Arabic language.............. 91
4.2 Arabic version of the Family ontology in Protégé.................................. 93
4.3 Polyglot library in Tawny-OWL............................................... 94
4.4 Adding a new Arabic class to the Family Ontology in Protégé.................... 95
4.5 Multilingual Pizza ontology..................................................... 96
4.6 Emacs display of Arabic text in two directions.................................... 97
4.7 Cursor movement with mixed text of Arabic and English characters.......... 98
4.8 Cursor movement in text inside Tawny-OWL environment........................ 99
4.9 Eclipse screen shot of the Arabic Pizza ontology using the graphical right-
to-left layout............................................................... 100
4.10 Eclipse screen shot of the Arabic Pizza ontology................................ 101
4.11 IntelliJ screen of Arabic Pizza ontology........................................ 102
4.12 Atom screen of Arabic Pizza ontology........................................... 103
4.13 NightCode screen of Arabic Pizza ontology...................................... 104
5.1 The interactions in ontology development using Excel spreadsheet............ 110
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.6 Experience level of the participants.</td>
<td>168</td>
</tr>
<tr>
<td>7.7 The ease of reading the ontology results.</td>
<td>169</td>
</tr>
<tr>
<td>7.8 Understanding the construction flow of the ontology results.</td>
<td>169</td>
</tr>
<tr>
<td>7.9 Results of finding errors in the ontology.</td>
<td>170</td>
</tr>
<tr>
<td>7.10 An overview of the correct/incorrect answers.</td>
<td>172</td>
</tr>
<tr>
<td>7.11 Results of editing the ontology.</td>
<td>173</td>
</tr>
<tr>
<td>7.12 The clarity of ontology structure results.</td>
<td>174</td>
</tr>
<tr>
<td>8.1 Contribution fields of research in the area of ontology development.</td>
<td>181</td>
</tr>
<tr>
<td>8.2 Document-Centric workflow.</td>
<td>184</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Examples of ontologies........................................26
2.2 Biological ontologies comparison..............................38
2.3 Summary of languages used for ontologies......................39
2.4 RDFS vocabulary (Adapted from Brickley et al. (2014)).......49
2.5 RDF and OWL syntax..............................................53
2.6 OWL constructors and the equivalents in DLs and FOL........54
2.7 Ontology editing tools.............................................68

3.1 Tools and the use of comments for different purposes........86

4.1 Comparison of IDEs in dealing with Arabic script...........104
4.2 Tawny-OWL commands and their equivalent in Arabic.......105

5.1 Statistics of the tolAPC catalog and the tolAPC ontology......113
5.2 Amount of data extracted from the Microsoft Excel spreadsheet source..123
5.3 Comparison of ODEW with other tools..........................128

7.1 Criteria for awarding marks....................................153
7.2 The percentage of correct answers for the open-ended questions in the survey.154
7.3 SUS statements with calculation score of each statement........156
7.4 Error finding performance in both versions of the ontology during the controlled experiment........165
7.5 The percentages of correct answers in both wordified ontology and Protégé.171
7.6 Overall positive percentage of user preference between the wordified version and Protégé. 175

8.1 The SWOT Analysis areas for this research. 191
Acronyms

**A-FACTT**  Action to Focus and Accelerate Cell-based Tolerance-inducing Therapies

**BDD**  Behaviour Driven Development

**BFO**  Basic Formal Ontology

**CLR**  Common Language Runtime

**CSV**  Comma Separated Values

**DAML**  DARPA Agent Markup Language

**DOLCE**  Descriptive Ontology for Linguistic and Cognitive Engineering

**DOSDP**  Dead Simple OWL Design Patterns

**FMA**  Foundational Model of Anatomy

**GNU**  GNU’s Not Unix

**GO**  The Gene Ontology

**GUI**  Graphical User Interface
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCI</td>
<td>Human Computer Interaction</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>iCAT</td>
<td>Collaborative Authoring Tool</td>
</tr>
<tr>
<td>ICD-10</td>
<td>International Classification of Disease version 10</td>
</tr>
<tr>
<td>ICD-11</td>
<td>International Classification of Disease version 11</td>
</tr>
<tr>
<td>ICOS</td>
<td>Interdisciplinary Computing and Complex BioSystems</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IR</td>
<td>Infra-Red</td>
</tr>
<tr>
<td>IRI</td>
<td>Internationalized Resource Identifier</td>
</tr>
<tr>
<td>JARO</td>
<td>Juvenile Rheumatoid Arthritis Ontology</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>LODE</td>
<td>Live OWL Documentation Environment</td>
</tr>
<tr>
<td>NCBI</td>
<td>National Centre for Biotechnology Information</td>
</tr>
<tr>
<td>OBO</td>
<td>Open Biomedical Ontologies</td>
</tr>
<tr>
<td>ODEW</td>
<td>Ontology Development Excel Workflow</td>
</tr>
</tbody>
</table>
OIL  Ontology Interchange Language

OPPL  Ontology Pre-Processor Language

Org-mode  the UNICs of ORgaNizers

OWL  Web Ontology Language

PDF  Portable Document Format

RDF  Resource Description Framework

RDFS  RDF Schema

REPL  Read-Eval-Print Loop

RIF  The Rule Interchange Format

ROBOT  ROBOT is an OBO Tool

SIO  Semanticscience Integrated Ontology

SNOMED  Systematized Nomenclature of Medicine

SWO  The Software Ontology

tolAPC  tolerogenic antigen-presenting cells

URI  Uniform Resource Identifier

URL  Uniform Resource Locator
UTF-8 Unicode Transformation Format which uses 8-bit blocks to represent a character

WIDOCO WIzard for DOCumenting Ontologies

XML eXtensible Markup Language
Chapter 1

Introduction

Contents

1.1 Ontologies in Biology and Medicine ........................................... 2
1.2 Tools for Building Ontologies .................................................. 3
1.3 Research Problem ................................................................. 4
   1.3.1 Research Questions ......................................................... 5
   1.3.2 Research Objectives ....................................................... 6
1.4 Research Contributions ............................................................. 6
1.5 Thesis Structure ................................................................. 8
Abstract

Knowledge representation is an essential way to understand and manipulate information, especially when this information is complex knowledge. A wide range of methods represent data computationally and conceptually, such as graphs, dictionaries, and ontologies. In this research, the main focus is on ontologies and particularly, ontologies that represent biomedical knowledge.

1.1 Ontologies in Biology and Medicine

An ontology is “a kind of controlled vocabulary of well defined terms with specified relationships between those terms, capable of interpretation by both humans and computers” (NCBO, 2015). Ontologies are increasingly used to facilitate the management of knowledge and the integration of information as in the Semantic Web (Bernus et al., 2007). In addition, they have been used extensively to describe many parts of biology, where biological data is not only heterogeneous, but requires special knowledge to deal with, and can be large (Stevens et al., 2000b). As ontologies are useful for representing complex and potentially changeable knowledge, they have two key features which make their usage attractive: first, they can provide a mechanism for standardizing and sharing the terms used in descriptions; and second, they provide computationally amenable semantics for these descriptions, making it possible to draw conclusions that are not explicitly stated. Furthermore, they provide a query mechanism for answering questions and retrieving information about the knowledge represented. They are widely used in biomedicine with examples such as The Gene Ontology (GO) (Ashburner et al., 2000), International Classification of Disease version 10 (ICD-10) (WHO (ICD-11), 2016), and Systematized Nomenclature of Medicine (SNOMED) (IHTSDO, 2016) being the best known.

However, building an ontology is a challenging task (Lord, 2013b). Ontologies often use languages with a complex underlying formalism (such as Web Ontology Language (OWL)\footnote{https://www.w3.org/TR/2004/REC-owl-semantics-20040210/}, for instance), especially when modeling complex domain areas such as biology or medicine. Normally, ontologies are built collaboratively between domain specialists who know the domain, and ontology developers who know how to structure and represent the knowledge.
They must work together to construct a robust and accurate ontology. However, any form of multi-disciplinary collaboration is difficult. Therefore, community involvement is crucial during the process of building ontologies (Mankovskii et al., 2009) through meetings, focus groups, and the like. For example, in the case of Gene Ontology, the biological community’s involvement is important for its successful uptake (Bada et al., 2004). In addition, Bult et al. (2011) stated that the development of Protein Ontology requires a wider range of involvement to include other users and developers of the associated ontologies, such as GO, to ensure the consistent architecture of the ontology.

1.2 Tools for Building Ontologies

There are various tools for constructing and developing ontologies with a variety of user interfaces and environments. The most popular tool in academia is Protégé (Horridge et al., 2011b), which is an open-source tool that provides a user interface to develop and construct ontologies of any domain. It has been widely used for developing ontologies due to the variety of plug-ins and frameworks it provides (N. F. Noy et al., 2003). Protégé provides an easy interface for the editing, visualization, and validation, as well as a useful tool for managing large ontologies\(^2\).

Conversely, Tawny-OWL is a textual interface for developing ontologies in a fully programmatic manner (Warrender, 2015a). This provides a convenient and readable syntax that can be edited directly using an Integrated Development Environment (IDE) or text editor; in this style of ontology development, the ontologist ceases to manipulate an OWL representation directly and instead, develops the ontology as a programmatic source code (Lord, 2013b).

These tools are used and maintained by ontologists and knowledge engineers who are familiar with the formal representation of knowledge, and have the skills to structure and maintain the representation computationally.

\(^2\)http://protege.stanford.edu/
1.3 Research Problem

Despite the computational advantages of ontologies, the oldest and the most common form of description in biology is free text or a semi-structured representation using a standardized fill-in form. These representations have numerous advantages compared to ontologies: they are richly expressive, widely supported by tooling, and while the form of language used in science (‘Bad English’ (Wood et al., 2001)) may not be easy to use, understand, or learn, it is widely taught, and most scientists are familiar with it. Similarly, most biologists are familiar with the tools used for producing free text and forms; either a word-processor document or a spreadsheet. Tools for producing this form of knowledge have broad functionality, both in the application and cloud-delivered form. Moreover, it supports highly collaborative development, such as Microsoft Office, Google Drive, and One Drive. In contrast, environments for creating ontology are designed to produce a formal representation of a domain, either using software for a GUI, such as Protégé, or a programmatic textual environment, such as Tawny-OWL; they are rarely familiar to new users and do not necessarily support collaboration.

Consequently, there are several challenges associated with ontology building. The process often touches on different subject areas:

- A knowledge of the domain.
- An understanding of the ontology formalism.
- Tools and software in use.
- Sometimes, an understanding of the philosophical background is needed.

Practically, it is very rare that an single person can complete an ontology, since they are unlikely to combine all of these skills. Therefore, individuals with these diverse abilities and multiple resources must work together. Face-to-face meetings are one effective option for communication, but they can be costly and time-consuming for teams that are not
co-located. Remote collaboration is also an option, but one challenge is that domain specialists represent and share their data in various formalisms.

Taking into account the support for multiple languages in the textual interface Tawny-OWL, and the adoption of internationalization technology where programs with a graphical user interface can now support different languages, this allows users to read and write the ontology in their own language, which is obviously will improve ontology readability and usability. Therefore, it is suggested that non-English options be made more accessible so that ontologists can communicate and complete all stages of the development process in their own language.

Nevertheless, the most common ‘formalism’ is the Office file, which can take a word processor document or a spreadsheet to organize the knowledge. Because of this difference in tools, this research proposed a solution to bridge the gap between the two groups while building an ontology. By providing a practical method for both parties to read or manipulate the ontology, that solution would be helpful in facilitating communication between domain experts and ontologists.

1.3.1 Research Questions

RQ1 Is it attainable to internationalize the environment of Tawny-OWL and present ontologies in the Arabic language?

RQ2 What would be the process of using Microsoft Office Excel to build an ontology?

RQ3 Is it possible to represent literate ontologies as Microsoft Word documents?

RQ4 How can this Microsoft Office interaction encourage domain specialists and ontology developers to work collaboratively?

RQ5 As a framework, what are the possible pros and cons of iterative enhancement of ontologies to different groups of people involved in the ontology development process?

Additionally, the following study objectives are established to aid in achieving and resolving those research questions.
1.3.2 Research Objectives

**RO1** To represent ontologies in other natural languages (Arabic in this research). This objective is aimed to address RQ1.

**RO2** To investigate the transformation of existing semi-structured knowledge, such as Excel, into an ontology. This is to answer RQ2.

**RO3** To build state-of-art ontologies that are fully literate, such as Pizza Ontology and Cloud Ontology, as exemplars. This is to address RQ2 and RQ3 and to help with RQ4 and RQ5.

**RO4** To investigate tooling for the transformation of literate ontology forms into readable and editable Word documents. This corresponds with RQ3.

**RO5** To use domain specialists and ontologies to validate the outcome of the proposed methods. This corresponds with RQ5 and RQ4.

This study investigates a document-centric workflow using Arabic Language and standard office tools, which helps bridge the gap between domain specialists and ontologists in ontology development.

1.4 Research Contributions

The primary goal here is to incorporate tools used by users into the ontology development pipeline.

The author of this thesis is aiming to close some tooling gaps. Methodologies for communication between the two groups, who frequently utilize different technologies, are suggested using these tools. Each chapter will elaborate on the interaction strategies utilizing the most popular ontology development tools, as shown in the Figure 1.1. This figure depicts the two main players in this thesis who interact during the development of ontologies. The communities are shown as either domain experts or ontology experts. The tools, how-
ever, are showed as either ontology tools or Office programs, particularly Excel and Word. Finally, the Arabic language is adopted as a communication medium as well.

Figure 1.1: The interaction elements in the thesis. (The arrows represent all potential interactions, which will be discussed in detail in each chapter.)

As such, the following research contributions are listed, and the source code for the work can be found in author’s GitHub repository:

1) Background Research: An extensive exploration of the literature is presented to show and discuss ontologies’ importance through time. Additionally, ontology engineering is investigated with user-centric interests and explores the tooling behind the development process (see Chapter 2).

2) Internationalization of Tawny-OWL: An Arabic version of Tawny-OWL environment is developed and produced specifically for Arabic users. This Arabic version can be used to build ontologies using the Arabic alphabet commands (see Chapter 4).

3) Ontology Development method using Excel spreadsheets: An Ontology Development Excel Workflow (ODEW) has been developed in this research, which incorporates a Microsoft Excel spreadsheet into the development process. The main distinguishing
feature in this research is the flexibility of modifying the Excel spreadsheet without the necessity of recreating a new document (see Chapter 5).

4) Wordification of ontologies: A novel representation of ontologies has been introduced by generating a Microsoft Word document from the source code, by which Tawny-OWL statements can be hidden, if required. This allows domain specialists to read and modify the text in the Microsoft Word document without editing ontology statements. Furthermore, the Microsoft Word documents can be shared using any cloud facilities such as Dropbox or Google docs and version control systems such as GitHub (see Chapter 6).

5) Generating ontology exemplars using thesis approaches: Several ontologies have been developed by implementing the proposed approaches—for instance, the tolAPC and Amino Acids ontologies. Additionally, the Arabic Ontology of Pizza and the Ontology of Family is among them. Some of them are available in the GitHub repository, while some are included in the Appendix (see Chapters 5, 6, and C).

6) Appraisal of thesis approaches: Users were allowed to interact with Wordified ontologies and try out Arabic commands of Tawny-OWL during the evaluation process. They performed various tasks based on the evaluation criteria. The results of user behavior were provided, as well as their feedback. ODEW was also validated by creating ontologies in Excel spreadsheets from various sources (see Chapter 7). SWOT (Strengths, Weaknesses, Opportunities, and Threats) analysis is provided as a mechanism to evaluate the proposed methodologies.

1.5 Thesis Structure

The remaining chapters of the thesis are organized as follows:

- Chapter 2 introduces ontologies and their history in knowledge representation and describes ontology types and their applications. Furthermore, it addresses existing
information of ontology engineering, focusing on collaboratively designed ontologies and the implementation of standard office documents such as spreadsheets and text documents.

- Chapter 3 describes the crucial tools used to develop the ontologies in this thesis, which is the Tawny-OWL. It also addresses other relevant tools and technologies used in this research, especially Clojure, Emacs, Org-mode file, Pandoc, and Microsoft Word/Microsoft Excel.

- Chapter 4 describes how the textual environment of Tawny-OWL can completely support internalization in developing ontologies. It further discusses the Arabic language with its distinctive features and grammar, plus the limitations of current ontology engineering tooling that can be used to build an ontology employing one’s native language.

- Chapter 5 investigates how to incorporate Microsoft Excel spreadsheets into the development process of an ontology. By generating the tolerogenic antigen-presenting cells (tolAPC) Ontology, it demonstrates one aspect of the document-centric approach using the tolAPC catalog and predefined Tawny-OWL patterns. This approach is also tested by creating a musical ontology using a free musical spreadsheet resource.

- Chapter 6 investigates how Microsoft Word documents can be implemented into the ontology creation process and fostering further collaboration among field specialists and ontologists. This demonstrates another aspect of the document-centric approach by generating two Wordified ontologies: the Pizza Wordified Ontology and the Amino Acid Wordified Ontology.

- Chapter 7 shows the user evaluation results of the Arabic version of Tawny-OWL and the document-centric approach through the action of construction and beta testing. Furthermore, it analyzes how the beta testing was conducted in two stages (i.e., using
controlled and semi-controlled environments) and discusses how it compares against its competitors.

- Chapter 8 summarizes the research presented in this study and includes the thesis conclusions, emphasizing their relevance to the research questions themselves (see Section 1.3.1). This chapter also goes into detail on SWOT analysis. Eventually, it addresses the limitations, potential improvements, and the contribution it makes to the ontology community.
Chapter 2

Background Research

Contents

2.1 Conceptual Framework: Ontologies .......................... 13
  2.1.1 What is an Ontology? .................................. 13
  2.1.2 Why Ontologies? .................................. 16
  2.1.3 Examples of Ontologies ............................. 26
2.2 Technologies ........................................... 37
  2.2.1 Ontology Languages ................................. 38
  2.2.2 Ontology Representation ............................. 56
2.3 Ontology Engineering .................................... 61
  2.3.1 Community Role in Ontology Development .......... 61
  2.3.2 Methodologies .................................. 63
  2.3.3 Tools and Editors .................................. 64
2.4 User Interaction with Ontologies ......................... 68
  2.4.1 User-centred Representation ......................... 68
2.5 Domain Specialization ................................ 74
2.6 Conclusions ........................................... 76
Abstract

Ontologies are artifacts for knowledge representation that have been widely used in different sciences. There are other types of knowledge representation models, including trees, graphs, and thesauri. Each one of these models has its use, features and is applicable in many fields. This chapter describes what ontologies are and their purposes in the scientific field. In addition, it shows the various methods for developing ontologies and describes technologies and editors to develop the ontologies. It touches on the mechanisms used to build ontologies collaboratively and shows the existing approaches to represent an ontology in a more human-readable form to help in the development process. However, the scope of this thesis covers ontologies, specifically OWL ontologies.

In this chapter, an overview of the ontologies, their development technologies, and tools are described briefly, including the human aspect, which covers users/communities in the development pipeline.

A sketch of themes in this chapter is illustrated in Figure 2.1.

Figure 2.1: The flow of topics in Chapter 2.
There are four themes. Firstly, a conceptual framework about ontologies, the description of the ontologies, and their importance in the science field. Secondly, an exploration of various technologies related to ontologies and how they can be represented and displayed is offered. Thirdly, a review of ontology engineering is described with various representations of ontologies outside the editors. Finally, user interaction with ontologies is presented with a model derived from the different roles of users in the development process. As the main topic of this thesis is ontology development, the focus is on how ontologies are developed, both in terms of technologies, and also social aspects. Moreover, how people with different skill sets use this technology to interact during the development lifecycle is under consideration.

2.1 Conceptual Framework: Ontologies

In this section, the concept and motives of using ontologies are considered. First, I will explore the definitions of ontologies, describe the technologies behind modern ontologies, and how they evolve. Second, the uses of ontologies that have been exposed in the biological and medical fields will be reviewed, showing the advantages of their usage. Finally, a few examples of the ontologies will be described, showing their different applications in the biomedical area.

2.1.1 What is an Ontology?

It is imperative to start by describing what an ontology actually is. This is somewhat complicated by the long history of the word, which was originally borrowed from philosophy. I will consider this history briefly, describing the development of critical features and antecedents that give rise to modern ontologies. I complete with a specific definition for this thesis which is rather more constrained than the history would suggest. In the seventeenth century, the term 'ontology' started from philosophy, referring to the science of existence (Ferraris, 2005). Eventually, with the revolution of information, the term has evolved in Artificial Intelligence to describe the formal semantic representation
Chapter 2

2.1 Conceptual Framework: Ontologies

Figure 2.2: The evolution of knowledge representation.

of knowledge. An early forerunner of ontology, shown in Figure 2.2, are dictionaries and thesauri, which are used to define terms and organize different meanings of words to unify the understanding of these terms and control the sharing of information. Those tend to focus on high complex meaning, are distinct from databases where information is rigidly structured into tables (with relations between them), managing the essential data of systems. These allow for highly efficient storage and querying of substantial quantities of data.

Another concept frequently used to assist in developing database schemes and associated software is conceptual data modeling, which UML (Unified Modelling Language) perhaps exemplifies. UML provides a standardized representation for entities in a domain, their representation in software, and the way these software components interact.

In a development largely separate from these came the web in the 1980’s. At the same time, this was initially focused on representing documents, both in text and image. It also contained a mechanism for hyper-linking documents together, in a single worldwide identifier space, identified using the URL (Uniform Resource Locator), which subsequently became URI (Uniform Resource Identifier) and then, IRIs (Internationalized Resource Identifier). This has subsequently expanded to cover something more data-centric, in
the form of the semantic web and later, linked data. Here, XML (Extensible Markup Language) and RDF (Resource Description Framework) are used for representing data and again, linked with IRIs. We will explain more about the web and its technologies later in Section 2.2.1.2.

Going back to the history, ontologies evolved as a combination of dictionaries, thesauri, databases, and linking data. They have features from each of these computational data models. Thus, in any ontology, terms are defined (dictionary) and annotated with labels to describe any synonyms (thesaurus), then linked with other terms to describe the relationships (database). There is also a hierarchy of these terms (taxonomy) to show the level of represented knowledge.

I will discuss the development of ontologies and their representations with associated technologies in detail, later in this chapter. For the moment, I simply say that there have been many representations, and I concentrate on the Web Ontology Language (OWL) representation specifically. However, as the main focus of this thesis is user interactions with ontology, I will look first at the definitions and the expansion of ontology applications.

### 2.1.1.1 Definitions of Ontologies

There is no universal agreement of the meaning for ontology in computing. The most common definition by Gruber (1993) has been dominating the literature in the last few decades, which describes ontology as the “explicit specification of a conceptualization”. This definition is relatively brief and ambiguous as it offers little explanation of the term. In a broad sense, it is a form of knowledge representation that enables users to describe and share domain knowledge precisely. Alternatively, it can be considered an organization of information using a logic-based language to describe terminologies and concepts, with their associated relationships in a particular domain. This definition suggests that ontologies are logical representations of domain concepts and their relationships.

It can also be defined as “the knowledge body which describes domains usually knowledge domains of common sense” (Fardoos et al., 2014). Within this definition, an ontology is a shareable artifact of specific domain knowledge that is understandable within that domain.
Chapter 2 2.1 Conceptual Framework: Ontologies

From these definitions, an ontology can be viewed as mainly a mechanism for representing knowledge, computationally, that can be reasoned over and queried off. I will talk about this in more detail later in Section 2.1.2.1.

Due to the variety of meanings for ontologies being either inconclusive, vague, or simply fitting a particular reason only, it is not easy to decide on a single definition. However, a more constrained definition will be considered, that is relevant to this thesis which will be described in Section 2.2.1.4. Within the subsequent section, reasons behind the expanding use of ontologies, their advantages, and different applications will be discussed.

2.1.2 Why Ontologies?

With the enormous amount of data published on the internet, the development of technologies, and data representation in various formats, users can access information from any domain through numerous systems (e.g., database systems, webpages). Therefore, managing, sharing, and exchanging information becomes crucial as this information can be used for learning, decision-making, or any other purpose, serving a diverse array of users. As a semantic solution, ontologies provide a mechanism to represent the data in a meaningful scheme and manage the knowledge interoperability among different systems. In other words, ontologies enable computers to understand the data according to agreed vocabularies and descriptions by the domain community.

A vital benefit of an ontology is that it provides a common language for users in a domain, as they all need to refer to the same information. In the next section, I will discuss the importance of using ontologies to represent, share, and manage knowledge.

2.1.2.1 Importance

Nowadays, we have access to various data on the web with an ever-increasing amount of information. This makes the management of digital data crucial. The management entails, but is not limited to, comprehending data to facilitate its correct linking, by relevance. Additionally, it is the formal representation of information that lends utility and value to this digital knowledge.
In fact, ontologies are a formal specification of knowledge; they have their objects, relations, assertions, and constraints, which help manage and control the information. For instance, a shared understanding of domain concepts can be communicated and integrated between users and machines, ensuring that appropriate services are provided through web applications (Hepp et al., 2007). Moreover, they are successful at representing domain information knowledge, integrating data from diverse sources, and supporting various semantic applications (Middleton et al., 2009; Salatino et al., 2020). Unlike other forms of knowledge representation, such as dictionaries and databases, ontologies prove their efficiency in various areas, such as semantic web and biotechnology. For data management, ontologies are essential since they maintain the common understanding of the knowledge, which facilitates interoperability and the queries of data (Ontotext, 2020).

In that regard, Martin (2010) claims that the importance of ontology in the philosophy of ‘creative’ science helps to understand the science itself. That is, “we must take care not to reduce science to its ability to inspire and acknowledge, and science can inspire and acknowledge if we take ontology seriously”.

By the use of ontologies, information can semantically be processed by computers, and this helps humans and machines to communicate by associating machine-processable content with human-consensual terminology (Hepp et al., 2007). As ontologies provide a formal semantics of data, they have benefited the revolution of data representations, as will be described next by showing various ontology uses in the field of Biology. Though it will be explained in further detail, the biology field was selected because ontologies are widely implemented and used to organize and share biological knowledge, which will be shown in examples later in Section 2.1.2.5.

2.1.2.2 Affordances

In the biology domain, ontologies are used for a variety of different purposes. Some of these uses are described in Stevens et al. (2009) and given below.

Reference ontology: Having a domain described as entities or classes would provide such a 'community discourse' for the domain, which can be considered a reference
source. This makes the knowledge precise and questionable for any domain. For example, Foundational Model of Anatomy (FMA) (Rosse et al., 2008) is a standard reference of anatomical terminologies. This ontology also includes using existing domain knowledge as an input to build new ontologies (Bontas et al., 2005). It can help to reduce the cost of ontology engineering and increase the quality of the final ontology (Lonsdale et al., 2010). A relevant case is when Jiménez-Ruiz et al. (2008), constructed the Juvenile Rheumatoid Arthritis Ontology (JARO) by reusing parts from NCI and GALEN.

**Controlled vocabulary:** Ontologies can be used as a controlled vocabulary, where categories are labelled to control conceptual modeling and meaning, reducing ambiguity and misconception. On that note, ontologies are controlled vocabulary, in which the categories represented are attached with labels to control their conceptual modeling and meaning to reduce ambiguity and misconception. For example, the Plant Ontology is used as a community resource to unify the terms for plant structure and development stages (Avraham et al., 2008). Another instance is UMLS (The Unified Medical Language System) (Bodenreider, 2004), the most extensive available compendium of biomedical vocabularies, which provides terms commonly used by physicians and molecular biologists.

**Schema and value reconciliation:** A schema and value reconciliation can be derived using the ontology model, by a community agreement on that model and all definitions and descriptions.

**Consistent query:** Querying is one of the most functional benefits of ontologies. By using any query tool, SPARQL, for instance, any desired information can be retrieved from the ontologies. Facilitating the consistent query is another use of ontologies; since the conceptualization, labels, and all categories are agreed and shared in the domain. As in Ramli et al. (2016), a historical event ontology is built by reusing the existing ontology (SNaP), which is then used in a semantic retrieval framework associated
with ontologies.

**Knowledge acquisition:** Ontologies in bioinformatics can be used as templates to identify the attributes of instances or to generate forms where instances can be acquired.

**Clustering and similarity:** Ontologies can be used to group a collection of semantically similar items in the domain. Ontologies organize the information into categories where terms are groups according to their relationships and the conceptual design.

**Indexing and linking:** Indexing and linking can be achieved since the ontologies represent structured and controlled vocabularies where retrieving information can be done through querying and searching. For example, The Medical Subject Headings (MeSH), is a controlled and hierarchically-organized vocabulary produced by the National Library of Medicine. It is used for indexing, cataloging, and searching of biomedical and health-related information\(^1\).

**Guidance and decision trees:** Ontologies can provide a powerful tool for creating guidance and decision trees that are consistent and accurate within a particular domain, and can help to support decision-making in a wide range of applications. This is because they model a domain with the appropriate constraints which support any decisions.

**Results Representation:** Ontologies can be used to facilitate the sharing and integration of data from different sources in a standardized way. For example, ISA-TAB (Investigation/Study/Assay Tabular format), is a format for representing and exchanging data in life sciences research. The format is based on a set of tabular files that describe the experimental design, sample characteristics, and assay data in a structured and machine-readable way. It includes three main types of files: Investigation, Study, and Assay, each providing different levels of detail about the research project. This format is designed to be flexible and extensible, allowing researchers

\(^1\)https://www.nlm.nih.gov/mesh/meshhome.html
to customize the format to their specific needs. It is also compatible with other data standards in the life sciences, such as the Minimum Information About a Microarray Experiment (MIAME) and Minimum Information About a Proteomics Experiment (MIAPE) standards (Sansone et al., 2012).

Adding domain assumptions and making them explicit: When the assumptions of the domain are explicitly implemented in an ontology, this makes it relatively easy to change the ontology if knowledge about the domain changes (Boyce et al., 2007).

Understanding communication: In order to exchange and share information, it has to have a precise meaning to ensure the effectiveness of communication between different users.

Ontologies have been used in science for many purposes, as some of their uses are listed earlier. They have various advantages that can be determined here. For example, sharing a common understanding of the knowledge in a domain can be facilitated by ontologies in a different usage, as in the controlled vocabulary, which helps in knowledge acquisition and understanding communication. Moreover, domain knowledge can be reused, which helps to reduce the time needed to build ontologies, which was illustrated earlier when reference ontologies are used to build new ones.

Notably, structuring the information is significantly important in order to obtain an accurate understanding of knowledge. Ontologies are perfect models to organize knowledge into accurate taxonomies. Going further, it can also be beneficial to retrieve information from the ontologies by using the queries; this is done via SPARQL Query Language² which can be used to query diverse data sources.

These features help analyze domain knowledge and support decisions because ontologies apply constraints and assertions to the domain knowledge.

All these uses and more are essential for users in dealing with knowledge, because the knowledge can be complicated and large to explore, without using such computational

²https://www.w3.org/TR/rdf-sparql-query/
models. Scientists are expanding the use of ontologies beyond the representation of their domain. Such scientific expansion may result in a diversity of affordances for ontologies, which reaches a wide range of users who benefit from the increased usage of ontologies. Therefore, the involvement of the domain scientists with the ontology developers plays a crucial role in improving and evolving the uses of ontologies over time.

In Section 2.4, I will describe the variety of users who are involved and interact with the ontologies.

2.1.2.3 Challenges in Using and Developing

Ontology adoption can be hampered by several factors, including the cognitive sophistication of ontology languages, such as in OWL, which will be described in detail in Section 2.2.1.4. In addition, the learning and comprehension of these sophisticated languages can be difficult, plus the expertise knowledge of modelling specific domains is crucial for developing successful ontologies (Vigo et al., 2014).

The complexities of ontology authoring and the prevalence of ontology authoring tools among ontology adopters have been consistently found to reduce the quality of ontologies and affect their adoption. It is still unclear how many current tools would support ontology creation, exploration, refactoring, and debugging (ibid.).

In Vigo et al. (2019), axiom addition was explored, which is carried out individually. Their approach directly asks ontology authors open-ended questions to ascertain their problems when performing authoring tasks individually. They carried out an interview study with 15 ontology experts to identify design recommendations for ontology development, and they came up with the following conclusions:

- In order to design and implement ontologies to represent and incorporate new domains of knowledge, appropriate software tools are required.

- When more knowledge domains are brought into ontology models, ensuring that they are accurate and consistent becomes a must.
• The wider the use of ontologies, the greater the importance of their models’ ontological consistency and accuracy.

• To provide ontology engineers and ontology developers resources, one must understand the authoring pattern for ontology workflow.

• As a result, technical specifications have been prioritized over the user interface.

As pointed to, ontologies are designed to facilitate the sharing of knowledge and the organisation of information within domains and communities. Therefore, users tend to facilitate convenient software and interfaces to record their knowledge and organize information to share and exchange it. So, the development process is crucial to produce such robust resources.

Moreover, biomedical ontologies within the same domain may exhibit significant structural variations to the point of conceptual incompatibility due to overlapping constraints (Faria et al., 2018).

Additionally, carefully developed user interfaces, graphical representations, and interaction design strategies can assist in increasing the adoption of ontologies and leading to higher quality results in other ontology-based applications and more (Ivanova et al., 2019). These challenges emphasize the suitability of tools and the methods for interacting with various users during the ontology creation process. Furthermore, they demonstrate the critical importance of tools and interaction techniques for developing ontologies.

2.1.2.4 Applications

This section describes the broad range of applications that ontologies are used for, other than the use of ontologies within the biological domain as mentioned earlier in Section 2.1.2.2.

As previously demonstrated, ontologies are helpful for effectively representing and organizing knowledge. They also had much help organizing and presenting commonly accessible information and knowledge on the internet (Gokhale, 2009), in medical health ser-
2.1 Conceptual Framework: Ontologies

Ontologies (Macedo et al., 2011), and decision support systems (Musen et al., 2014). Additional applications include:

a) Linguistic and Lexical Ontologies: they are for natural language processing and other language aspects including meanings, translations, and corpora such as WordNet\(^3\) and Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE)\(^4\).

b) Health and Life Sciences Ontologies: they are used to organize and represent the increasingly evolving knowledge in life sciences such as Gene Ontology and SNOMED.

c) Upper ontologies: these are basic and foundational ontologies with general terms that are applied to all domains such as Basic Formal Ontology (BFO)\(^5\) and SUMO\(^6\).

d) Ontologies for Industry: these are used for putting smart industrial information into a standardised format that is interoperable (Sampath Kumar et al., 2019) such as Enterprise Ontology (Dietz, 2006) and IOF\(^7\).

According to these applications, various ontologies will be described next, based on their abstraction level, applications, and usage.

**Ontology Types:**

Abstraction is a fundamental concept in various disciplines, including computer science, software engineering, and modeling. It can simplify any knowledge and experience to its essentials, using brief words (Kramer, 2007). Moreover, an *ontological approach* is defined as using abstract concepts rather than description techniques, and in this way, any level of ontology can be defined using the same ontology language. For example, the class **Person** can be used in multiple ontologies that make use of humans in their model (Shekhovtsov et al., 2014).

---

\(^3\)[https://wordnet.princeton.edu/]

\(^4\)[http://www.loa.istc.cnr.it/dolce/overview.html]

\(^5\)[http://basic-formal-ontology.org/]

\(^6\)[https://www.ontologyportal.org/]

\(^7\)[https://www.industrialontologies.org/]
Thus, ontologies can be broadly categorized based on their level of abstraction: these types are summarised in Figure 2.3, and described as follows (Petrov, 2011):

**Upper ontology** has concepts that are not domain specific but are broadly applicable across many domains; they are most useful in supporting the interoperable development of ontologies.

**Domain ontology** has the concepts relevant to an area of interest. In this type of ontology, the terms are more domain-specific, particularly the area pertaining to knowledge of interest. Several examples of domain ontologies will be described thoroughly in the next section.

**Inference ontology** has concepts relevant to the juncture of two disciplines. The multi-view representations are suited for domain models and knowledge management. Thus, one single model may support several views of the same thing; this reveals the implicit knowledge and limits modeling effort (Fortineau et al., 2015).

**Process ontology** has inputs, outputs, constraints, and sequencing information involved in business/engineering processes.

**Interface ontology** for a specific interface defines the structure, content, messaging, and other constraints. For example, User Interface Ontology (UIO) defines semantic rules of the user interface for developing a GUI. A domain ontology can be mapped to a UIO model to provide an abstract base model for the domain ontology’s user interfaces (Shahzad et al., 2011).

**Service ontology** defines a core collection of constructs for defining service vocabularies and capabilities (e.g., Semantic Web Services Ontology (SWSO)\(^8\)).

**Role ontology** defines the applicable terminologies and concepts for a specific end-user. Fukazawa et al. (2006) developed a role-ontology and used it to enhance the usability

\(^8\)https://www.w3.org/Submission/SWSF-SWSO/
of task-based service navigation. They associate tasks with role concepts identified in the role ontology to enhance a fundamental task model.

These ontologies differ in their degree of abstraction and thus in their intended use and application. This is because the concepts, not the ways they are described, matter at the ontological level. The following section will cover biological applications of ontologies.

### 2.1.2.5 Sample Ontologies

Table 2.1 briefly describes sample domain ontologies, and much of the information from the previous sub-sections is included as well. These ontologies are widely used, and many integrated tools are built to interact with the information presented by these ontologies. The diversity of the ontology applications in biology and medicine indicates how these ontologies manage such live-growing information.
Table 2.1: Examples of ontologies.

<table>
<thead>
<tr>
<th>Ontology</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAMBIS</td>
<td>Transparent Access to Multiple Bioinformatics Information Sources is an application that allows biologists to ask rich and complex questions over a range of bioinformatics resources. It is based on a model for the knowledge of concepts and their relationships in molecular biology and bioinformatics (Stevens et al., 2000a).</td>
</tr>
<tr>
<td>FMA</td>
<td>Foundational Model of Anatomy(^9) is a domain ontology that describes a coherent body of clear declarative information regarding human anatomy. It can be used and customized for any computer-based application requiring anatomical data. It is intended to be human-readable and computer-navigable. The ontological structure of the FMA enables machine-based inference, allowing future powerful computational tools to reason about biomedical data (Rosse et al., 2008).</td>
</tr>
<tr>
<td>Uberon</td>
<td>Uberon (Mungall et al., 2012) is a cross-species ontology that includes 6,500 groups, clustered around conventional anatomical concepts. The ontology describes structures in a species-neutral manner and provides detailed associations to established species-centric anatomical ontologies, enabling model organism and human data to be integrated. It employs innovative taxonomic variation representation methods and has proven to be crucial for translational phenotype analyses. Uberon serves as a vital link between anatomical structures across taxa, allowing for cross-species inference(^10).</td>
</tr>
</tbody>
</table>

Section 2.1.3 discusses other widely used examples of ontologies in the domain of biology.

2.1.3 Examples of Ontologies

2.1.3.1 Gene Ontology

The Gene Ontology (GO) (Ashburner et al., 2000) is a large ontology\(^11\) that provides controlled vocabularies of defined terms, representing gene product properties which cover three domains: cellular component, molecular function, and biological process\(^12\). These

---

5https://www.ebi.ac.uk/ols/ontologies/fma
6http://uberon.org
7https://www.ebi.ac.uk/ols/ontologies/fma
8http://uberon.org
9https://www.ebi.ac.uk/ols/ontologies/fma
10http://uberon.org
11With over 44 thousands valid terms and over 8 million annotations in the latest statistics at http://geneontology.org/stats.html (accessed on September 2020).
12http://geneontology.org/
domains appear in the annotation of the gene product. For example, the gene product ‘cytochrome c’ can be described by the cellular component mitochondrial matrix, the molecular function oxidoreductase activity, and the biological process oxidative phosphorylation.\textsuperscript{13} GO is designed to include terms of any species (both prokaryotes and eukaryotes), as well as single and multicellular organisms (The Gene Ontology Consortium, 2018).

**GO Structure**

Classes in GO (sometimes denoted as terms) are the basic elements in the ontology. Each class, essentially has a unique identifier, aspect, definition, and relationships to other terms. Other optional elements can be associated with the class. All essential and optional elements of the class are briefly described in Figure 2.4.

The structure of GO terms can also be described and represented as a graph. In Figure 2.5,\textsuperscript{13} Taken from [http://geneontology.org/docs/ontology-documentation/](http://geneontology.org/docs/ontology-documentation/)
nodes represent the terms, and edges represent the relations. This graph is generated using the OntoGraf tool in Protégé (Falconer, 2010). A term in GO can have many parents. For example, 'hexose biosynthetic process' has 'hexose metabolic process' and 'monosaccharide biosynthetic process' as parents. This allows for more expressivity than a simple hierarchy (Gaudet et al., 2017).

Figure 2.5: GO Overview.

Relations in GO are categorized and defined by using the following often occurring relationships: is a (subclass), part of, has part, regulates, negatively regulates, and positively regulates. These relations can be illustrated and found as follows:

<table>
<thead>
<tr>
<th>GO:0050794 regulation of cellular process</th>
<th>is a</th>
<th>GO:0050789 regulation of biological process</th>
</tr>
</thead>
<tbody>
<tr>
<td>GO:0050794 regulation of cellular process</td>
<td>regulates</td>
<td>GO:0071840 cellular process</td>
</tr>
<tr>
<td>GO:0022607 cellular component assembly</td>
<td>part of</td>
<td>GO:0044085 cellular component biogenesis</td>
</tr>
</tbody>
</table>

For example, Figure 2.6 shows GO relations with the colored edges\(^\text{14}\).

**Reflections on GO**

GO is an example of domain ontology (mentioned in 2.1.2.4) since it represents the three domains of gene product mentioned earlier. Also, it gives an example of several uses from Section 2.1.2.2 as we described.

\(^\text{14}\)This graph is generated using the AmiGO browser on 21/09/2020
2.1 Conceptual Framework: Ontologies Chapter 2

Figure 2.6: The structure of GO as a graph showing the relations between terms.

Firstly, it provides a standardized vocabulary for describing gene and gene product functions which illustrates the use of controlled vocabulary, as mentioned earlier 2.1.2.2. Secondly, GO can be used to query a database in search of genes’ functions or locations within the cell, to search for genes that share characteristics (Arnaud et al., 2009), as well as to determine which processes are different between sets of genes (fulfilling the use of queries from Section 2.1.2.2).

In addition, GO is helpful for analyzing the results of high-throughput experiments (Lovering et al., 2008) and inferring the functions of genes that have no annotation (Burge et al., 2012) (showing uses from 2.1.2.2 about results representation). Moreover, it is possible to compare proteins annotated to different terms if these terms are related using the
hierarchical structure of the GO (Gaudet et al., 2017).

2.1.3.2 ICD-11

International Classification of Diseases (ICD)\textsuperscript{15} is the preeminent tool for coding diagnoses and documenting investigations or therapeutic measures within the health care systems of many countries (Treede et al., 2015).

It is an international standard for systematic recording, reporting, analysis, interpretation, and comparison of mortality and morbidity data. It is used as a diagnostic tool for epidemiology, health management, and clinical purposes. Many health-related users, such as physicians and researchers, use it mainly to categorize diseases and other registered health problems, including when recording the cause of death in health records. It also provides an overview of the public health conditions of the world. Additionally, ICD helps in making decisions by countries for repayment and resource allocation (Reflow et al., 2013).

It is also used to monitor health of populations (large and small), plan how services are delivered, resources are allocated, help diagnose diseases, and detect a change in disease patterns.

ICD provides a common language used worldwide by doctors, nurses, policymakers, researchers, and many others to classify diseases/conditions to have the best possible picture of everyone’s health. More than 100 countries have used it to record the reasons people die, monitor the health of populations, and help make financing decisions about the health systems.

It is designed to map health conditions to corresponding generic categories and specific variations, assigning a designated code up to six characters long. This data forms the basis of comparison and sharing between health providers, regions, countries, and over time (Tudorache et al., 2011).

ICD-11 Components

\textsuperscript{15}https://icd.who.int/dev11/l-m/en
The current review of ICD-11 functions as an electronic environment that can be used online or offline. It has the following components, which are illustrated in Figure 2.7:

**ICD-11 Browser:** The web-based browser tool allows users to retrieve concepts by searching terms, anatomy, or any other element (see Figure 2.8).

**ICD-11 Coding Tool:** The Coding Tool works by searching ICD content. As the user types in a term 'cholera', for example, it generates (and dynamically updates) three different outputs; a word list, matched entities with a link to the browser, and the chapters associated with the target term (see Figure 2.9).

**Foundation Component - Index, Guidance:** The underpinning repository or database with all ICD entities. These entities comprise diseases, disorders, injuries, external causes of injuries with signs/symptoms, and their relationships.

**Linearization for Mortality and Morbidity Statistics (MMS):** A subset of the foundation component, that is fit for a particular purpose (reporting mortality, morbidity, or other uses). It is organized as a hierarchical structure with tens of thousands of
entities. All entities are mutually exclusive of each other and can only have a single parent.

The Translation Tool: This tool allows specific Language User Centers to build their translations close to the original. The tool also notifies other registered translators of changes and provides the same outputs in multiple languages.

Application Programming Interface (API services): Users must first register via the site, and may then, use it to access up-to-date documentation on using the API, as well as managing the keys needed for using it.

ICD-11 Purpose and Uses

ICD delivers an eclectic and hierarchical list of a wide range of diseases, disorders, injuries, and other health-related conditions that allows for the following\(^\text{16}\):

- Easy storage, retrieval, and analysis of health information for evidence-based decision-making

\[^{16}\text{https://www.who.int/classifications/icd/en/}\]
2.1 Conceptual Framework: Ontologies

2.1.3.3 SNOMED CT

SNOMED CT is a structured clinical vocabulary that is used in an electronic health record. It is the most comprehensive and precise clinical health terminology product in the world\(^\text{17}\). As ICD-11, SNOMED CT also has multilingual clinical healthcare terminology and enables the consistent representation of clinical content in electronic health records. Further, it is mapped to other international standards such as ICD, and is used in more than eighty countries.

\(^{17}\text{https://digital.nhs.uk/services/terminology-and-classifications/snomed-ct}\)

---

**Figure 2.9: ICD-11 Coding Tool.**

- Sharing and comparing health information amongst hospitals, regions, settings, and countries
- Data comparisons in the exact location across different periods

**Reflections on ICD-11**

This ontology example of ICD-11 shows the use of controlled vocabulary and sharing common terminologies, which helps to conduct analysis and statistics in the health domain, plus perform queries on the system, helping diagnoses.
2.1 Conceptual Framework: Ontologies

Advantages

The benefits of using SNOMED CT in electronic care records are subsequently explained:

- Sharing vital information consistently within and across health and care settings
- Comprehensive coverage and greater depth of details and content for all clinical specialties and professionals
- Including diagnosis, procedures, symptoms, family history, allergies, assessment tools, observations, and devices
- Supporting clinical decision-making
- Facilitating analysis to support more extensive clinical audits and research
- Reducing risk of misinterpreting records in different care settings

SNOMED Components

The core component types in SNOMED CT are concepts (classes), descriptions, and relationships, described below. The high-level logical model, illustrated in Figure 2.10, shows the concept-centric design of the SNOMED, which has been modified from the online library of SNOMED\(^{18}\).

- **Concepts:** Each concept represents a unique clinical meaning with a unique numeric and machine-readable SNOMED CT identifier. The identifier provides an unambiguous unique reference to each concept and does not have any ascribed human interpretable meaning.

- **Descriptions:** Two types of description are used to represent every concept (Fully Specified Name (FSN) and Synonym). The FSN represents a unique, unambiguous description of a concept’s meaning. It is beneficial when different concepts are referred to by the same commonly used word or phrase. Each concept can have only one FSN.

\(^{18}\)https://confluence.ihtsdotools.org/display/DOCRELFMT/2.1+High+Level+Logical+Model+of+SNOMED+CT
in each language or dialect. A synonym represents a term that can be used to display or select a concept. A concept may have several synonyms. This synonym allows users of SNOMED CT to use the terms they prefer to refer to a specific clinical meaning.

- **Relationships:** A relationship represents an association between two concepts. Relationships are used to logically define the meaning of a concept in a way that a computer can process. Actually, a relationship type (or attribute) represents the meaning of the association between the source and destination concepts.

![Figure 2.10: A high-level SNOMED Logical model.](image)

SNOMED provides an international browser tool that supports multiple countries. For example, The NHS Digital SNOMED CT Browser provides ways for the browser to search the SNOMED CT UK Edition. The SNOMED CT UK Edition is released twice per year. It consists of the International Edition, plus the UK-specific content provided within the UK Clinical Extension and UK Drug Extension, including maps to ICD-10 and OPCS-4. This is for use in the UK only\(^\text{19}\).

\(^{19}\)https://termbrowser.nhs.uk
In Figure 2.11, a screenshot of the browser is shown. There are two main parts in the browser; Taxonomy Panel (Left) and Concept Details Panels (Right).

The taxonomy panel displays the SNOMED CT hierarchy. It has configuration options for each panel, where the view can be set to Inferred or Stated view. Concepts can be navigated using the hierarchy tree. Moreover, the search facility in this panel allows searching SNOMED CT content (see Figure 2.12).

Concept Details Panels display the attributes, descriptions, and relationships of SNOMED
CT concepts (either in Summary, as a quick overview of the definition, or Details, with a detailed list of the concept components). Finally, there exists a Classification Maps option, which is a link from SNOMED CT to other resources such as ICD-10.

Other tooling provided by the SNOMED CT includes the following:

- Refset Management and Translation, which allows the translation of SNOMED into other languages.
- Mapping Tool, which enables mappings from SNOMED to other code sets.
- Authoring Platform, which allows customers and stakeholders to author the content in the SNOMED.
- Collaborative environment, which encourages collaboration between different users and members.

Reflections on SNOMED CT

SNOMED CT standardizes how health data is captured, shared, and aggregated across specialty and treatment facilities. This detailed terminology is for input, not reporting. ICD-11 combines various diseases and procedures, grouping them together for easy retrieval. These terms are used in applications where data aggregation is beneficial, like measuring the quality of care, monitoring resource usage, or processing insurance claims. Both systems (SNOMED and ICD-11) are similar in that they use alphanumeric codes to represent clinical concepts. However, the systems are functionally and structurally distinct, and are best utilized in different ways in healthcare.

2.2 Technologies

This section describes the technologies used with their ontologies. It starts from the languages used to encode and represent knowledge in ontologies and then explains how logic and semantic web led to the domination of OWL (primary focus of this thesis). Lastly, it shows various representations of OWL ontologies, using different tools and formats to explore the ontologies.
Table 2.2: Biological ontologies comparison.

<table>
<thead>
<tr>
<th>Item</th>
<th>GO</th>
<th>ICD-11</th>
<th>SNOMED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>cellular component -</td>
<td>Classification of Diseases</td>
<td>clinical health terminology</td>
</tr>
<tr>
<td></td>
<td>molecular function -</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>biological process</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of classes</td>
<td>44272 (plus 8,054,893</td>
<td>over 100,000</td>
<td>354,383 (2020-07-31)</td>
</tr>
<tr>
<td></td>
<td>annotations)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Community</td>
<td>GO Consortium</td>
<td>Medical and healthcare with Statistics</td>
<td>Clinical and Healthcare</td>
</tr>
<tr>
<td>Update period</td>
<td>Monthly</td>
<td>10 years</td>
<td>6 months</td>
</tr>
<tr>
<td>Organization</td>
<td>W3C</td>
<td>WHO</td>
<td>ihtsdo</td>
</tr>
<tr>
<td>Countries</td>
<td>Worldwide</td>
<td>&gt;100</td>
<td>&gt;80</td>
</tr>
</tbody>
</table>

2.2.1 Ontology Languages

Formal languages are used to encode the knowledge and construct ontologies with reasoning support. In the following sections, these languages are described, starting with first-order logic, description logics, and then OWL, which is the main focus of this research. Table 2.3 summarizes the languages described here, their advantages, and challenges of use.

2.2.1.1 On the Use of Logic for Ontologies

Logic has appeared as a philosophical term, referring to the study that concerns questions about reference, predication, identity, truth, quantification, existence, entailment, modality, and necessity (Lowe, 2013). It is also used in maths and computer science which refers to studying specific mathematical properties of artificial and formal languages, as well as dealing with valid inferences and reasoning based on them (Hofweber, 2004). In addition, the reasoning is essential to utilize the rich information and structure of ontologies, as well as to ensure the quality of an ontology (Baader et al., 2003) because it helps to detect errors and sometimes correct them.

First-order Logic

First-order logic (FOL) is used in artificial intelligence to represent knowledge. It has a sufficient expressivity mechanism to represent the natural language statements concisely,
Table 2.3: Summary of languages used for ontologies.

<table>
<thead>
<tr>
<th>Users (Who?)</th>
<th>Logic</th>
<th>Web</th>
<th>Semantic Web</th>
<th>OWL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Logicians and mathmaticians</td>
<td>Web developers</td>
<td>Web and ontology developers</td>
<td>Ontology developers</td>
</tr>
<tr>
<td>Reasons to use (Why?)</td>
<td>Makes ontologies easier to read and understand</td>
<td>Opens connectivity for humans</td>
<td>Links documents to each other but also information and recognizes the meaning</td>
<td>Has a good level of expressivity</td>
</tr>
<tr>
<td>Challenges of using (What cons?)</td>
<td>Undecidability and computational complexity</td>
<td>Misunderstanding of knowledge</td>
<td>Limited expressivity and capturing the necessary relationships</td>
<td>Might increase the complexity to reach a higher expressivity</td>
</tr>
<tr>
<td>Advantages to use (What pros?)</td>
<td>Expressivity and inference capabilities (DLs)</td>
<td>Share and exchange information</td>
<td>Model domain knowledge and helps to represent/structure of data</td>
<td>Expressivity and inference capabilities due to the use of DLs</td>
</tr>
</tbody>
</table>

Unlike propositional logic\(^{20}\) which is the foundation of first-order logic (Wikipedia contributors, 2020b). In addition, first-order logic can easily describe information about objects and express the relationship between those objects.

In first-order logic, the world has objects, relations, and functions, besides the facts (like natural language). Objects can be any variable such as people, numbers, cars, A, B. Relations can be a unary relation such as green, round, is adjacent, or n-any relation such as the sister of, brother of, has color. Functions are a special type of relations which apply binary relations where one input has only one value, for example, a function can be father of, best friend, end of (JavaTpoint, 2020).

First-order logic is also known as predicate logic. A predicate is based on the distinction between concepts and objects. It is a statement that contains a variable, and it may be true or false depending on the values of its variables. For example,

\(^{20}\)Propositional logic describes the world as facts; therefore the expressivity is limited.
$P(x) = "x^2 \text{ is greater than } x"$ can be true if $x = 2$ and false if $x = 1$.

First-order uses quantifiers such as $\forall$ universal (for all), $\exists$ existential (there exists), as well as logical connectives, $\land$ (and), $\lor$ (or), $\neg$ (not), $\rightarrow$ (implies), $\equiv$ (equality) (Barwise, 1977), to fully express a model of the world. The following sentences can be interpreted into first-order logic:

Sentence1: All students are people.
FOL: $(\forall x (\text{is-student}(x) \rightarrow \text{is-person}(x)))$

Sentence2: Everyone is a friend of someone.
FOL: $(\forall x (\exists y (\text{is-friend-of}(x, y))))$

Although first-order logic is richly expressive, it is undecidable, which is the inability to construct a single algorithm, which always leads to a correct yes-or-no answer (ibid.). This limitation is reflected in the computational complexity of solving first-order logic problems. Other logics exist which can represent knowledge, such as frame-based or description logic, which do not have this problem.

**Frame-based Logic**

Frame-based logic (F-logic) is a logic for frame-based knowledge representation. Frame logic integrates the benefits of conceptual modeling with object-orientation and provides a declarative, compact, and simple syntax, as well as the well-defined semantics of a logic-based language (Wikipedia contributors, 2020a).

Like object-oriented systems, frame-based systems utilize the concepts of classes, instances, and inheritance (Alkrimi et al., 2013). The class/frame has attributes (or slots) that determine the properties of that object.

For example, the class *City* is defined with subclasses *EuropeanCity*, *AmericanCity*, *HolyCity* and so on. These subclasses can inherit the attributes from the superclass, such as country, place, population, and more. Then, any instance can be defined using this

---

information like London, Texas, and Rome, which will inherit the information from their classes (see the following example).

\[
\text{(AmericanCity} \\
\text{<:IS-A City>} \\
\text{<:Place NorthAmerica>} \\
\text{<:Country USA>}...) \\
\]

Then an instance can be declared as:

\[
\text{(dallas} \\
\text{<:INSTANCE-OF AmericanCity>} \\
\text{<:State texas>} \\
\text{<:Population 6.301M}>...) \\
\]

F-logic is one of the early formalisms for ontologies. This frame-based representation aims at making ontologies easier to read and understand, particularly for users not familiar with (first-order/Description) logics, because it is simple and easy for humans to understand. The frames paradigm has been used in several well-known knowledge representation systems, including Protégé (N. F. Noy et al., 2000). Protégé-frames are illustrated in Figure 2.13.

Although frame-based logic is less expressive than FOL, it is more scalable, and the frame-based representation is much easier to understand for domain experts than FOL (N. F. Noy et al., 2004) or Description Logic.

However, frame-based formalism limits the inference ability from the knowledge, such as classification and the detection of any inconsistent concepts (Abu-Hanna et al., 2005). These inference facilities require more expressive and formal language combined with reasoning. Therefore, Description Logic came to provide formalisms with such features. For example, OWL is a syntactic layer over different description logic levels. We will describe OWL in more detail later on in Section 2.2.1.4. Description logic will be discussed next.

Description Logics (DLs)
As mentioned earlier, computational ontologies were developed using frames and First-order logic (Gómez-Pérez et al., 2004). However, first-order logic is undecidable, even though its expressivity is rich. Additionally, frame-based representation has limited inference facilities. In contrast, Description logics are more advanced in that they are decidable and less complex than First-order logic. This concept is defined as a subset of first-order logic fragments and is developed as a logical formalization of semantic networks (Krötzsch et al., 2012). The first-order logic subsets are chosen, so they remain decidable while keeping the strong inferencing ability of first-order logic. Indeed, the use of description logics have advanced, and these what are presently in use. Description Logics have different expressivity levels and can formalize the semantic networks and frame-based systems. Examples of description logics are $\text{ALC}, \text{SHOIN}(D), \text{SHOIQ}$ etc.

Description logic has two formalisms; terminological formalism (called TBox) and assertional formalism (called ABox) (Baader et al., 2003). An example of these formalisms is given below:
TBox: \( \text{HappyMan} \sqcap \exists \text{hasChild}.\text{Human} \sqsubseteq \text{Human} \)

ABox: \( \text{HappyMan}(BOB), \text{hasChild}(BOB, MARY) \)

Since DLs provide well-defined semantic and powerful reasoning tools, they became the basis for several web ontology languages, including OWL.

In order to maintain the consistency and quality of an ontology, reasoning is an essential chore in different phases of the ontology life cycle, which helps to utilize the rich structure of knowledge within it (Baader et al., 2007). Description logic also has various inference capabilities that enable users to deduce implicit knowledge from the explicitly represented knowledge (ibid.). Moreover, reasoning over logics has gone through many variations, initially using structural rearrangements, then moving into tableaux algorithms.

There are many different representations and syntaxes for Description Logic. The following two are briefly described since they are the predecessors of OWL:

**Ontology Interchange Language (OIL)** is an ontology language that is based on description logic. It also incorporates essential modeling basics of frame-based systems such as the notion of a class and the definition of its superclasses with attributes. Relations are defined as independent entities with domain and range, plus they can be arranged in hierarchy-like classes. OIL inherited from DL, its formal semantics, and the efficient reasoning support (Fensel et al., 2001). It was also intended to be an expressive description logic integrated with Web technology (Mcguinness et al., 2002).

**DARPA Agent Markup Language (DAML)** project aimed to develop a language and tools to facilitate the concept of the Semantic Web\(^{22}\). DARPA stands for Defence Advanced Research Projects Agency and is the central research and development organization for the Department of Defence. DAML was then merged with OIL to become DAML+OIL; a successor which combines features of both languages under W3C standard for ontological representation (Bechhofer et al., 2000; Giri, 2011).

Eventually, those languages were superseded by OWL. As the goal of these technologies

\(^{22}\text{http://www.daml.org/}\)
is to integrate ontologies and knowledge representation with web technologies, I will next explore the evolution of web technology to adapt to the expansion of sharing knowledge.

### 2.2.1.2 The Web

The World Wide Web (WWW)\(^{23}\) was invented by Sir Tim Berners-Lee in 1989\(^{24}\). It is a convenient way for people to share, browse and navigate data. It is aimed mainly for human use (Chang et al., 2006) due to the facilities of linking and displaying various information with different formats, such as images, texts, and documents. It facilitates the connection between the users and enables them to share their information easily on the internet.

The main technologies behind the web are HyperText Markup Language (HTML), Hypertext Transfer Protocol (HTTP), and Uniform Resource Locator (URL). HTML is the main markup language that has been used for creating a webpage where each document/resource in the web is given a unique URL and browsers can use HTTP protocol to resolve these URLs into documents that they can display. Figure 2.14 illustrates the early web model. This revolution of technologies has been spread and grown all over the world.

HTML was primarily designed for viewing, but it was also used informally for structuring knowledge. For machine interpretation, HTML was generalized into eXtensible Markup Language (XML), which allows the creation of arbitrary new markup and the description of data. XML is, therefore, a generalized syntax for the web. More details about XML will be explained in Section 2.2.1.3.

The web is an environment with open connectivity that enables many people to share data; however, there is also a need for precise semantics because humans and computers need to communicate meaningfully. That is, the terms should be understandable by the computers in order to deliver correct services. XML can be a partial solution, but it requires people to agree in advance on the underlying meaning of the markup.

Furthermore, there may be widely divergent perspectives and assumptions about the information due to diverse requirements and situations. This diversity can be related to the

---

\(^{23}\)Which is also known as W3 or Web.

\(^{24}\)[https://www.w3.org/People/Berners-Lee/Overview.html](https://www.w3.org/People/Berners-Lee/Overview.html)
same subject matter (Hepp et al., 2007).

This misunderstanding of knowledge can lead to poor communication within and between people, organizations, and systems. Hence new needs arose to share knowledge and enable multiple information systems to communicate and exchange data. This leads to the new generation of the ‘Semantic Web’ discussed and explained afterwards.

2.2.1.3 Semantic Web

The Semantic Web will bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users (Berners-Lee et al., 2001).

The semantic web is an evolutionary version of the web that gives meaning to things in the web. The semantic web is involved in sharing, linking, and reusing data from multiple sources. It not only links documents to each other but also recognizes the meaning of the information in those documents (Frauenfelder, 2001).
Computers struggle to recognize the precise meaning of information, which is a human capability of using the natural language. For example, the word ‘Jaguar’ can be interpreted by two different meanings; either, the animal or the car. Therefore, adding semantics will enhance the web and enable search engines to read and collect any data from diverse sources, process it, and infer new facts (Frauenfelder, 2001).

**eXtensible Markup Language (XML)**

XML enhances the readability of people and machines by using tags to describe the data (Bray et al., 2008). This language enables users to define the representation and structure of data, where values are assigned in each field in the structure. Developers can create and manage their formatting tags, content, and hyperlinks (Laurent, 1997). An example of XML script follows as the web page containing contact information of the author.

```xml
<?xml version = "1.0"?>
<contactinfo>
 <address category = "university">
   <name>Aisha Blfgeh</name>
   <School>School of Computing</School>
   <University>Newcastle University</University>
   <mobile>0123456789</mobile>
 </address>
</contactinfo>
```

While XML provides an interoperable syntactical foundation upon which the more critical issue of representing relationships and meaning can be built, URIs provide the ability to identify resources and relationships among resources uniquely. The Resource Description Framework (RDF) pulls the advantages of URIs and XML to provide a stepwise set of functionalities to represent these relationships and meanings (Berners-Lee et al., 2002), as shown next.

**Resource Description Framework (RDF)**

RDF is a data model to describe the semantics of information formally, but it can also
represent and process metadata. It provides interoperability between applications by exchanging machine-understandable information on the web. Also, it emphasizes facilities to enable automated processing of Web resources (Lassila et al., 1999).

The RDF model consists of the following triples: Subject, Predicate, and Object. In this context, the subject is the entity, the predicate is the attribute or property, and the object is the value. Each one of these elements in the trio is uniquely identifiable, and they can be represented in a directed graph with nodes and links, as in Figure 2.15, where many trios can also be connected. For example, for the trio 'Bob is-a Man', the subject is Bob, the predicate is is-a, and the object is Man. Similarly, for 'Fred hasAge 5', each entity in the triple has its unique identifier, as follows:

- Fred has “http://www.w3.org/Family/Fred”,
- hasAge has “http://www.w3.org/Family/#hasAge”, and
- The value 5 has “http://www.w3.org/Family/#hasAge/5”.

This provides a robust and straightforward framework for representing information on the web.

**RDF Schema (RDFS)**

RDF Schema is a semantic extension of RDF. It provides mechanisms for describing groups of related resources and the relationships between them. On that account, a schema is a place where definitions and restrictions of usage for properties are documented. In order to avoid confusion between independent – and possibly conflicting – definitions of the same term, RDF uses the XML namespace facility.

In the RDF Schema (RDFS), the class and property system is similar to the systems of object-oriented programming languages such as Java. The distinction is that RDFS expresses properties in terms of the classes they apply via domains and ranges. In other systems, a class is defined in terms of its properties, and any instances it may have (Brickley, 2000). For instance, in RDFS, one could define the eg:author property with a domain of eg:Document and a range of eg:Person, whereas in a classical object-oriented
system, a class `eg:Book` is typically defined with an attribute called `eg:author` of type `eg:Person` (Brickley et al., 2014).

While RDF and RDFS languages already allow modeling domain knowledge, they are not very expressive and often insufficient for capturing the necessary relationships and constraints (Guha et al., 2014).

As presented in Table 2.4, those are the agreed data schema: XML to structure data using some tags; then RDF, which uses triple schema of Subject Predicate and Object to represent the data in a more precise way. Thereafter, comes the ontology to give formal semantics to the data, with the language being OWL. Therefore, the development of such richer representations was an early goal in the Semantic Web initiative, which eventually led to the OWL ontology language (Siekmann, 2014). In the next section, I will finally explain OWL and describe the underlying technologies which help to represent and develop ontologies.
Table 2.4: RDFS vocabulary (Adapted from Brickley et al. (2014)).

<table>
<thead>
<tr>
<th>Classes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs:Resource</td>
<td>The class resource or everything</td>
</tr>
<tr>
<td>rdfs:Literal</td>
<td>literal values, e.g. textual strings and integers</td>
</tr>
<tr>
<td>rdfs:langString</td>
<td>language-tagged string literal values</td>
</tr>
<tr>
<td>rdf:HTML</td>
<td>HTML literal values</td>
</tr>
<tr>
<td>rdf:XMLLiteral</td>
<td>XML literal values</td>
</tr>
<tr>
<td>rdfs:Class</td>
<td>The class of classes</td>
</tr>
<tr>
<td>rdf:Property</td>
<td>The class of RDF properties</td>
</tr>
<tr>
<td>rdfs:Datatype</td>
<td>The class of RDF datatypes</td>
</tr>
<tr>
<td>rdf:Statement</td>
<td>The class of RDF statements</td>
</tr>
<tr>
<td>rdf:Bag</td>
<td>The class of unordered containers</td>
</tr>
<tr>
<td>rdf:Seq</td>
<td>The class of ordered containers</td>
</tr>
<tr>
<td>rdf:Alt</td>
<td>The class of containers of alternatives</td>
</tr>
<tr>
<td>rdfs:Container</td>
<td>The class of RDF containers</td>
</tr>
<tr>
<td>rdf:List</td>
<td>The class of RDF Lists</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Properties</th>
<th>domain</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdf:type</td>
<td>rdfs:Resource</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdfs:subClassOf</td>
<td>rdfs:Class</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdfs:subPropertyOf</td>
<td>rdf:Property</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>rdfs:domain</td>
<td>rdf:Property</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdfs:range</td>
<td>rdf:Property</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdfs:label</td>
<td>rdfs:Resource</td>
<td>rdfs:Literal</td>
</tr>
<tr>
<td>rdfs:comment</td>
<td>rdfs:Resource</td>
<td>rdfs:Literal</td>
</tr>
<tr>
<td>rdfs:member</td>
<td>rdfs:Resource</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdf:first</td>
<td>rdf:List</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdf:rest</td>
<td>rdf:List</td>
<td>rdf:List</td>
</tr>
<tr>
<td>rdfs:seeAlso</td>
<td>rdfs:Resource</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdfs:isDefinedBy</td>
<td>rdfs:Resource</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdf:value</td>
<td>rdfs:Resource</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdfs:subject</td>
<td>rdf:Statement</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdf:predicate</td>
<td>rdf:Statement</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdf:object</td>
<td>rdf:Statement</td>
<td>rdfs:Resource</td>
</tr>
</tbody>
</table>
2.2.1.4 OWL

A descendent of the previously described languages in Section 2.2.1 has become a dominant ontology language, namely the Web Ontology Language (OWL) (W3C, 2012). OWL is an ontology language for the Semantic Web, developed by the World Wide Web Consortium (W3C) Web Ontology Working Group. OWL is primarily designed to represent information about objects and how objects are interrelated (Horrocks et al., 2003).

OWL 1 is based on $SHOIN(D)$, and it provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users$^{25}$. These languages outlined below are canonically represented in the semantic web and provide additional

---

$^{25}$Modified from W3C website https://www.w3.org/TR/2004/REC-owl-features-20040210/#s1.2
expressivity to RDFS.

**OWL-Lite** supports those users who primarily need a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for OWL-Lite than its more expressive relatives. It must be noted that, OWL-Lite provides a quick migration path for thesauri and other taxonomies. OWL-Lite also has a lower formal complexity than the next OWL-DL; therefore, the conclusion is that OWL-Lite ⊑ OWL-DL.

**OWL-DL** supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL-DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class). OWL-DL is therefore named due to its correspondence with description logic.

**OWL-Full** supports those users who seek maximum expressiveness with the syntactic freedom of RDF and can accept no computational guarantees, which leads to the undecidability as in FOL. For example, in OWL-Full, a class can be treated as a collection of individuals and as an individual simultaneously. OWL-Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. However, the lack of computational guarantees makes it hard to build reasoning software for OWL-Full.

Thus, OWL-Lite is the simplest version with limited expressiveness, then OWL-DL, based on description logic, and OWL-Full with maximum expressiveness but with no guarantee for logical consistency (Weller, 2010). It is also denoted that as:

OWL-Lite ⊑ OWL-DL ⊑ OWL-Full
OWL 2 is an extension and revision of the previously mentioned OWL1, and is based on $SROIQ$. It is also designed to facilitate ontology development and sharing via the web, with the ultimate goal of making Web content more accessible to machines (Group, 2012). Similar to OWL1, OWL2 has sublanguages, which are explained below:

**OWL2-EL** enables polynomial-time algorithms for all the standard reasoning tasks; it is particularly suitable for applications where very large ontologies are needed, and where expressive power can be traded for performance guarantees.

**OWL2-QL** enables conjunctive queries to be answered in LogSpace (more precisely, AC0) using standard relational database technology; it is particularly suitable for applications where relatively lightweight ontologies are used to organize large numbers of individuals and where it is valuable or necessary to access the data directly via relational queries (e.g., SQL).

**OWL2-RL** enables the implementation of polynomial-time reasoning algorithms using rule-extended database technologies operating directly on RDF triples; it is particularly suitable for applications where relatively lightweight ontologies are used to organize large numbers of individuals and where it is useful or necessary to operate directly on data in the form of RDF triples.

These are more restrictive than ‘OWL2-DL’ and ‘OWL2-Full’. ‘OWL2-Full’ refers to the language determined by the set of all RDF graphs being interpreted using the OWL2 RDF-Based Semantics, and ‘OWL2-DL’ is used informally to refer to OWL2-DL ontologies interpreted using the Direct Semantics. As such, it can be concluded that:

\[ \text{OWL2-EL, OWL2-QL, OWL2-RL} \subseteq \text{OWL2-DL} \subseteq \text{OWL2-Full} \]

These OWL expressivity levels are represented in Figure 2.17.

Going further, OWL has various syntaxes (e.g., RDF/XML, OWL/XML, and Manchester Syntax (Horridge et al., 2012)). In addition, it has additional features where it can import other terms and add annotations to concepts. It is also logically underpinned by Description Logic (Krötzsch et al., 2012).
In the succeeding section, an example of OWL ontologies will be shown in different syntaxes, including Tawny-OWL (see Section 2.2.2.3 for more details) and Manchester Syntax.

For the moment, an ontology is considered as a collection of terms and concepts organized logically with formal descriptions of relationships amongst them, covering a specific domain of knowledge. In Figure 2.18, a diagrammatic representation of the following ontological statement in Tawny-OWL is shown:
### Table 2.6: OWL constructors and the equivalents in DLs and FOL.

<table>
<thead>
<tr>
<th>OWL Axiom</th>
<th>DL</th>
<th>FOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersectionOf</td>
<td>Human □ Male</td>
<td>Human(x) ∧ Male(x)</td>
</tr>
<tr>
<td>unionOf</td>
<td>Doctor △ Lawyer</td>
<td>Doctor(x) ∨ Lawyer(x)</td>
</tr>
<tr>
<td>complementOf</td>
<td>¬Male</td>
<td>¬Male(x)</td>
</tr>
<tr>
<td>oneOf</td>
<td>expressive john, mary</td>
<td>john, mary</td>
</tr>
<tr>
<td>allValuesFrom</td>
<td>∀ hasChildDoctor</td>
<td>∀x.y hasChild(x,y) → Doctor(y)</td>
</tr>
<tr>
<td>someValuesFrom</td>
<td>∃ hasChildDoctor</td>
<td>∃x.∀y. hasChild(y,x) → Doctor(x)</td>
</tr>
<tr>
<td>maxCardinality</td>
<td>≤ 2 hasChild.Lawyer</td>
<td>≤ 2 hasChild(y,x) → Lawyer(x)</td>
</tr>
<tr>
<td>minCardinality</td>
<td>≥ 2 hasChild.Lawyer</td>
<td>≥ 2 hasChild(x) → Lawyer(x)</td>
</tr>
<tr>
<td>owl:Thing</td>
<td>⊤</td>
<td>⊤ Universal concept</td>
</tr>
<tr>
<td>owl:Nothing</td>
<td>⊥</td>
<td>⊥ Bottom concept</td>
</tr>
</tbody>
</table>

![Diagram](image.png)

Figure 2.18: A class Person and its equivalent class Human and disjoint subclasses, Man and Woman.

**Tawny-OWL**

``` Scheme
(defclass Person
 :equivalent Human)

(as-disjoint
 (defclass Man
 :super
 (annotate Person
 (owl-comment "Every man is a person")))))

(defclass Woman
 :super
 (annotate Person
 (owl-comment "Every woman is a person")))))
```
The above example is modified from OWL primer; it shows that the Person class is equivalent to the Human class. Moreover, the classes Man and Woman are subclasses of the Person class, and they are disjoint classes which means that no instance can be an example of both classes at the same time. For example,

```
(defindividual Chloe
  :type Woman)
```

Chloe is defined as a Woman class which infers that she cannot be a Man, as well as being a Person since Woman is a subclass of Person. This is illustrated in Figure 2.19.

Before exploring the OWL primer and syntax, the following are the building blocks of an OWL ontology. In OWL ontologies, the basic concepts are (modified from Keyser (2012)) expressed below.

- **Individual**: a single basic entity of the ontology.
- **Class**: a group of individuals sharing a number of properties (‘thing’ is defined as the class of all individuals).
- **Subclass**: a subset of a class that has specific attributes.
• **Property**: a relationship between individuals or between individuals and other objects.

• **Domain**: states from which class the property can be used.

• **Range**: limits the individuals the property can be linked to.

The importance of ontologies is growing due to their capacity for knowledge management and information integration, as demonstrated by the semantic web. Therefore, various methods represent these ontologies, including the graphical representations, which will be discussed next.

### 2.2.2 Ontology Representation

In this section, the representation of ontologies is described using various examples. As ontologies are based on formal representations and logics, they need specific formats to display their entities to be explored and visualized. The coming representation formats are a few of the various methods to visualize the ontology. Some are computationally
focused on being consumed by machines, and some are more human focused on being more readable by humans.

### 2.2.2.1 RDF/XML

There exist three official serialisation syntaxes for RDF: RDF/XML, N-Triple\(^{26}\), and more recently, the Terse RDF Triple Language (Turtle)\(^{27}\).

First, the RDF/XML syntax has the advantage of being a native XML format; though parsing it can be challenging as the native order-dependent tree model of XML documents gets in the way of the RDF graph model. This also makes the syntax notoriously verbose.

Back to the family ontology, the following sentence is an example from that ontology, meaning that all classes of father are the classes with a child from type Person:

\[
\text{Father} \sqsubseteq \forall \text{hasChild.Person}
\]

![RDF/XML Script](image)

This RDF/XML script tells that the ontology here is called family, Father is a subClassOf the class Man, and a subclass of the restriction on Father, that the restriction is owl:onProperty object property hasChild with the restriction that is allValuesFrom the class Person. This representation is not reading-friendly for humans; however, the equivalent Manchester syntax is more readable by a human, and it would look like below:

\(^{26}\)https://www.w3.org/TR/2004/REC-rdf-testcases-20040210/#ntriples  
\(^{27}\)https://www.w3.org/TeamSubmission/turtle/
As shown, RDF/XML is a verbose syntax and requires frequent adjusting of its tags and balancing the tree format. This tends to make it inconvenient for a human to use directly. The following format is user-friendly, with an entire graphical interface for users to click and browse the ontology.

### 2.2.2.2 Graphical User Interface (GUI)

In this user-friendly form of ontology representation, a graphical user interface is used to visualize the ontologies. The popular software of developing ontologies, Protégé (see Section 2.3.3), is depicted here. Protégé is an open-source tool that provides a user interface to develop and construct ontologies. It has been widely used for developing ontologies due to the variety of plug-ins and frameworks (N. F. Noy et al., 2003). Protégé provides an easy interface for the editing, visualization, and validation of ontologies (Horridge et al., 2011b). Figure 2.20 shows the ontology representation from Protégé screen. It has the hierarchy form in the left panel showing the structure of classes. On the right-side panel, the information about the selected class regarding axioms and properties is attached with the class and any individuals related to this.

### 2.2.2.3 Tawny-OWL

The format of Tawny-OWL was presented previously in Section 2.2.1.4 when the OWL example was described. Tawny-OWL is a textual interface for developing ontologies in a fully programmatic manner. Tawny-OWL has a simple syntax which was initially modeled on the Manchester Syntax (Horridge et al., 2012), modified to conform to standard Clojure syntax and to increase regularity (Lord, 2014).

This section shows the programmatic environment of Tawny-OWL displayed in the fol-
following script, showing the command for defining class and properties, plus adding some restrictions. However, more a detailed description about Tawny-OWL and its features will be discussed later in Section 2.3.3.6 and more information exists in the project Github page28.

28https://github.com/phillord/tawny-owl/
Here, various syntaxes are used to represent and view ontologies: RDF/XML is verbose and mainly for machine processing, GUI is of benefit to users due to the graphical interface and icons but requires skills to deal with the software, and Tawny-OWL is the programmatic textual environment with a relatively straightforward syntax to be comprehensible for users. Nevertheless, building an ontology is difficult and requires iterations and refinement as well as collaboration between different experts and tools.

In the subsequent section, different methodologies will be described, with the tools for building the ontologies.
2.3 Ontology Engineering

"the set of activities that concern the ontology development process, the ontology life cycle, and the methodologies, tools and languages for building ontologies" (Gómez-Pérez et al., 2006).

Section 2.1 discussed how ontologies are used in a variety of areas to represent and manage data computationally across a wide range of applications and consumers. Therefore, ontology engineering is a well-investigated field where numerous methodologies have been proposed and inspired by software engineering, some of which incorporate Human Computer Interaction (HCI) techniques that enable efficient collaboration during the ontology engineering cycle. Furthermore, ontology development is not a linear process; it requires multiple iterations, refinements, and above all, practical cooperation between various users, such as domain experts, domain users, and ontology developers. This will be explained in the ensuing section.

2.3.1 Community Role in Ontology Development

As previously mentioned, the ontology development process involves different user types, depending on their role in the development cycle. Since the development of ontologies is a collaborative practice, collaboration communities will be defined based on their roles and tasks (see Figure 2.21).

On that account, Pinto et al. (2004b) has described five main steps to build ontologies:

1. **Specification**: Where the scope and purpose of the ontology are identified.

2. **Conceptualization**: Defining a conceptual model of the ontology, which consists of concepts, relations and properties that can occur in the domain. This step can include studying other existing ontologies that can be reused.

3. **Formalization**: A formal model is defined based on the previous conceptual model (e.g., by adding axioms that restrict its possible interpretations).
4. Implementation: The implementation of the formal model in a knowledge representation language, OWL.

5. Maintenance: This step includes constant evaluation and any updates to be implemented. The previous steps can be used for this.

Each stage requires certain skills and roles. For the Specification and Conceptualization, domain experts are mainly involved in these steps in order to ensure the proper knowledge and information is captured for the ontology. In Formalization and Implementation, ontology and knowledge engineers are the primary players because they can convert previously defined models into ontological forms. The last stage of Maintenance requires continuous interaction between all users (see Figure 2.22).

In summary, the roles necessary to construct ontologies comprise domain experts (also end-users), with domain knowledge and knowledge management and ontology developers (also knowledge engineers), who can formally structure and implement knowledge to be computationally processed and deployed.

In the ensuing section, multiple methodologies will be described based on collaborative support and user involvement.
2.3 Ontology Engineering

2.3.2 Methodologies

There are numerous methodologies and tools for constructing and developing ontologies with various user interfaces and environments. In this section, we describe some of the methodologies that have been used for building ontologies (see Figure 2.23).

One of the early development methodologies was based on the competency questions for defining the scope of the ontology, then extracting the main concepts of the ontology. This technique was conducted by Grüninger et al. (1995) where their focus was on developing a first-order logical model of the ontology. The methodology is centered on basic ontology development tasks, such as requirements analysis, conceptualization, implementation, evaluation, and maintenance. However, little attention was paid at the time to interact with domain specialists. Similarly, while Uschold and King’s method (Uschold et al., 1996) defines well-grounded steps for ontology creation, it still lacks good interaction with the end-users of the ontology.

In contrast, some methodologies adopt a collaborative strategy, where continuous coop-
eration between domain experts and ontology engineers is undertaken during ontology development. For example, METHONTOLOGY (López et al., 1999) and OnToKnowl-
edge (Sure et al., 2004) enable domain experts and ontology engineers to cooperate in
the development process, especially during the knowledge acquisition stage. Furthermore,
DILIGENT (Pinto et al., 2004a), NeOn (Suárez-Figueroa et al., 2012) and The Software
Ontology (SWO) (Copeland et al., 2012) are focused on the collaboration between ex-
erts and engineers, allowing the evolution of ontology throughout the iterations of the
development cycle.

Figure 2.23 illustrates the classification of various methodologies of ontology engineering.
This includes the features mentioned previously and some others, classifying them accord-
ing to their emphasis on collaboration with various users and their ability to reuse existing
ontologies.

![Figure 2.23: Ontology engineering methodologies.](image)

### 2.3.3 Tools and Editors

This section considers some of the many ontology development tools that facilitate some
or part of the ontology development process.
There exist various ontology development tools, both GUI or not, which facilitate some or part of the ontology development process.

### 2.3.3.1 Ontolingua

Ontolingua was the first ontology tool created at Stanford University (Farquhar et al., 1997). This tool is mainly focused on the collaboration between users to build and share ontologies (see Table 2.7). The researchers created the Ontolingua Server to ease the development of Ontolingua ontologies with a form-based web application. Creating a new ontology becomes more accessible by the flexibility of including parts of existing ontologies from a repository. This repository consists of a large number of ontologies from different fields. After completing a newly generated ontology, it can be added to a repository for possible reuse. This tool is mainly focused on the collaboration between users to build and share ontologies (see Table 2.7).

### 2.3.3.2 OilEd

OilEd is an ontology editor that has an easy-to-use frame interface, yet at the same time, allows users to exploit the full power of an expressive web ontology language (OIL) (Bechhofer et al., 2001). Classes are defined in terms of their superclasses and property restrictions, with additional axioms capturing different relationships such as disjointness. Although this tool has a graphical interface, it does not provide a collaboration facility for different users to collaborate in building or editing ontologies (see Table 2.7).

### 2.3.3.3 TopBraid Composer

Topbraid Composer[^29] is a graphical editor for the creation of concept taxonomies and relations. It combines the world’s leading semantic web modeling capabilities with the most comprehensive data conversion options and a powerful Integrated Development Environment (IDE) for implementing Knowledge Graph/Linked Data services. It is based on the Eclipse platform and the Jena API. It is a complete editor for RDF(S) and OWL mod-

[^29]: http://www.topbraidcomposer.com/
TopBraid Composer can also load and save any OWL2 file into formats such as RDF/XML or Turtle (Alatriste, 2012). It supports reasoning, query, and consistency checking mechanisms as well as enabling collaborative ontology construction.

2.3.3.4 SWOOP 2.3

SWOOP 2.3 is a browser and ontology editor that is one of the most frequently used (Malik et al., 2012). It is a dynamic, free, open-source, and systematic java-based editor based on Venn diagrams’ representation. In addition, it is primarily designed for OWL ontologies. Therefore, it often serves as the Web browser paradigm, meaning they must be concerned with URI. The browser’s features such as address bar, history buttons, navigation side bar, hypertextual navigation, and others are all supported for web ontologies, conforming to the mental model people have of URI-based online tools according to their existing Web browsers (Kalyanpur et al., 2006).

2.3.3.5 Protégé

The most popular software is Protégé, an open-source tool that provides a user interface to develop and construct ontologies of any domain. It has been widely used for developing ontologies due to the variety of plug-ins and frameworks it offers (N. F. Noy et al., 2003). Protégé provides an easy interface for the editing, visualization, and validation of ontologies and is a helpful tool for managing large ontologies (Horridge et al., 2011b). However, the desktop version of Protégé cannot enable collaborative editing involving multiple users. To address this missing feature, WebProtégé has been developed to enable collaborative editing using the web platform. WebProtégé uses Protégé infrastructure to support the collaboration on the backside and the Google Web Toolkit for the front end (Tudorache et al., 2013).

Historically, Protégé 3 was an ontology frame editor. OilED’s user interface was inspired by Protégé 3, but the underlying formalism was OWL (or more accurately, Oil; then, DAML+OWL). Later versions of Protégé 3 helped OWL by describing it with a meta-
frame ontology. Protégé 4 and WebProtégé were subsequently designed from the ground up to support OWL.

2.3.3.6 Tawny-OWL

Tawny-OWL has a complete programming language with a simple syntax which was originally modeled on the Manchester OWL notation (Horridge et al., 2012), modified to conform to standard Clojure syntax and to increase regularity (Lord, 2014). In addition, Tawny-OWL enables creating arbitrary patterns since it has access to the entire Clojure infrastructure. For example,

```clojure
(defn some-only [property & classes]
  (list (some property classes)
        (only property
         (or classes)))))
```

The `defn` declares a new function `some-only`, the arguments are `property` and `classes`, and the return values are packaged as a list by `list`. `some`, `only`, and `or` are defined by Tawny-OWL as the appropriate OWL class constructors. This enables the following definition of an existential relationship with a closure axiom:

```clojure
(defclass D 
  :super (some-only r A B))
```

This development environment is selected off the shelf from software engineering because it has a programmatic feature that enables programming the ontology and adding any functionalities that aid the development of the proposed methodologies.

Table 2.7 highlights other tools with different formalisms and a comparison summary based on the criteria of providing services of GUI and the collaboration support.
Table 2.7: Ontology editing tools.

<table>
<thead>
<tr>
<th>Tool</th>
<th>GUI</th>
<th>Collaboration support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ontolingua</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>OilEd</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>TopBraid Composer</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SWOOP</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Protégé</td>
<td>Yes</td>
<td>No (WebProtégé support the collaboration)</td>
</tr>
<tr>
<td>Tawny-OWL</td>
<td>No</td>
<td>No (Version control systems support collaboration)</td>
</tr>
</tbody>
</table>

2.4 User Interaction with Ontologies

2.4.1 User-centred Representation

In Section 2.2.2, the various representations of ontologies were described from machine perspectives. In this section, user-centered representation is shown to explore the possibilities of displaying ontologies to users for interacting and engaging with the development process.

2.4.1.1 Representing Ontologies Outside of an Editor

In the previous section, various ontology editors were described, with some facilitating the collaboration between different users, and some not. This section shows that ontology can be represented in different forms to read and explore outside the editing mode. There are three ways of interacting and representing ontologies outside an editor: these forms can be graphical such as in OWLViz\(^{30}\), navigational such as a web browser or QuickGO (Binns et al., 2009), and textual such as Tawny-OWL Manchester Syntax and DL.

**Graphical** (OWLViz\(^{31}\)): an ontology is represented as a directed graph where edges represent parent-child (is-a) relationships based on Graphviz\(^{32}\). Although this visualization is useful, sometimes it gets cluttered depends on the ontology size.

**Navigational** (Protégé, QuickGO): it has been illustrated in Section 2.3.3.5 that Protégé is a GUI, with a relevant screenshot shown in Section 2.3.3.5. Users navigate and explore

---

\(^{30}\)https://code.google.com/archive/p/co-ode-owl-plugins/wikis/OWLViz.wiki

\(^{31}\)http://protegewiki.stanford.edu/wiki/OWLViz

\(^{32}\)http://www.graphviz.org/
the ontology structure and information using different panels of the software. Additionally, the navigational service, QuickGO browser\textsuperscript{33}, provides users with access to all GO term information and the GO annotations. Moreover, users can view and search information about GO terms as well as protein data from UniProtKB\textsuperscript{34}.

**Textual** (Intermediate Representation, Manchester Syntax, Tawny-OWL, First order, or DLs syntax in papers): these representations all have a specific kind of user in mind: the domain user (IR), the ontology developer (MS), or the logician (DLs syntax). Additionally, they are adapted to their professional language, jargon, or notation of these domains. We note that they rarely adapt to the human language of the user, assuming American English as a standard. Moreover, the Pen&Pad project (Kirby et al., 1996) adopted the user-centered approach to designing and developing a medical workstation for daily inpatient care. It involves users in the development process and thus, produces a shared culture between themselves and the developers. The advantage of this is the resulting common language that can enhance understanding of some aspects of the problem.

Intermediate Representation (Rector et al., 1998), designed for domain specialists to help them read and edit ontologies, will be explained later in Section 2.5. In addition, representing ontologies as text always targets specific domain users. For example, FOL and DL are for logicians. In contrast, Manchester Syntax was developed for non-logicians, domain experts who do not have a DL background (Horridge et al., 2006), and was aimed at editing and representing all the aspects of an ontology. These user-friendly textual representations are adapted to the appropriate professional language, jargon, or notation of their domain users.

**Language Motivation**

As previously mentioned, targeting a human language has motivated some software developers to design programming languages using their mother language. For example, ARABLAN is an Arabic programming language that has been developed for use in teaching programming for school children in Arab countries (Al-A’Ali et al., 1995). Alterna-

\textsuperscript{33}http://www.ebi.ac.uk/QuickGO
\textsuperscript{34}https://www.uniprot.org/help/uniprotkb
tively, Oladosu et al. (2019) designed an African Native Language-based Programming Language to empower those only literate in their mother tongue for programming or learning to program in their language. Other international programming languages are also developed, such as Chinese, Dutch, French\(^{35}\). As such, internationalization of the software environment is the primary driver to convert the ontology development environment of Tawny-OWL into a non-English environment, such as Italian and Arabic. More details about this multilingual environment will be discussed in Chapter 4.

Moreover, the ontology can be represented in various languages through graphical, navigational, or textual styles, and this is based on the tooling used for the representation. For example, using text to represent ontology helps enhance the readability and improve the comprehension of ontologies. Another advantage of the text is to allow for complete documentation of ontologies, which is explored in the next section.

There are linguistic ontologies developed for multilingual purposes, such as EuroWordNet (Vossen, 1998) and BalkaNet (Tufis et al., n.d.), with several languages, having the same design as WordNet (Fellbaum, 1998; Miller, 1995).

In this research, the Arabic language is of focal interest. Regarding ontologies, there is the Arabic Ontology\(^{36}\), which is a formal representation of Arabic concepts with about 1,300 well-investigated concepts (Jarrar, 2011).

### 2.4.1.2 Ontology Documentation

Ontology documentation is an essential issue for understanding the ontology correctly and enabling sharing and reusing ontologies. There have been many efforts to record and present ontology documentation in predominately human-readable formats. There are three main concerns about the documentation of ontologies: 1) representing the content of ontology in a human-readable format, as the content of ontology may include complex information like metadata, the definition of classes and properties, visualization, and sometimes, versions history; 2) creating machine-readable annotations of documentation

\(^{35}\)https://en.wikipedia.org/wiki/Non-English-based_programming_languages

\(^{36}\)https://ontology.birzeit.edu/concept/293198
metadata; 3) making the documentation files accessible to read as a web resource (Garijo, 2017).

There exist a wide variety of documentation tools to extract and document the ontology. For example, OWLDoc is a Protégé plugin\(^{37}\) inspired by JavaDoc that generates HTML documentation (CO-ODE OWL Plugins, 2016). OWLDoc also offers two options: export to HTML and a dynamic view to browse from within Protégé (ibid.). OWLDoc aims to provide a browsable and shareable, but not editable, experience of the ontology inside Protégé or a standalone web browser (see Figure 2.24). While this works well, if errors are discovered, the user must move into a different environment to fix them.

![Figure 2.24: An OWLDoc representation inside Protégé environment.](image)

Importantly, Parrot is The Rule Interchange Format (RIF) and OWL documentation service. RIF is a standard for exchanging rules among rule systems, in particular, among Web rule engines (Alonso et al., 2015). Parrot generates documentation with a table of contents, summaries, and vocabulary. It can be used directly from the website\(^{38}\), or as NeOn Toolkit plugin (Alobaid, 2015).


\(^{38}\) [https://labs.mondeca.com/parrot/parrot](https://labs.mondeca.com/parrot/parrot)
Further, Live OWL Documentation Environment (LODE) is an open-source web service used to generate human-readable documentation of OWL ontologies. LODE automatically extracts classes, object properties, data properties, named individuals, annotation properties, general axioms, and namespace declarations from an OWL and OWL2 ontology. It represents them as ordered lists, together with their textual definitions, in a human-readable HTML page designed for browsing and navigation using embedded links (Peroni et al., 2012).

Following the narrative, WIzard for DOCumenting Ontologies (WIDOCO), is a software wizard that helps users document their ontologies. Starting with the ontology file or URL, WIDOCO guides the user through a wizard that generates a customized full HTML documentation for the ontology (Garijo, 2017). WIDOCO builds on top of LODE, and it extends LODE by expanding the properties a user may use to qualify a term in the ontology (ibid.)\(^{39}\). It has been used to document over a hundred ontologies across multiple domains. WIDOCO enables users to update their documentation with new text or diagrams, as the software supports Markdown, a markup language that is generally easier to edit than HTML. Moreover, it generates separated individual HTML files for each section in the documentation, which can be edited individually. Any modifications or additions will be integrated and the documentation updated consistently.

All of the tools mentioned result in an HTML-driven, navigable representation of the ontology. The user can interact, allowing them to understand the structure and individual classes described by the ontology.

### 2.4.1.3 Literate Ontologies

While many documentation tools use an ontology as it stands, an alternative approach enriches the ontology with additional information, resulting in a richer documented form. This is based on the literate programming approach, which will be described in Chapter 3. Even though ontologies have their description and annotation mechanism, literate ontologies have complete documentation. Thus, literate ontologies are those which incorporate

\(^{39}\) [http://dgarijo.github.io/Widoco/](http://dgarijo.github.io/Widoco/)
their full documentation text using the literate programming approach. Our work on Literate ontologies is an extension of the work done by Lord et al. (2015b), who have developed two literate ontologies: the Amino Acid Ontology and the Karyotype Ontology (see Figure 2.25). To accomplish this, they used LaTeX or AsciiDoc\textsuperscript{40} to represent these literate ontologies.

As Tawny-OWL is used for developing ontologies, the commentary is freely inserted with the source code; markup language is also used to format the commentary as we desired. This enables a formatted representation of the ontology documentation with headings, lists, etc. after conversion.

Literate ontologies can be represented in various forms, using the numerous techniques for converting the markup text into different formalisms or web pages, for example. Representing the ontology as an HTML web page gives users the ability to navigate and browse the documentation, either in order (i.e., section by section) or with a navigation facility (i.e., jumping between sections). It is also possible to hide or expose the ‘source’ sections, leaving the reader to see just the documentation as appropriate. From the developer’s perspective, while the reader may still not be able to see the axiomatization, in this way, the comments that they have checked are embedded directly next to the code, which is an interpretation of it.

**Wordifying Ontologies**

It is interesting to enable specialists to read and navigate through the ontology and its documentation. However, with HTML, there are no editing facilities to modify and update the ontology. It is, of course, still possible to read and edit the source code, which includes all of the documentation. Although logicians and software engineers may be used to interacting with knowledge in this form, this environment is unfamiliar to most domain specialists or biologists.

Therefore, rather than using HTML, we have also investigated the possibility to Wordify

\textsuperscript{40}AsciiDoc is a lightweight markup text document for writing notes, documentation, articles, books, ebooks, etc. (see \url{http://asciidoc.org/})
2.5 Domain Specialization

In the previously mentioned co-design approach in Section 2.3.2, the domain experts’ involvement is carried out through a series of workshops during the ontology development lifecycle. Whereas in Zeginis et al. (2014), the researchers interacted with users through brainstorming, interviews, and completing questionnaires. These interactions require meeting with experts/users, either to input, discuss, or give/get feedback, which can be time-consuming, especially if the experts are distant from each other.

This process is similar to the analysis requirements in software engineering. Recently, we have seen the rise of Behaviour-Driven development and an associated tool such as Cucumber, which allows semi-structured statements (see Figure 2.26b) to define requirements that can be tested computationally, as part of the software testing lifecycle (Wynne et al., 2014).

41This is fully covered in Chapter 6.
42https://cucumber.io/docs/cucumber/api/
Furthermore, the use of alternative and domain-user-friendly syntaxes has been investigated previously for ontology development. For example, the Galen Intermediate Representation (Rector et al., 1998) was developed in order to produce semi-structured text representation (see Figure 2.26a), which was created for domain specialists to read and edit before being corrected by knowledge engineers, who would fix their syntax and semantics.

An alternative to an intermediate representation is to use an ontology syntax, which is designed to be readable to knowledge engineers such as Manchester Syntax. The goal was to edit and represent all aspects of an ontology, including extra-logical parts that are critical for understanding the ontology in the context of its domain (Horridge et al., 2006). Unlike another syntax, it is frame-based, grouping statements about a single domain concept, rather than axiom-based, where statements are essentially unordered.

These techniques were built to enhance the readability of computational artifacts (e.g., ontology model or software testing) to empower users’ interaction during the process of the development.
Several systems have used other applications and formats to drive ontology development. For example, Cellfie (O’Connor et al., 2010) is a Protégé plugin that can transform a spreadsheet into an OWL ontology, which can then be developed further. However, this is a one-off process and – once ingested, the data in the spreadsheet is converted into OWL with further updates impossible to make using the original spreadsheet formalism. Other tools, such as RightField (Wolstencroft et al., 2011) and Populous (Jupp et al., 2010) add ontological features to office documents by allowing the selection of spreadsheet cells from a controlled vocabulary. This is typically followed by export to OWL using Ontology Pre-Processor Language (OPPL) (Egaña et al., 2009) to express the patterns used in the transformation (Jupp et al., 2011).

Finally, a number of different syntaxes have been developed that allow instantiation of ontology patterns; for example, in OPPL (Ontology Pre-Processing Language) (Egaña et al., 2009), and Dead Simple OWL Design Patterns (DOSDP) (Osumi-Sutherland et al., 2017) where an abstract syntax is used for effectively editing the ontology.

2.6 Conclusions

Ontologies are used in many different areas. Like software development, they require collaboration between those with expertise in a domain and computational representation. Ontologies are most useful for highly complex knowledge domains such as biology and medicine, meaning that this collaboration is itself complex.

There have been many tools and methodologies that can be used for ontology development. In addition, a broad classification of uses has been introduced in this chapter, with an emphasis on their contribution to the collaborative process of ontology development. The various tools and methodologies discussed, apply or consider these uses to different degrees. As there are different users, there is a need to develop the ontology as knowledge engineers. Drawing inspiration from biologists, how can existing tools be incorporated in new ways to provide a more engaging user experience?

To that end, the work in this thesis is aimed squarely at domain specialists. Overall, the
research investigates new methods that will enable them to contribute their knowledge to ontology development without becoming ontology experts.
Chapter 3

Technologies Employed in the Thesis

Contents

3.1 Introduction .................................................. 80
3.2 Clojure ..................................................... 80
3.3 Tawny-OWL ................................................ 81
3.4 Emacs ....................................................... 82
3.5 Literate Programming Approach ................................. 84
3.6 the UNICs of ORgaNizers (Org-mode) .......................... 86
3.7 Pandoc ..................................................... 87
3.8 Microsoft Excel and Microsoft Word ........................... 87
3.9 Summary ................................................... 88
Abstract

The primary tool used to develop ontologies in this thesis is Tawny-OWL. In this chapter, Tawny-OWL is described along with its functionality together with other tools/technologies that are used in the thesis such as Clojure, Org-mode files, Pandoc, and Microsoft Office Excel/Word.

3.1 Introduction

Ontology construction is a challenging task, as previously described in Chapter 2, which involves multiple parties cooperating. Moreover, ontology development is not a one-off process; it is an ‘iterative routine’ where several iterations are conducted to reach the final product of the ontology (Bermejo, 2007).

On that note, Tawny-OWL is a textual interface with a fully programmatic development environment for OWL. It has been implemented in Clojure and runs on the Java Virtual Machine (JVM) (Lord, 2013b). Similar to Protégé, it wraps the OWL-API (Horridge et al., 2011a), which performs much of the actual work, including interaction with reasoners, serialization, and the like.

Since Tawny-OWL is implemented in Clojure, this chapter briefly overviews the Clojure programming language before discussing Tawny-OWL and its features. Following that, it describes the additional tools and approaches employed in ontology development.

3.2 Clojure

Clojure is a functional programming language with a Lisp dialect. Furthermore, any Clojure program runs on the JVM, Common Language Runtime (CLR), and JavaScript engines (Hickey, 2008). Since Clojure runs on JVM, it can access a wide range of Java libraries and use it for data processing, networking, presentation, and imaging (Tolpin et al., 2016), as well as having the ability to access Java frameworks, with optional type hints and type inference (Gupta, 2017). Like other Lisp languages, Clojure employs the code-as-data philosophy\(^1\), which makes it a dynamic and compiled language that maintains

\(^1\)This means that the code of the program is also treated as data that can be manipulated and executed by the program.
every feature at runtime.
In Clojure, Leiningen\(^2\) is used for project automation to help provide support for Maven integration, which handles project package management and dependencies. The open-source development of Clojure is a community-driven tool, which is led by its creator, Rich Hickey. It is a feature-rich programming language, has a wide set of immutable data structures, and provides a powerful macro facility. Some examples of the remarkable Clojure features are dynamic development, in addition to functional and concurrent programming. Other features can be found on the Clojure official website\(^3\).

A Clojure library, called Docjure is used for parsing the spreadsheet. Docjure is a library for reading and writing Microsoft Office documents within the Clojure program (Martin, Jul, 2018). Docjure has features such as evaluating a formula cell in a spreadsheet and providing a way for exposing a formula in a cell as a Clojure function. However, we did not need this feature; we only use the library for parsing the spreadsheet and retrieving cell information.

### 3.3 Tawny-OWL

Tawny-OWL has been described previously in Chapter 2. It is a fully programmatic development environment for OWL. Its syntax is based on the Manchester OWL notation, which has been adjusted to meet the requirements of standard Clojure syntax. For instance, a new class can be created with the following existential constraint:

```
(defclass A
   :super (some r B))
```

Or, a new individual can be defined with a property assertion:

\(^2\)https://leiningen.org/
\(^3\)https://clojure.org/
Tawny-OWL library, called Bubo, is used in this research to design ontology patterns for Microsoft Excel spreadsheets. Bubo is a batch library for running Tawny-OWL in command line. It reads CSV and XLSX files, then transforms the values into an ontology. It has been developed in cooperation between Phillip Lord and the author. For the spreadsheets, Bubo extracts the values from the spreadsheets and converts these into ontology’s objects according to the predefined patterns. Spreadsheets in this case must be integrated with the patterns in order to be extracted correctly. More detail about the predefined patterns and the use of Bubo is described in Chapter 5.

The following section discusses Emacs, the Integrated Development Environment (IDE) used when writing Tawny-OWL, and a description of its features.

### 3.4 Emacs

Emacs is a popular and portable open-source text editor with a long history. It is still actively maintained and extended by GNU’s Not Unix (GNU) Emacs and the community. Emacs is simply described in the (GNU) Emacs manual as "the extensible, customizable, self-documenting, real-time display editor"\(^4\). Further to just editing text files, Emacs has distinctive features that make it a popular IDE for many programming languages. These features include, but are not limited to the following:

- Content-aware editing modes, which perform syntax highlight, for various file types.
- Complete built-in documentation, including a tutorial for new users, as illustrated in Figure 3.1.
- Highly customizable abilities, using Emacs Lisp code or a graphical interface.

• Adaptation of a packaging system for downloading and installing extensions.

• A Graphical User Interface (GUI) implementation which supports different font sizes, graphics, line drawing, and displaying Portable Document Format (PDF).

• An entire ecosystem of functionality beyond text editing, including a project planner, mail and newsreader, debugger interface, calendar, and more.

• Unicode support for numerous human scripts.

Since Emacs Unicode support helps writing in many different languages and programming/-markup languages, this facilitates the work on Wordifying the ontologies and transforming the ontology development environment into Arabic, as will be explained later in the thesis.
3.5 Literate Programming Approach

The term literate programming was coined by Knuth (1992) in which both literature and a source code are considered in the program. The main idea of this paradigm is to insert text along with code, meaning the program will also be its own documentation. The intention here is that the program should become easier to understand and, conversely, the documentation is less likely to become out-of-date, as it is maintained in the same place. Dr. Knuth developed the WEB system to implement literate programming by inserting source code inside the explanatory text, rather than the standard practice in most programming languages, which is more understandable by human readers. WEB has two secondary programs: TANGLE, which produces compilable Pascal code from the source text, and WEAVE, which produces well-formatted, printable documentation using TeX. There is also the C programming language version of WEB, called CWEB (ibid.). A simple example is shown in the following script, where the explanation text appears before and after a source code for a factorial function.

\begin{code}
\text{This function calculates a factorial number.}

\text{\begin{verbatim}
fact 0 = 1
fact (n+1) = (n+1) * fact n
\end{verbatim}}
\end{code}

WEB has not taken off, probably because (a) it is complicated, (b) it is not compatible with many tools; it is limited to Pascal, C, and LaTeX.

However, some of the ideas behind literate programming are widely spread in computer programming languages, and they have been applied with various mechanisms. Adding comments to source code is a common practice in programming; however, the type and format of the comments play a significant role in generating a readable format. Consequently, nowadays, comments are used to serve different purposes; these are examples of
the use of comments:

1. Comments can be used interactively to help in creating the documentation of programming languages:

   Documentation strings are Self Documentation tool that have been implemented in many programming languages such as Lisp\textsuperscript{5}, Python\textsuperscript{6}, and R\textsuperscript{7}. A documentation string is a string that is written within the function declaration; it explains what a function or variable is for, what are the possible arguments of the function, and what are the returns, where to find more information, and so on (EmacsWiki, 2018).

2. Comments can be used to generate out text files: In Perl, perldoc looks up documentation (in .pod format) embedded in the Perl installation tree or a Perl script and displays it using various formats. This is mainly used for Perl library module documentation. For example, the following command\textsuperscript{8} instructs the system to save the output as \LaTeX format to the specified file name, and the (\texttt{Text::Wrap}) is for forming simple paragraphs:

   \begin{verbatim}
   perldoc -oLaTeX -dtextwrapdocs.tex Text::Wrap
   \end{verbatim}

3. Comments can be used to generate web files (HyperText Markup Language (HTML)) with tools such as JavaDoc or Doxygen. JavaDoc is a software documentation generator which automatically creates the documentation based on comments and annotations in the source code\textsuperscript{9}. In addition, Doxygen is a documentation generator that can be used for many programming languages. These include C / C++, Java, Objective-C, Python, IDL, Fortran, VHDL, PHP, and C# (Heesch, 2018).

\textsuperscript{5}https://www.emacswiki.org/emacs/DocString
\textsuperscript{6}https://www.pythonforbeginners.com/basics/python-docstrings
\textsuperscript{7}https://cran.r-project.org/web/packages/docstring/vignettes/docstring_intro.html
\textsuperscript{8}https://perldoc.perl.org/5.30.0/perldoc.html
\textsuperscript{9}https://www.oracle.com/technetwork/java/javase/tech/index-137868.html
4. Comments can be used to define tests:

In Rust, documentation comments are written in Markdown and support code blocks. Rust continuously checks the correctness of these blocks, so they are compiled and used as tests\(^\text{10}\). Python, like many other programming languages, has this feature\(^\text{11}\).

A summary of the tools that use several features of the comments in source code is shown in Table 3.1.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Core Language</th>
<th>use with other languages</th>
<th>Documentation output format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docstring</td>
<td>Lisp</td>
<td>Yes</td>
<td>Emacs</td>
</tr>
<tr>
<td>perl燃油</td>
<td>Perl</td>
<td>No</td>
<td>HTML and latex</td>
</tr>
<tr>
<td>JavaDoc</td>
<td>Java</td>
<td>No</td>
<td>HTML</td>
</tr>
<tr>
<td>Doxygen</td>
<td>C++</td>
<td>Yes</td>
<td>HTML and more other formats</td>
</tr>
<tr>
<td>Rust</td>
<td>Rust</td>
<td>No</td>
<td>test outputs of the Rust program</td>
</tr>
</tbody>
</table>

### 3.6 Org-mode

As described earlier, one of the Emacs features is the content-aware editing modes. A mode in Emacs is a collection of behavior related to a particular content style. An Emacs Org-mode is a note-taking and task management software package using plain text files. It uses a markup language to format the text like italic/bold font styles and links. Org files can be exported to different file formats such as HTML, \LaTeX, Open Document, and Markdown.

In this research, Org-mode files were used to write and format the comments in Tawny-OWL file; afterwards, those Org files are converted to Word documents using the Pandoc tool, which will be described next.

\(^{10}\)https://doc.rust-lang.org/rust-by-example/testing/doc_testing.html

\(^{11}\)https://docs.python.org/3/library/doctest.html


3.7 Pandoc

Pandoc is free software that converts files from one markup format into another (MacFarlane, 2013). Pandoc has a Haskell library and has no GUI; it can be implemented using a standalone command-line tool. The following example is a command used in Pandoc to convert a text file to an HTML file:

```
pandoc -o output.html input.txt
```

It is a fairly straightforward command, which only needs to specify the output file with the file extension. Therefore, in the example above, a text file will be converted into an HTML. Similarly, in order to convert the text file into another format, the extension required must be used (i.e., `.docx` for word document, `.tex` for \LaTeX, and `.pdf` for Portable Document Format). There are other options to use, but that is the simplest conversion command.

Pandoc can convert multiple markup and word processing formats; this includes, but is not limited to, various flavors of Markdown, HTML, \LaTeX, and Word docs. Moreover, Pandoc can understand useful markdown syntax extensions, such as document metadata (title, author, date), footnotes, tables, and definition lists. In addition, it includes an effective mechanism for automatic citations and bibliographies.

3.8 Microsoft Excel and Microsoft Word

Microsoft documents are widely used in science for organizing and/or structuring information. In biology and medicine, the use of an excel spreadsheet is popular. It is used to structure (sensitive) information and enables statistical analysis using spreadsheet facilities (Buckingham et al., 2014). Word documents are common too, especially for documentation; scientists and non-scientists have widely used this word processor. The track change feature is an excellent addition to Microsoft Word, as it enables users to share documents and keep track of their revisions. This is why this research examines how these
prevalent tools can build ontologies with domain experts collaboratively.

3.9 Summary

Ontologies are hard to construct—such to program. Tawny-OWL redefines ontology construction as a form of programming. Using Tawny-OWL's facilities and incorporating other technologies helps to generate a novel visualization of ontologies as Word documents. This chapter showed various technologies, all of which played a vital role in the ontology creation process, using proposed approaches. The next chapter demonstrates the translation of the programming environment of Tawny-OWL into Arabic.
Chapter 4

Arabic Ontologies

Contents

4.1 Introduction ...................................................... 90
4.2 The Arabic language ........................................... 91
4.3 Identifier Types in Tawny-OWL .............................. 92
4.4 Tawny-OWL Polyglot Library ............................... 93
4.5 An Internationalized Source Code Environment ............. 94
4.6 Arabic Language Challenges ................................. 95
  4.6.1 Text Direction ........................................... 97
  4.6.2 Cursor Movement with Text ............................... 97
  4.6.3 Integrated Development Environments (IDEs) ............. 99
4.6.4 Alphabetic Features ...................................... 104
4.7 Conclusions .................................................... 105
Abstract

Internationalization technologies are endemic and enable applications to support multiple languages, which fulfill the preference of users to use their native language. This chapter shows how the textual environment of Tawny-OWL can allow full internationalization while developing ontologies. The Arabic language is a challenging script, due to the divergent features and grammar, compared to the English that Tawny-OWL was originally developed to support.

4.1 Introduction

One of the main preferences of people with applications or programs is to use their native language, so that they can better understand various commands/functions. This is why most applications nowadays support multiple languages to reach and support a broader range of users around the globe by facilitating a multilingual environment.

For an ontology development environment to support a wide variety of users, including domain specialists, internationalization has an obvious advantage. Therefore, the textual programmatic environment of Tawny-OWL was used to translate its commands into Arabic. This thesis extends that described in Tawny-OWL Github repository\textsuperscript{1}, which uses the Italian language in Tawny-OWL.

The first and most obvious advantage of this is to improve the readability of an ontology which enables users to interact with the ontology, using their native language. Internationalization technologies are widespread, and it is possible to enable support for multiple languages for applications with a graphical user interface. In the Figure 4.1 below, the languages could be used to initiate the interaction. Domain experts and ontologists, according to the current study, can create an ontology in Arabic. Tawny-OWL can provide fully translated ontology commands. Protégé, on the other hand, can display Arabic text for exploring ontology components.

This chapter describes the use of multilingual programming environments to facilitate the use of Arabic to construct ontologies. The RQ1 is addressed in this work to demonstrate how to present the ontology in Arabic. The author starts with the existing Tawny-OWL

\textsuperscript{1}https://github.com/phillord/tawny-owl
polyglot library; then explains how to enable support for Arabic in the textual user interface of Tawny-OWL, giving the ontologists the ability to use their native language for all parts of the development process. Finally, the difficulties of using the Arabic language, which differs from English in many ways, are discussed, as well as strategies for overcoming them.

4.2 The Arabic language

First, general features of the Arabic language are described because of the relevance to the work in this chapter. The Arabic language is one of the Semitic languages with over 300 million speakers in the world, which makes it the 5th most spoken language (Cactus, 2015; Sayed, 2015). It is also one of the six official languages of the United Nations (United Nations, n.d.). It has 28 basic letters in its alphabet; although some books say 29 letters by adding the letter *hamza*\(^2\) as in Awde (1986), or in Nasr (1967) by adding *laam-alif*\(^3\).

Arabic has no upper nor lower case forms for its letters; instead, all letters change their shapes depending on their position in words (i.e., beginning, middle, or end) (ibid.).

\(^2\)Sometimes, it is considered as a diacritical symbol

\(^3\)A combination of the two letters: *laam* and *alif* in one letter
over, letters in Arabic are either connectors or non-connectors. Most letters are connectors which means that they must connect to the adjacent letter in the word; unlike the non-connectors, which cannot be connected with the adjacent letter. Therefore, cursive writing style (Habash, 2010) is vital in the Arabic language. Although Arabic text direction is right-to-left, numbers are not; they follow the universal style of left-to-right order.

### 4.3 Identifier Types in Tawny-OWL

This section describes how Tawny-OWL uses identifiers and how these identifiers relate to each other. There are three identifier spaces: the Internationalized Resource Identifier (IRI) (or IRI fragment), the label, and the Tawny Name.

**IRI:** This is the primary identifier used in OWL for entities and annotations. While this is not required, most ontologies use a single base IRI, shared between all entity IRIs, which differ in their fragment.

**Label:** This is a human-readable name for an entity identified by an IRI. In Web Ontology Language (OWL), this is an annotation; there can be multiple labels for any IRI and, indeed, multiple types of labels.

**Tawny Name:** This is the programmatic identifier that is used within Tawny-OWL. By default, it is also serialized into OWL as an annotation property.

By default, the IRI fragment and the Tawny Name are the same. In Protégé, only the IRI fragment, and the label are displayed\(^4\). Typically, Protégé displays the label, if there is one, or the IRI fragment if not. However, it can be configured to disable any annotation or other property of an entity. All of these are internationalized already. IRIs are internationalized (since they became IRIs rather than Uniform Resource Identifiers (URIs)). Labels support full Unicode and can also be annotated with language. The Tawny Name also can be Unicode Transformation Format which uses 8-bit blocks to represent a character (UTF-8).

\(^4\)The Tawny Name can, but need not, be stored as an annotation, which is displayed in Protégé also.

- 92 -
However, there are a few Clojure limitations, the most obvious being no spaces. Therefore, in Tawny-OWL Arabic, terms can be used in the ontology as identifiers in any of the three aforementioned types.

In Protégé, Arabic identifiers can be used as IRI and labels. In Figure 4.2, a screenshot of Protégé is displaying an Arabic ontology with Arabic class names. Next, I explain how the Tawny-OWL polyglot library was used to incorporate term translations into the ontology.

![Figure 4.2: Arabic version of the Family ontology in Protégé.](image)

### 4.4 Tawny-OWL Polyglot Library

The Tawny-OWL polyglot library (Lord, 2013b) uses a fairly standard technique for internationalization of programmatic code; the ontology is developed with a set of programmatic labels, which are then referenced in a language, or locale bundle with an appropriate translation. In the case of Tawny-OWL, this translation appears as `rdfs:label` annotations on the ontology entities (classes, properties, etc.). This overall process is shown in Figure 4.3, placing the Italian and Arabic language translations onto the pizza ontology.

The next step is to use Arabic for the identifiers in Tawny-OWL, as in the following script\(^5\).

\(^5\)The code in here is not syntax highlighted, unlike most of the thesis. This is because \LaTeX\, package...
Here, any language could be used in specifying the name of the classes/properties. This is also applied with any other environment such as Protégé, where Arabic can label the identifiers. This is shown in Figure 4.4 where a new class has been added to the ontology using an Arabic term. IRI can also be written using Arabic terms or any other language as it has been already internationalized with all languages.

```lisp
(defclass نكهة_البيتزا :annotation
  (label "Pizza Base" "ar"))
```

Although this enables the usage of native language terminologies for the end-users of the ontology, it does not change the English-centric editing environment of the program.

### 4.5 An Internationalized Source Code Environment

The aim is to internationalize the entire source code of the ontology. Internationalization will make the ontology more comprehensible and readable for all developers who communicate using Arabic in this case. In other words, same as translating the identifiers used within the ontology, the keywords and identifiers used by Tawny-OWL are also intended to be translated. This is fully supported with a total conversion of the environment using the multilingual feature of Tawny-OWL, as in Figure 4.5, which shows the English, Italian, and Arabic version of Pizza ontology (Lord, 2012). This is relatively easily implemented struggles with mixed direction text, demonstrating the problems described in this chapter.
with Tawny-OWL— the Arabic or Italian keywords are simply aliases defined in a different namespace, pointing to the equivalent functions in the main English namespace.

In this case, there is no trace of English left in the non-English versions\(^6\). The latter of these is a right-to-left Arabic alphabet, and I use the Emacs editor to change the direction that Tawny-OWL code is rendered in Figure 4.6b. This demonstrates the capability of Tawny-OWL to adapt to any language.

### 4.6 Arabic Language Challenges

Although it is possible to internationalize Tawny-OWL in theory, there are many challenges and difficulties with achieving this in practice. In this section, these challenges are described.

\(^6\)In the complete source code, there is some English – namely the statements at the start of the file which import the appropriate language environment; this would be possible, but not easy to remove.
Chapter 4  4.6 Arabic Language Challenges

(a) English Pizza Ontology
(defontology pizzaontology
  :iri "http://www.ncl.ac.uk/pizza"
  :prefix "piz:"
  :comment "An example ontology modelled on the Pizza tutorial ontology from Manchester University, written using the tawny-owl library"
  :version-info "Unreleased Version"
  :seealso "Manchester Version")

(defaproperty myOpinion
  :subproperty owl-comment-property
  :label "My Opinion"
  :comment "Do I think this is a good pizza to eat?"
)

(defclass Pizza
  :label "Pizza")

(b) Italian Pizza Ontology
(defontology pizzaontologia
  :iri "http://www.ncl.ac.uk/pizza"
  :prefix "pizza"
  :comment "Un esempio ontologico modellato su un tutorial di ontologia sulla Pizza dall'Università di Manchester, scritta utilizzando la libreria Tawny-OWL"
  :version "Versione non pubblicata"
  :vedianche "versione di Manchester")

(defproprietà miaOpinione
  :etichetta "La mia opinione"
  :comment "Penso che sia una buona pizza da mangiare?")

(defclasse Pizza
  :etichetta "pizza")

(c) Arabic Pizza Ontology

Figure 4.5: Multilingual Pizza ontology
4.6 Arabic Language Challenges

4.6.1 Text Direction

The most significant challenge when using Arabic is the text direction. Since Arabic has a right-to-left text, it creates other sub-problems to deal with: not only the identifiers or names, but also all other symbols and characters such as numbers, parentheses, and colons, as they play a crucial role in the programming process to develop the ontology.

The first step in dealing with that is to flip the whole editor environment direction. In this case, Emacs is used, which enables it, as shown in Figure 4.6; the text can be optionally shown with right or left-alignment within the window.

(a) Arabic Pizza Ontology in Emacs Displayed RTL.
(b) Arabic Pizza Ontology in Emacs Displayed LTR.

Figure 4.6: Emacs display of Arabic text in two directions.

Usually, on an Arabic installed operating system, tree structures most commonly seen in file browsers are shown on the screen’s right, with the hierarchy and text right-aligned. However, in Protégé, the most popular tool, it is only partially possible to change the whole environment layout to the opposite. The display of the class hierarchy is largely correct; however, because of the right-to-left script of Arabic, the label for a class is displayed some distance away from the class icon, as illustrated in Figure 4.2. It is not possible to enable this in Protégé, to our knowledge.

4.6.2 Cursor Movement with Text

One difficulty with Arabic text embedded in English is that the cursor movement is predictable but erratic. For example, image a string of 9 characters, with the middle three
in Arabic (see Figure 4.7). From the beginning to the end of this string, the cursor would
start at the left of the first English block and move to the right for the first three char-
acters. Then, it will jump to the right of the Arabic block and move left over the next
three characters. Afterwards, it will jump back to the left of the second English block
and move right for the last three characters. The sequence of movement is as follows:
$L1, L2, L3, R6, R5, R4, L7, L8, L9$ where L is a left-to-right character and R is right-to-left
class character, and the number indicates successive cursor positions. This is illustrated in Fig-
ure 4.7.

![Figure 4.7: Cursor movement with mixed text of Arabic and English characters.](image)

Once translated in Arabic, Tawny-OWL mostly overcomes this problem, but with a few
limitations. As a Lisp, Clojure (and this Tawny-OWL) has minimal syntax, most of which
is textual. However, punctuation characters are used and are defined in syntax, notably
", \(, \), and \. Full Clojure syntax also uses [ ], \ and # but these are rarely used in
Tawny-OWL. Finally, \ is defined as a white-space alignment character, and \ introduces
a comment. This textual feature affects cursor motion; essentially, it changes according
to the text direction. In Figure 4.8, the impact of this is illustrated. The cursor changes
direction and jumps at Clojure syntax \ and \. In addition, the http protocol also causes
a jump.

This also shows the advantage of translating the whole environment into Arabic. Displaying
the full document right-aligned can be selected, with cursor motion being predictable and
stable. This is something not authentic with mixed English/Arabic documents. Tawny-
OWL nearly achieves this, but not quite.
4.6.3 Integrated Development Environments (IDEs)

Although Emacs is well-used within the Clojure developer community, it is not the only Integrated Development Environment (IDE) available. There are existing tools that support this, with each one having their own techniques. Hence, the author investigated some other IDEs that support Clojure in order to test the behavior and performance of Arabic source code. The three main features to explore are ’displaying the Arabic text’, ’changing text direction’, and ’cursor behavior with Arabic text’.

1. **Eclipse**

Eclipse is one of the most popular IDEs. Support for Clojure comes from the plugin CounterClockWise\(^7\). Many Clojure features are supported using this plugin in Eclipse, including Leiningen builds and code evaluation (Elizabeth, 2018). The option to change text direction in Eclipse can be found in the window preferences settings menu. There are two options: graphical layout and text direction. The graphical right-to-left layout seems promising. However, it is unusable for Arabic text. As well as changing the layout, it uses a mirror-image of the text symbols also; (\(\text{د}١\text{غ}١\text{ذ}١\text{ی}١\text{ظ}١\text{ل}١\text{م}\) \(١\text{ی}١\text{ذ}١\text{غ}١\text{ی}١\text{ظ}١\text{ل}١\text{م}\)). Figure 4.9 illustrates this case in Eclipse.

\(^7\)https://github.com/ccw-ide/ccw
The cursor movement is unpredictable because it does not display directly on the text displayed in the cursor position; it always displays somewhat away from the actual text. This makes it challenging to navigate and update the script (see Figure 4.10 for a screenshot of the Eclipse layout).

Figure 4.9: Eclipse screen shot of the Arabic Pizza ontology using the graphical right-to-left layout.

2. Cursive

Cursive IDE\textsuperscript{8} for Clojure is built on IntelliJ. The Clojure extensions for IntelliJ are implemented in Clojure and have wide support for clojure.test, a debugger, and formatting code including Paredit (Elizabeth, 2018).

The text direction feature is a command in the 'View' menu with three options: contextual, left-to-right, and right-to-left. The cursor movement is correct when moving with the arrow keys; the cursor changes shape between Arabic words and English letters or symbols (see Figure 4.11). However, the graphical space does not

\textsuperscript{8}https://cursive-ide.com/
Figure 4.10: Eclipse screen shot of the Arabic Pizza ontology.
convert the direction of the text – the letters of each word are displayed sdawkacb. Nonetheless, only the text alignment does layout from right-to-left.

Figure 4.11: IntelliJ screen of Arabic Pizza ontology.

3. Atom

Atom\(^9\) is a modern desktop text editor that is open source. It is designed to be easy and simple for beginners and new programmers\(^10\). The package Proto Read-Eval-Print Loop (REPL) is intended for Clojure developers, offering many features including autocompletion. Evaluation can be performed easily by clicking a button, and results will be displayed in the REPL or online. In addition, the automatic evaluation mode follows along with writing the code (Elizabeth, 2018).

When displaying Arabic text in Atom, there is no direct way to change the text direction to right-to-left. There are some overlapping letters and spaces when the Arabic text is displayed, and the cursor cannot move over Arabic characters. It

\(^9\)https://github.com/atom
only moves over the English and any other symbols (see Figure 4.12). Even clicking on the Arabic text, the cursor does not appear in the correct place. In this sense, the performance of Atom in the mixed text is similar, but slightly less usable than Eclipse.

![Atom screen of Arabic Pizza ontology.](image)

Figure 4.12: Atom screen of Arabic Pizza ontology.

4. **Nightcode**

This is a smart editor that packages Leiningen\(^\text{12}\) and Boot build tools. It provides quick access to the Clojure cheatsheet and REPL in its home screen. Additionally, it supports the highlighting features of the code (ibid.).

The standard build of this editor does not support all Unicode letters, and in particular, cannot display Arabic. As a result, when the Arabic Pizza ontology is opened, it cannot display the letters, as shown in Figure 4.13. This is clearly unusable for editing.

\(^{11}\)https://github.com/oakes/Nightcode
\(^{12}\)https://leiningen.org/
4.6 Arabic Language Challenges

Figure 4.13: NightCode screen of Arabic Pizza ontology.

Table 4.1: Comparison of IDEs in dealing with Arabic script.

<table>
<thead>
<tr>
<th>IDE</th>
<th>Displaying Arabic text</th>
<th>R-to-L direction</th>
<th>Cursor movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emacs</td>
<td>Yes</td>
<td>Bidirectional text alignment is possible</td>
<td>Useful and easy to navigate</td>
</tr>
<tr>
<td>Eclipse</td>
<td>Yes</td>
<td>2 options: text alignment and window layout</td>
<td>Not useful and difficult to navigate the text</td>
</tr>
<tr>
<td>Cursive</td>
<td>Yes</td>
<td>Text alignment only</td>
<td>Useful and easy to navigate (change cursor head on text)</td>
</tr>
<tr>
<td>Atom</td>
<td>Yes (overlapping letters)</td>
<td>No R-to-L option</td>
<td>Not useful and difficult to navigate</td>
</tr>
<tr>
<td>Nightcode</td>
<td>No</td>
<td>No R-to-L option</td>
<td>Not useful</td>
</tr>
</tbody>
</table>

4.6.4 Alphabetic Features

As mentioned earlier in Section 4.2, most letters in the Arabic language are connectors. Therefore, when translating the ontology construction commands, a separator is used between the words to maintain the readability of the script. For example, 'عرف الأنتولوجيا'
4.7 Conclusions

and 'عرف_الخصائص_الكراني' correspond to 'defontology' and 'defaproperty'. In the English language, words without spaces can be readable and meaningful in terms of programming commands. Table 4.2 shows more examples of the Arabic commands and the equivalent English ones.

The acronym IRI is transliterated as it sounds using Arabic alphabets separated by dashes to avoid the connectivity between Arabic alphabets.

Table 4.2: Tawny-OWL commands and their equivalent in Arabic.

<table>
<thead>
<tr>
<th>Function</th>
<th>Tawny-OWL command</th>
<th>Arabic command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define a new ontology</td>
<td>defontology</td>
<td>عرف_الإنتولوجيا</td>
</tr>
<tr>
<td>Define a new class</td>
<td>defclass</td>
<td>عرف_الصنف</td>
</tr>
<tr>
<td>Define a new individual</td>
<td>defindividual</td>
<td>عرف_المثال</td>
</tr>
<tr>
<td>Define an object property</td>
<td>defoproperty</td>
<td>عرف_الخاصية_الكائن</td>
</tr>
<tr>
<td>Define a data property</td>
<td>defaproperty</td>
<td>عرف_الخاصية_البيانات</td>
</tr>
</tbody>
</table>

4.7 Conclusions

Multilingual environments are advantageous for being readable and comprehensible by users when employing their language. This chapter has shown that it is possible to translate Tawny-OWL to support an alternative language. For a European language, such as Italian, this is straight-forward and works well. However, for a right-to-left language, such as Arabic, users also dependent on the IDE working correctly. The experiences here are mixed, with some IDEs working well, and some unusable. This is perhaps not surprising; as software engineering is biased toward English. There are only a few exceptions that use other scripts such as قلب_13 which is a Scheme influenced language implemented in Arabic. Despite this, it is, however, clearly possible to edit a right-to-left Tawny-OWL file with a subset of Clojure editors.

This still leaves us in a programming environment, an environment unlikely to be familiar or comfortable to most domain users. Moreover, the ontology lacks a narrative structure;

13https://qlb-repl.herokuapp.com
meaning it cannot be read in a literate fashion, and how to enable this is considered in the next chapter.
Chapter 5
Ontology Development Excel Workflow (ODEW)

Contents

5.1 Introduction ................................................. 108
5.2 A Motivating Example ................................. 110
  5.2.1 The tolerogenic antigen-presenting cells (tolAPC) Catalog 111
  5.2.2 Initial ODEW for the tolAPC Catalog ............ 112
5.3 The tolAPC Ontology ....................................... 114
  5.3.1 Building the tolAPC Ontology .................. 114
  5.3.2 Modeling in the tolAPC Ontology .............. 116
5.4 Generalization of ODEW .............................. 119
  5.4.1 The Patterns in Bubo ......................... 119
  5.4.2 Bubo Validation Using a Musical Spreadsheet 123
5.5 Discussion ................................................. 124
5.6 Conclusions .................................................. 129

This chapter is modified from Blfgeh et al. (2017b)
Chapter 5

5.1 Introduction

Abstract

Large amounts of knowledge in biology are captured in Microsoft Excel spreadsheets. These are easy to build, maintain, and develop collaboratively. This chapter will discuss how to incorporate this type of knowledge into a document-centric approach for developing ontologies. To begin, the motivation for this study is the use of the tolAPC catalog, a spreadsheet that contains information about immunological cell lines, and is created and maintained by a group of biologists. In the first place, the tolAPC catalog, a spreadsheet that describes immunological cell lines created and maintained by a group of biologists, served as inspiration. The tolAPC Ontology was generated by using this catalog, and the patterns’ design will be demonstrated using Tawny-OWL, followed by the extraction of information from the spreadsheet to generate axioms. This process means that Tawny-OWL includes the spreadsheet as part of the ontology source code, which can be freely updated if required, and the ontology can be regenerated accordingly. In this manner, the spreadsheet provides a bulk of data, with the ontology environment providing a richer and more precise axiomatization.

5.1 Introduction

Chapter 4 previously showed how to bring ontology development closer to some users through the internationalization of ontology development, specifically into Arabic. While this is useful for scientific ontology development, internationalization is perhaps less of a problem, as most texts are in English. However, there is a different problem, namely that of tooling. Biologists and Medics use a toolset that is different from the standard development tools of ontologists.

Chapter 2 discussed several approaches to incorporate documents in the ontology engineering process in order to facilitate the gap in tooling between biologists and ontologists. There have been several responses to this problem. First, it is possible to take existing ontology tools and customize them for use within a specific community so that they have a familiar look and feel. This is the approach taken by Collaborative Authoring Tool (iCAT) – a version of WebProtégé, built explicitly for the International Classification of Disease version 11 (ICD-11) community (Tudorache et al., 2011).

A second approach is to enable existing ontological tools to ingest office documents. As previously mentioned in Section 2.5, for example, Cellfie (O’Connor et al., 2010) is a Protégé

\[\text{https://webprotege.stanford.edu/}\]
plugin that can transform a spreadsheet into a Web Ontology Language (OWL) ontology, which can then be developed further. However, this is a one-off process – once ingested, the data in the spreadsheet is converted into OWL; further updates cannot be made using the original spreadsheet formalism. Subsequently, tools such as RightField (Wolstencroft et al., 2011) and Populous (Jupp et al., 2010) add ontological features to office documents by allowing selection of spreadsheet cells from a controlled vocabulary, followed by export to OWL, using Ontology Pre-Processor Language (OPPL) (Egaña et al., 2009) to express the patterns used in the transformation (Jupp et al., 2011). Lastly, ROBOT is an OBO Tool (ROBOT) (Overton et al., 2015), which is a command-line tool that can deal with Comma Separated Values (CSV) format, only to extract information for the ontology.\(^2\)

These tools, however much they support the use of Office software, at some point, require leaving the software and moving into an ontology-specific environment. Therefore, the new Ontology Development Excel Workflow (ODEW) developed, using the highly-programmatic environment for ontology development, Tawny-OWL. In this approach, the Microsoft Excel values are extracted, and the ontology is developed as programmatic source code, which is then evaluated to generate the final ontology, either in memory or as an OWL file. With Tawny-OWL, spreadsheets can also directly become a part of the source code\(^3\); therefore offering a new methodology that allows the biologists to generate and maintain their dataset in an unconstrained Microsoft Excel spreadsheet. In this model, arbitrary validation and transformation of the data held in the spreadsheet can be applied into an ontological form. As the spreadsheet is part of the source code, rather than being used as a knowledge capture interface, it can be freely updated and the final ontology regenerated. In Table 5.3, a comparison between those tools is shown using criteria centered on user activities of editing and reading the documents.

The following Figure 5.1 provides an illustration of the narrative of the interactions that started in Chapter 1. As can be seen here, the domain experts design their knowledge in

\(^2\)https://github.com/ontodev/robot/

\(^3\)The source code means that the spreadsheet is not imported, but remains the preferred form for editing.
this semi-structured format using Microsoft Excel and create the spreadsheets to store it. They cannot access the source code of Tawny-OWL, they only have the ability to explore the finished product of the ontology. On the other hand, ontology experts design the source code and use that information in the spreadsheet to build the ontology inside the environment of Tawny-OWL. By this approach, information is extracted from the source of knowledge that is developed by domain experts as a spreadsheet.

![Diagram showing interactions in ontology development using Excel spreadsheet.](image)

Figure 5.1: The interactions in ontology development using Excel spreadsheet.

This chapter demonstrates the application of ODEW through the construction of the tolAPC ontology, using the tolAPC catalog. This shows that the approach is more generally applicable using pre-existing spreadsheets and predefined patterns in spreadsheet formats. In doing so, the research questions RQ2 and RQ4 are addressed.

5.2 A Motivating Example

The tolAPC catalog is used as this was the original motivating example for the spreadsheet driven work. Unlike other forms of ontology construction, the tolAPC catalog already existed as a spreadsheet and was being used to share knowledge between collaborators as
part of an EU Cost Action BM1305 Action to Focus and Accelerate Cell-based Tolerance-inducing Therapies (A-FACTT)\(^4\), which is aimed at increasing data sharing and collaborative working across the community (Brinke et al., 2015). While the knowledge was helpful in a spreadsheet form, the knowledge did not link to other resources. This raises the concern, whether it would be possible to use knowledge in the spreadsheet to generate an ontology without simply rewriting the data in an ontological format. Nevertheless, the community chose to use Microsoft Excel because they understand it adequately and are more familiar with Microsoft Excel formats. This triggers the question: Is it feasible to take this particular spreadsheet catalog, which is how the scientists prefer to work with, and convert it into a formal ontology?

### 5.2.1 The tolAPC Catalog

As the research question in this chapter was motivated by the existing tolAPC catalog, this dataset will be described next. The tolAPC catalog is a list of immunological cell types which have been ‘tolerized’\(^5\). In addition, these cell types have been created to be used therapeutically in various situations, such as the treatment of auto-immune diseases like rheumatoid arthritis, or to reduce rejection after transplantation (Hilkens, 2016). Information about these cells is, therefore, of high value. Thus, ontologies can support the sharing of such information among the group of collaborative scientists. However, they tend to use Microsoft Excel spreadsheets as a mechanism for sharing information. This motivates the combination of tools to produce an ontology.

The tolAPC catalog contains extensive details about these cell lines, including nine ‘sheets’ of data. The catalog has been created as a Microsoft Excel spreadsheet, although it uses the spreadsheet only to represent tabular information (i.e., there is no use of equations or calculations in the spreadsheet). The spreadsheet has been created by individual scientists freely; that is, there is no formal constraint on the legal set of values in each cell, just the social convention of copying previous cells. It also incorporates all preferred facilities of

\(^4\)http://www.cost.eu/COST_Actions/bmbs/BM1305  
\(^5\)that is treated so that they suppress the immune response.
Microsoft Excel, such as colors and fonts. Figure 5.2 shows the structure of the spreadsheet filled with false information due to the confidentiality of the tolAPC catalog.

![Figure 5.2: Mock sample of the tolAPC catalog, showing the structure of the spreadsheet.](image)

### 5.2.2 Initial ODEW for the tolAPC Catalog

The data of the tolAPC catalog was captured directly in a spreadsheet and coordinated mainly through email. The reason behind choosing Microsoft Excel spreadsheet is to enable collaboration between individuals who were geographically separated. As a pre-existing resource, it is impractical to rewrite directly in OWL, either using Protégé or Tawny-OWL. To do so, would have resulted in transcription errors and made updates more complex. However, as described in Chapter 3, all the components needed to build an ontology are directly available from a spreadsheet.

Therefore, the development process was established using the new document-centric approach that incorporates Microsoft Excel spreadsheets (This approach is shown in Figure 5.3). Tawny-OWL has programmatic facilities to design functions that will utilize the purposes of the ontology development process. The spreadsheet is directly read, and all values needed to instantiate the ontology patterns are extracted. However, the patterns are designed according to the Microsoft Excel spreadsheet layout and cell organization. So any ontology designed using this mechanism will have its extraction techniques depending on the design of the spreadsheet. The final ontology can be saved as an OWL file to be browsed using Protégé or a web browser.
The tolAPC ontology was developed using two sheets of the catalog 'workbook'. Moreover, the patterns were designed based on the layout of rows and columns in the spreadsheet. The ontology has 13 main classes, 8 subclasses, and 13 object properties. Using features of Tawny-OWL and Clojure, such as user-defined variables, lists, functions, and mapping, the ontology patterns were filled with their values while parsing the spreadsheet information. As can be seen from Table 5.1, while parts of the tolAPC catalog have been recast, more sheets have not been converted.

In the next section, the development of tolAPC Ontology will be described in detail, showing steps and challenges faced during the modeling process.

Table 5.1: Statistics of the tolAPC catalog and the tolAPC ontology.

<table>
<thead>
<tr>
<th></th>
<th>tolAPC Catalog</th>
<th>tolAPC Ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Sheets</td>
<td>9</td>
<td>Number of Classes</td>
</tr>
<tr>
<td>Number of Cells</td>
<td>1181</td>
<td>Number of Individuals</td>
</tr>
<tr>
<td>Number of Cell types</td>
<td>15</td>
<td>Number of Object Properties</td>
</tr>
</tbody>
</table>

Figure 5.3: tolAPC approach using Microsoft Excel spreadsheets and Tawny-OWL patterns.
5.3 The tolAPC Ontology

This section describes the first application of this methodology (ODEW) to generate a catalog of immunological cell types, based on the tolAPC catalog. It discusses the building process, the design decisions faced, and the general implications of this approach for ontology development.

5.3.1 Building the tolAPC Ontology

This segment describes the construction process which can be conceptually divided into three phases:

1. Extraction:

The extraction phase is straightforward. Clojure offers many libraries capable of reading and parsing spreadsheets. In the case of the tolAPC catalog, the spreadsheet is read using the Docjure library, accessed directly from the file system. It would also be straightforward to read from a network that would support building ontologies from cloud-hosted spreadsheets. Previously in Warrender et al. (2015b), they have read and then cached the results of tests generated from a spreadsheet for performance reasons. However, for the case of the tolAPC catalog, performance is such that the spreadsheet can be read in full, every time the environment is initialized, significantly simplifying the development.

2. Validation:

In the second phase, values extracted from the spreadsheet are validated against a set of constraints, specifying those which are legal. For many of the fields, values are highly stereotyped, having only a few different options; for example, cells can either be Autologous or Allogeneic, while expression levels can either be positive (+) or negative (−). Currently, validation is performed through the use of ad hoc testing – moving to a more formal data constraint language is expected in the future. The
5.3 The tolAPC Ontology

choice of validation depends on the requirements and modeling choices made, which will be discussed in Section 5.3.2.

3. **Ontologization:**

In the third phase, values are ‘ontologized’. The top level of the ontology, which provides what is described as *schema terms*, is written by hand using Tawny-OWL. In the case of the tolAPC catalog, this includes terms such as *CellType*, *Species*, and *AntigenLoad*. Next, a set of patterns are defined using these schema terms. Finally, these patterns are instantiated using the values from the second phase, generating entities that are called *patternized terms*.

During the development process, both reasoning and manual inspection of the created ontology are used to ensure that the process is running as expected. For the latter process, the ontology is saved to file and examined, either in the Clojure development environment or within Protégé, as shown in Figure 5.4.

![Figure 5.4: tolAPC ontology displayed from Protégé screen.](image)
5.3.2 Modeling in the tolAPC Ontology

All entities in the ontology need to be represented by an Internationalized Resource Identifier (IRI). Two broad schemes are used to generate IRIs: semantics-free identifiers, which are generally numeric, and semantically meaningful identifiers, generally derived from the common name for the entities. Generally, the latter is easier to work with, while the former is easier to keep stable over releases.

Currently, for the tolAPC ontology, schema terms have IRIs which reflect their names (CellType uses an IRI with a fragment of 'CellType'), while patternized terms use an ad hoc schema based on several of their properties (a single property is not enough to ensure uniqueness). If a re-evaluation of this situation is required in the future, using Tawny-OWL simplifies the situation: so, developers can efficiently allocate IRIs to entities according to any scheme that they choose by changing a single function.

Another recurrent issue in ontology modeling is whether to use classes or individuals. Within the tolAPC ontology, this question is raised for cell types. There are several different criteria for making this decision (Stevens et al., 2013). Considering a 'Realist' perspective, the following can be stated: In terms of modeling as a single entity, cell types are probably best represented as a metaclass, akin to a taxonomic species (Schulz et al., 2008). It must be additionally noted that modeling as multiple entities (differentiating between the protocol and the cell type produced) would also be possible. However, there appears to be no clear principle to distinguish between these options. Similar problems also arise for proteins/cell-surface markers which are described in the ontology. As an additional problem, these representations introduce considerable unnecessary complexity (Lord et al., 2010).

Therefore, the needs of the current application are considered: it seems unlikely that subclasses of a cell type will ever be needed, but might reasonably be desired for cell types to be unique – to state that two cell types are necessarily the same (or different) individual. For these reasons, cell types are modeled as individuals. An example of the ontology class structure is shown in Figure 5.5.
The tolAPC ontology essentially models a set of cell types with the rest of the ontology designed to support these cell type descriptions. The ontology, as a result, contains very little hierarchy and is at the end of a normalized ontology (Rector, 2002). Cell types are defined as individuals with a large set of different property assertions, as can be seen from the following definition:

\[
\text{(individual cell-name} \\
\text{:fact (is fromGroup group)} \\
\text{(is hasLocation loc)} \\
\text{(is fromClinicalDisease clinic-disease)} \\
\text{(is fromSpecies from-species)} \\
\text{(is hasStatus stat)} \\
\text{(is hasType c-type)} \\
\text{(is hasDescription desc)} \\
\text{(is hasActivation active)} \\
\text{(is hasAntigenLoad anti-load)} \\
\text{(is itsOrigin cell-org)} \\
\text{(is withStartMaterial start-material)} \\
\text{(is hasIsolation isol)})
\]

Here, cell-org, group, loc, and others are variables; therefore, this definition describes a pattern. fromGroup, hasLocation, and others are specific object properties from the schema terms of the ontology. individual, :fact, and is are part of Tawny-OWL syntax. The whole definition defines a new cell type and its association with a set of individuals.

The values of the property assertions fall into one of three main categories:

1. **Open but Limited**: Many properties have a limited, but still open, range of values.
Examples of these are `withStartMaterial`, which describes the tissue or part of the tissue from which the cells are derived from. These values are modeled as disjoint classes, explicitly stated in the ontology. Although an external ontology could have been used at this point, as only a few options are used, the choice was not to import one to maintain simplicity.

2. **Constrained Partition:** Many properties support an exact number of options. These are modeled with the use of a Value Partition (Rector, 2005). Fortunately, Tawny-OWL explicitly supports this design pattern, allowing for a concise definition (Lord, 2013b). An example of this is `CellOrigin` which is defined as follows:

```deftier
(CellOrigin
  [Allogeneic :comment "Allogeneic stem cell transplant uses a donor blood"]
  [Autologous :comment "Autologous stem cell transplant uses a patient own blood"])
```

3. **Unconstrained Values:** Some properties have unconstrained values such as `Location`, `Group` (i.e., the people responsible for the cell type), or `AntigenLoad`. These are currently modeled as individuals and created on-demand.

In some cases, these values also reuse terms from external ontologies; currently, the `Species` term refers to the National Centre for Biotechnology Information (NCBI) taxonomy. However, there is no import for the full semantics of this ontology because it would cause a considerable increase in reasoning time for a relatively low reward. This is a common problem in biomedical ontologies.

In addition to these three main categories, phenotype descriptors are added to the cell types in raised or lowered expression levels. For these, the expression levels are modeled as a value partition, while the overall phenotype is modeled using the N-ary relationship pattern (N. Noy et al., 2006), as shown in Figure 5.6.
5.4 Generalization of ODEW

After the successful development of the tolAPC ontology using the approach ODEW\(^8\), a generalization of the framework is developed using the Tawny-OWL tool Bubo\(^9\). Bubo is a batch library to run Tawny-OWL. It reads and transforms CSV into ontology. It has been developed by Phillip Lord and the author. This generalization workflow is depicted in Figure 5.7. The spreadsheets now have restricted patterns to be followed when created for correct deployment in the next step of running the Bubo command to generate the ontology. Bubo can be run using the command line in Windows or Linux. Therefore, domain experts and ontology developers can run the command. The ontology is then saved and can be presented and checked using the proper tooling. This then validates the correctness of the ontology and shows the compatibility of the spreadsheet with the ontology output.

5.4.1 The Patterns in Bubo

The author has extended Bubo to include Microsoft Excel spreadsheets’ reading and extraction in this generalization task. Three predefined patterns are developed as starting points to extract values from the spreadsheets:

---

\(^8\)In addition, Amino Acid Ontology has been developed using the same approach.

\(^9\)https://github.com/phillord/tawny-bubo
5.4 Generalization of ODEW

Figure 5.7: A Workflow using Microsoft Excel spreadsheets and Bubo in Tawny-OWL.

1. **The horizontal pattern** where a class is linked to another via an existential constraint. This pattern reads the data in a horizontal style (row by row). Therefore, the data in the spreadsheet is horizontally organized as illustrated in Figure 5.8a.

   ```lisp
   (defclass A :super (some r B))
   ```

   This is a straightforward pattern that links two classes using a predefined property. The table, in this case, does not have a header row; all values are directly written in the table as shown in Figure 5.8a. This is the same pattern as the CSV pattern in Bubo.

2. **The individual pattern** where the header values are classes and the rest are individuals. This reads the spreadsheet horizontally as well; then, defines the first-row values as classes, and the rest of the sheet data are defined as individuals (Figure 5.8b).

   ```lisp
   ((doall (map owl-class header))
    (doall (map individual individuals)))
   ```
This pattern is applied to the case of tolAPC Ontology since most of the values are modeled as individuals, in conformity with Section 5.3.2 earlier. The tolAPC catalog is designed beforehand, and the format is unusual; therefore, the development of that ontology was restricted and designed to match the spreadsheet layout.

3. **The tier pattern** where the column values are defined as a tier (Lord et al., 2017). In this pattern, the column needs to be specified as an argument, and the data is extracted vertically. The first value of the column is the superclass, and the rest are the subclasses.

\[
\text{doall}
\quad \text{tier (first column-data) (rest column-data))}
\]

This pattern is applied to the example of Amino Acids Ontology, as can be seen in Figure 5.8c.

These patterns have been chosen due to their simplicity and the application of the examples used in this method.
5.4 Generalization of ODEW

(a) Microsoft Excel spreadsheet for ontology Pattern 1.

(b) Microsoft Excel spreadsheet for ontology Pattern 2.

(c) Microsoft Excel spreadsheet for ontology Pattern 3.

Figure 5.8: Microsoft Excel information for ontology patterns in Bubo.
5.4.2  Bubo Validation Using a Musical Spreadsheet

Afterwards, to validate those predefined patterns in Bubo, another independently generated spreadsheet was used to construct an ontology. For this purpose, a free resource about UK female musicians for the industry is used, which was obtained as a spreadsheet file\footnote{Downloaded from https://www.bbc.co.uk/news/entertainment-arts-51723513 in March 2020. Now it has its searchable directory site in https://thef-listmusic.uk}. This spreadsheet includes female songwriters and composers, covering all genres from rock and pop to jazz and classical. Two spreadsheets from this resource were used for the validation process and shown in Figure 5.10. Sheets with empty cells are avoided for this process as to maintain the accuracy of the implementation.

Bubo parses the spreadsheets and utilizes the data in the specified columns to create the ontology’s axioms using the predefined patterns. Pattern 3 suits the data provided in these spreadsheets. Accordingly, each column header is the superclass of the elements beneath it. The spreadsheets extracted from this musical resource each contain 364 and 1337 rows each (see Table 5.2). As a result, a total of 1403 subclasses of five superclasses were created in the ontology. A simple workflow of this conversion is illustrated in Figure 5.9.

<table>
<thead>
<tr>
<th>Sheet</th>
<th>No. of rows</th>
<th>No. of columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK Self-Releasing Artists&amp;Compo</td>
<td>364 rows</td>
<td>2 columns</td>
</tr>
<tr>
<td>UK ClassicalMediaPop Publishers</td>
<td>1337 rows</td>
<td>3 columns</td>
</tr>
</tbody>
</table>

Table 5.2: Amount of data extracted from the Microsoft Excel spreadsheet source.

Musical spreadsheet

- Number of Sheets
- Number of Cells
- Number of Columns converted

Musical Ontology

- Number of Classes
- Number of Individuals
- Number of Object Properties

A few difficulties arise during the process of converting the Microsoft Excel spreadsheet into an ontology. Firstly, there are white spaces between words in one cell, including entire empty cells. Secondly, there is a presence of special characters, such as &, accent letters, or dots. Lastly, URLs texts are a problem. The latter can be added as an annotation property.
in the future, whereas, the former two were adapted using one of the Tawny-OWL methods to deal with these characters, including spaces.

After all, Bubo’s predefined patterns are sufficient to construct the ontology. Even though no annotation values (website or email address) have been added, these could be included in the future.

5.5 Discussion

In this chapter, the author has described the ODEW and presented the initial example of the tolAPC ontology that describes immunological cell types. This ontology is unusual because it is derived directly from another data resource, the tolAPC catalog, which is maintained and remains as a Microsoft Excel spreadsheet. Essentially, the ontology provides context and semantics to data that is available in another form. This form is unconstrained and initially created by domain experts before the development of the ontology.

The value of recasting a spreadsheet into a form with precise machine-interpretable semantics is clear: contradictions can be recognized, documentation attached, and a global namespace means that it is possible to share names amongst spreadsheets. However, there are less apparent virtues arising from the process. In the initial validation step, further clarifications are required for certain parts of the tolAPC catalog that are otherwise unclear. For example, one cell line is described as ‘Autologous/Allogeneic’. The original intention of the spreadsheet’s author is unclear here. This could be intended to mean either autologous, allogeneic (possible), both (probably inconsistent), or just the absence of knowledge.
Figure 5.10: Microsoft Excel F-List of UK female musicians and writers.
Similarly, the process of 'ontologization' forces us to clarify some areas of the biology, including questions about whether cell types produced by the same protocol at different times are the same or not, which touches on issues of reproducibility. These issues have arisen; either the ontology schema, patterns, or the spreadsheet can be modified accordingly. Currently, validation is performed by hand-specifying constraints as enumerations of strings. In the future, this could be moved to a more declarative approach; fortunately, Tawny-OWL is implemented over a full programming language. On that note, there are many different data constraint languages such as Prismatic (Fan et al., 2010) or clojure.spec11. It must be remembered that validation will help to enhance the ontology development process further.

Additionally, while adding machine-interpretable semantics is useful in its own right, the issue of interoperability with other ontologies requires attention. Currently, child terms of Species reuse IRIs from the NCBI taxonomy, the mapping between the free text used in the tolAPC catalog and the NCBI taxonomy, is stored in a literal data structure in source, but could also be stored in a flat-file or subsidiary spreadsheet. The full ontology has not been imported for reasons of performance; the process is known as a 'soft import' (Lord, 2013a).

Developing a programmatically defined ontology allows developers to switch easily between 'soft', 'hard', and MIREOT-style 'semi' imports (Courtot et al., 2009). Conversely, child terms of ClinicalDisease do not currently relate to other ontologies. At the current time, this process has not been prioritized because of confidentiality restrictions on the tolAPC catalog, which limits the ability to share results. Adding this form of interoperability is not complex, as this has already been demonstrated with Species and by using the 'scaffolding' process described previously (Warrender et al., 2015c).

This research builds on earlier work using a karyotype ontology where patterns are instantiated using in-code literal data structures (Warrender et al., 2019); the mitochondrial ontology which is scaffolded using a variety of different input formats; or our reworking of Semanticscience Integrated Ontology (SIO) which patternizes a pre-existing ontology (Warrender et al., 2019).  

11https://clojure.org/news/2016/05/23/introducing-clojure-spec
Patternization allows the development of an ontology to be performed rapidly and repeatedly.

The use of Microsoft Excel spreadsheets to drive ontology patterns is not new as mentioned earlier in Chapter 2. It is directly supported with Protégé plugin Cellfie (O’Connor et al., 2010), as well as with tools such as RightField (Wulstencroft et al., 2011) and Populous (Jupp et al., 2010). The key addition of the current methodology is to incorporate the spreadsheet as a part of the ontological source code. The spreadsheet can be updated, changed, and consulted by the domain specialists who created it, and remain part of the ontology development process. The importance of the right format should not be underestimated; for example, early versions of The Gene Ontology (GO) were developed in their bespoke syntax (later to evolve into Open Biomedical Ontologies (OBO) Format), something which persisted for a considerable time after the development and release of OWL. The reasons for this were simple: OBO Format behaved well in a version control system and could be easily created, edited, and manipulated in a text editor; something that is not true of the Resource Description Framework (RDF) serialization of OWL available at the time. The aim is to build on these lessons: ontologists should seek to interact and build on the tools that domain specialists already use to describe the knowledge that these specialists have.

A summarised comparison between the tools mentioned earlier and the ODEW is shown in Table 5.3.

For a user-centric reason, there is no rigid Microsoft Excel template. Rather than that, experts must adhere to mild pattern restrictions to create a suitable spreadsheet for their purposes. Accordingly, domain users will be happy and comfortable using their usual tool (Microsoft Excel spreadsheets), and ontology developers can conveniently program and produce an ontology using Tawny-OWL. Conversely, one disadvantage of this approach is that domain users normally only interact with one part of the ontology source; the spreadsheet may be correct regarding the domain, but the ontology is wrong. Thus, investigating techniques for making the Tawny-OWL section of the ontology more readable (Lord et al.,

Table 5.3: Comparison of ODEW with other tools.

<table>
<thead>
<tr>
<th></th>
<th>Cellfie</th>
<th>Populous</th>
<th>Rightfield</th>
<th>ROBOT</th>
<th>ODEW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spreadsheet import</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>GUI</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Command-line</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Editing spreadsheet</td>
<td>Yes&lt;sup&gt;a&lt;/sup&gt;</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes&lt;sup&gt;b&lt;/sup&gt;</td>
</tr>
<tr>
<td>Multiple use of the spreadsheet</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Free design of the spreadsheet</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<sup>a</sup> The spreadsheet can be edited but this requires to re-import the new version and re-do the work again.

<sup>b</sup> This requires the ontologist to check and re-run the source code of Tawny-OWL.

2015b) will be discussed in Chapter 6.

Designing general templates for potential improvements would benefit domain experts who may not have a consistent framework for their knowledge, allowing them to begin organizing their data in a semi-structured manner. A variety of pre-existing schemes may be utilized, such as MAGE-TAB (Rayner et al., 2006) and ISA-TAB (Johnson et al., 2015) which have been used in data-sharing in spreadsheet forms. Additionally, annotation properties’ values are worth adding to the Microsoft Excel patterns as these properties serve the semantic of the data and assist in linking information between classes and individuals in the ontology. For example, rdfs:label, rdfs:comment, and rdfs:seeAlso can be added with the predefined pattern to enrich the semantic in the ontology. In addition, FAIR data principles (Wilkinson et al., 2016) are considered while converting musicians’ lists into something that is machine-readable. This mainly happens because most spreadsheets are big data tables rather than something computable by equations and functions.

The author has described the development of an ontology, where the information was initially developed by domain specialists using a Microsoft Excel spreadsheet for collaboration. This spreadsheet has been transformed into an ontology using highly programmatic and pattern-driven ontology development. Critically, the spreadsheet remains part of the
source for the ontology; the domain specialists are free to update it, and changes will per-
colate to the end ontology. Using this approach, The author has developed a new ontology
describing immunological cell lines built by instantiating ontology design patterns written
programmatically, using values from a spreadsheet catalog.

This method employs a spreadsheet that domain experts developed. The spreadsheet is
unconstrained in its usage and can be freely updated, resulting in a new ontology. This
provides a general methodology for ontology development using data generated by domain
specialists like the one presented in the tolAPC ontology earlier. It must not be forgotten
that Bubo is a Tawny-OWL tool that converts spreadsheet cell values to ontologies. Aside
from some annotation values that may be added in the future (website or email address),
the predefined patterns are adequate for building the ontology. Although current versions
of ontologies do not include annotations or hyperlinks, this could be a future upgrade to
Bubo.

5.6 Conclusions

In conclusion, the tolAPC ontology has been successfully developed based on the tolAPC
catalog, using Microsoft Excel spreadsheet as a source of information. Critically, the
spreadsheet is unconstrained by the ontology developers, having been freely developed by
the domain users. Moreover, the author has not converted the spreadsheet into a one-
off process; the spreadsheet is part of the source code for the ontology and can be freely
updated. Taken together, this demonstrates a new methodology for building an ontology
that enables us to interact with domain specialists using their opted tools.

While this chapter demonstrated that structured formats in Microsoft Excel could store
knowledge of ontologies, this capability is relatively limited. Ontologies do not fit cleanly
into a spreadsheet, but often require lots of associated text. It was, therefore, interesting to
consider whether more textual approaches could be applied using, for instance, Microsoft
Word in place of Excel. Importantly, the document-centric approach of ODEW embodies
a first step toward establishing a richer methodology, where the interaction with domain
specialists is upheld by using their toolchains to capture knowledge. In the next Chapter, Microsoft Word documents are introduced in the ontology development pipeline to build accurate and well-designed ontologies.
Chapter 6

Ontologies’ Wordification

Contents

6.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 132
6.2 A Path to Using Microsoft Word in an Ontology Development . . . . . . . 134
  6.2.1 Generating Microsoft Word Documents . . . . . . . . . . . . . . . . 135
  6.2.2 Implementation Use Case: Cloud Ontology . . . . . . . . . . . . . . 136
6.3 Reflections on Ontologies Wordification . . . . . . . . . . . . . . . . . . . 141
6.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 143

₀This chapter is modified from Blfgeh et al. (2017a)
Abstract

Collaboration between ontology developers and domain experts is necessary to build suitable quality ontologies. This collaboration requires a continuing interaction between the two groups, which is usually problematic since they tend to use different toolsets which can impede this interaction. This chapter aims to decrease the gap in tooling between domain experts and ontology developers by creating more readable forms of ontologies by wordifying them, which further enables editing in the standard Microsoft Office environment.

6.1 Introduction

Building ontologies is a highly collaborative process involving domain specialists. The previous chapter discussed incorporating Microsoft Excel into ontology development, as Excel is an environment where domain specialists are well-versed. Nevertheless, Microsoft Excel is still only a secondary tool. Probably the most popular tool for authoring among domain specialists is the word processor. Hence the question emerges, can Microsoft Word be used to help in building ontologies? This chapter investigates this question and presents a partial solution by combining the literate ontologies described earlier with Microsoft Word to develop a combined author, review, and correction approach used in developing ontologies.

Microsoft Word is a predominant text environment. In most science disciplines, this is the standard tool used for addressing many types of communication or documentation requirements. Therefore, visualizing an ontology in such an environment seems an attractive mechanism to improve the interaction between domain specialists and ontology developers in developing ontologies.

This research builds on several significant pieces of work. Previously, in ontology development, Solomon et al. (2000) introduced the notion of an intermediate representation that allows reconciliation of two contradictory requirements. The ontologist requires an ability to represent the precise, yet complex statements that description logic enables and an ontology requires. The domain specialist, by definition, has a complex and rich understanding of the domain, but will wish to express this in a technically simple way. Alternatively, according to Soloman and colleagues: “a successful clinical terminology must disappear
into the underlying infrastructure” even when producing a clinical terminology. To solve this problem, they introduced the GRAIL intermediate representation, a controlled format familiar enough for clinicians to write and edit, but precise enough to translate into a description logic.

Software engineering takes a related approach with Behaviour Driven Development (BDD). Tools such as Cucumber\(^1\) attempt to bridge the gap between written requirements documents—they describe the needs of the user in free (if somewhat formalized) text and can be read and checked by the user, but not computationally—and unit or integration tests that are directly computationally interpretable and executable, but quite incomprehensible to the user. With Cucumber, users write scenarios in a constrained free text format (called ‘Gherkin’). These are rendered into executable tests via regular expression matching. This is fragile, so it is generally assumed that Gherkin is written by developers and then approved by users (Wynne et al., 2012)—i.e., the intermediate representation is read-only for users.

As shown previously in Chapter 3, Tawny-OWL is a fully programmatic environment, which means that comments and any other markup can be freely added to the source code. It is this feature of the environment that allows the generation of alternative representations of an ontology.

Following the interactions approaches in the thesis, Figure 6.1 shows how the ontology is generated as a narrative format and use the Microsoft Word document to enable the facilities of editing and track changes in order to help in developing the ontology interactively using document tools. The domain experts read and explore the ontology as a Word document which enable them to edit, add comments or modifications as they desire. However, ontology experts by having the access to the source code they can incorporate any changes to the ontology document by using Word facilities and then generate the new version of the ontology using the environment of Tawny-OWL. This approach uses the documentation procedure to interact between the two groups using the document’s tool.

This chapter demonstrates the representation of literate ontologies using Microsoft Word.

\(^1\)https://cucumber.io/docs/cucumber/api/
This presentation provides a new mechanism to interact with domain specialists using a familiar tool, as well as deploying the features of Microsoft Word to aid in the development of ontologies. In doing so, the research questions RQ3 and RQ4 are addressed.

6.2 A Path to Using Microsoft Word in an Ontology Development

While the use of Microsoft Word for developing Ontologies is, to our knowledge, novel there has been prior work on developing literate ontologies, two of which are described in Chapter 2. The first is the Amino Acid Ontology, taken from the International Conference on Biomedical Ontology 2015 tutorial about Tawny-OWL², while the second is a version of a Karyotype ontology (Lord et al., 2015b). They were both produced using the Tawny-OWL source code in a two-step process: First, the comment characters ‘%%’ are moved from the beginning of comment lines to markers that define the start and end of the formal source code; this step is necessary because the source code must be interpreted as

²https://phillord.github.io/take-wing/take_wing.html
a Clojure source code and text formatting language. The Emacs lentic package is used for this purpose; this package enables the transformation to happen on the fly, which means the files are editable using either text markup or Clojure source code. Second, the markup in the comments is interpreted using a markup processing tool; in this case, AsciiDoc and Emacs Org-mode, respectively. Figure 6.2 shows a snippet from the literate ontology Amino Acids as a webpage.

![Figure 6.2: Literate Amino Acid ontology in HyperText Markup Language (HTML) representation.](image)

6.2.1 Generating Microsoft Word Documents

Following on from generating HTML, Microsoft Word documents were generated using a slightly modified approach. The Pandoc tool is used for this step because it natively supports various text markup languages, including Microsoft Word. Pandoc was used out-of-the-box for the initial experiments, providing an uncomplicated representation of the underlying Tawny-OWL source code and its commentary. In this context, this is referred to as wordification.

Additionally, the work described in Chapter 4 was combined by running Pandoc over an ontology with both the source code and commentary written in Arabic. Pandoc deals with this text appropriately and generates a readable Arabic Microsoft Word document. Figure 6.3 shows the first steps of including the Arabic text with the English in Pizza Ontology and displaying this in two formats, HTML and Microsoft Word.
In both cases, and unlike the HTML representation, the Microsoft Word document can be freely interacted with; it is possible to change or update in this manner. Moreover, it is possible to use standard 'Track Changes' technology to record these changes. In this sense, the format provides a form of intermediate representation; although the text is broadly similar to the underlying source (minus markup), the visual presentation is quite different, as is the interaction mechanism.

6.2.2 Implementation Use Case: Cloud Ontology

The investigation into the wordification of a literate ontology began using the Cloud Ontology and testing the output format. Cloud classification is a crucial task because the clouds...
influence the weather and climate locally and globally; therefore, understanding the cloud
types and movement is essential for air traffic control and weather forecast (Rumi et al.,
2013). There has been much research on cloud classification using Infra-Red (IR) images
or channels (Liu et al., 2011; Rumi et al., 2013) and neural networks classifiers (Heinle
et al., 2010; Lee et al., 1990).

Hence, the Cloud Ontology was developed based on the Met Office cloud classification;
the documentation and essential information for the ontology are obtained from
the same source\(^3\). This ontology has been developed by Mohammad Halawani, a Ph.D.
candidate in the Interdisciplinary Computing and Complex BioSystems (ICOS) Group at
Newcastle University, using a pattern-driven approach (Warrender et al., 2013) to build
the ontology. The author has added the ontology documentation to the source code and
then converted it to HTML and Microsoft Word files.

The author followed the same methodology as Lord et al. (2015b); lentic and the UNICs
of ORgaNizers (Org-mode) files are used for the wordifying process. Using these allow
options to display and format the documentation in the wordified version. For example, in
Figure 6.4, the Cloud Ontology is displayed in Microsoft Word with headings, hyperlinks,
and the ontological source codes. Alternatively, these source codes can be hidden using
\texttt{comment} blocks in markup. However, in the HTML format, this feature can be implemented
using a clickable 'button' element to show or hide the source code as the user desires.

Figure 6.5 shows the Cloud Ontology in HTML with the button 'show/hide Tawny-OWL'
to show the source code on the web page.

The next section shall describe wordification tools, explaining how they were used to de-
velop and generate the wordified literate ontology.

\subsection*{6.2.2.1 Lenticular Text and Org Files}

Lentic\(^4\) is an Emacs package that provides the ability to share the same or similar con-
tent of two different files simultaneously. It has specific support for Clojure that can be

\(^3\)https://www.metoffice.gov.uk

\(^4\)http://homepages.cs.ncl.ac.uk/phillip.lord/lentic/lenticular.html
Chapter 6

6.2 A Path to Using Microsoft Word in an Ontology Development

Figure 6.4: Wordified Cloud Ontology.

(a) Cloud Ontology with code hidden.  
(b) Cloud Ontology with code displayed.

Figure 6.5: Cloud Ontology represented as a web page.

combined with either LaTeX, AsciiDoc, or Org-mode. There is also support for Haskell, LaTeX literate programming, and ROT-13 transformation (Lord, 2015a). Lentic is a good literate programming tool due to its multi-modal editing ability (two different lentic files
in two different modes; one for the source code and one for the literate text, which can be concurrently edited). Figure 6.6 depicts the two lenticular files of the Pizza Ontology.

![Figure 6.6: Two files of the Pizza Ontology; the top view is the lentic view while the bottom is the original source file.](image)

After the source code file of the Cloud Ontology is written in Tawny-OWL, the author started incorporating the documentation content as text into the source file of Tawny-OWL. This is done by adding comments to the source code on the ontology and combining these comment lines with markup language tags to format the documentation output in the Microsoft Word file. For example, to define a header line, an asterisk is used at the beginning: (;;; * Classification of clouds), and the semicolons indicate that this is a comment line.

Firstly, the ontology script is divided into sections and subsections based on the developer’s comments in the source code; for each section/subsection, a header/subheader markup is added with the heading’s text. By doing this, not only the generated text would have a structure, but it could also be shown on the web page as a navigation list (see Figure 6.7). Afterward, explanatory statements and information in every part were inserted as regular text, like comments in the code. Finally, any source code of Tawny-OWL script is re-
As the ultimate goal is to produce a well-formatted documentation file, this Tawny-OWL Clojure file with the commentary and markup is converted to an org file using the lentic package in Emacs. Now, the ontology is saved as an org file that is directly viewable as a web page or a PDF file using the Org-mode commands in Emacs. Following this step, the next step is to make this literate ontology editable so that users can make changes to it. Thus, another tool, Pandoc, is used to accomplish that editable version, which will be described in the following section.

### 6.2.2.2 Pandoc and File Conversion

The Pandoc tool is described previously in Chapter 3, Section 3.7. Pandoc is used to convert the org file to a Microsoft Word document file, which was done successfully (see Figure 6.9). Given that the org file of the ontology is available, it can be converted to any other readable format with Pandoc commands, which can serve the conversion since the
Figure 6.8: Cloud Ontology file in Tawny-OWL (Emacs screen).

Since Pandoc is a command-line tool, a small bash file is written which converts all org files in the directory to Microsoft Word document files. Figure 6.10 illustrates the workflow of wordifying a literate ontology using Tawny-OWL and other tools described earlier.

### 6.3 Reflections on Ontologies Wordification

Having described and given the approach of wordifying ontologies, this now provides a form of ontology in which domain users can engage, interact with, and edit more naturally. Additionally, the ontological source code is translated into alternative visualizations such as HTML and Microsoft Word documents that map directly back to the source, but which can differ from it; for instance, enabling hyperlinks, adding section links, and hiding source code in favor of commentary. Particularly with wordified ontologies, there is a novel interaction mechanism for processing an ontology, where users can edit and make comments. Moreover, with 'track changes' option enabled, this can be used as a framework for reporting changes.
and feeding information back to the ontology developers, especially in the documentation process.

Previously in Chapter 4, it was shown that there is a possibility to translate a textual environment such as Tawny-OWL into another human language, or even a different script, including right-to-left text. While a fully multilingual solution may be less relevant for scientific ontologies, the same technology can change the text for other areas of expertise. For example, the use of common phrases such as some and only in Tawny-OWL, rather than the more formal universal and existential, already demonstrates the use of alternative language for ontology development.

This wordification is not a proposal that the wordified ontologies are intended to be used directly by domain specialists to edit ontologies. Ontologists are expected to work with domain users to incorporate improvements suggested by them; in this sense, a wordified ontology is used as an intermediate representation (Rector et al., 2001) with more facilities from Microsoft Word documents. Equally, the review features provided by the wordified ontology can help create a rich environment to support the ontologists through the devel-
6.4 Summary

Building on a programmatic ontology development environment like Tawny-OWL, the ability to wordify ontologies as a user-friendly, readable, and editable format from the raw source ontology and its embedded comments is explained in this chapter. Additionally, this has been translated to HTML for reading, where this environment supports extensive hyperlinking and functional features such as concealing source code in favor of comments. Additionally, the wordification of these ontologies is aimed to facilitate and enable editing by domain users. Together, this should provide a significant new route for collaboration between the ontologist and domain specialist. This wordification approach will be tested and evaluated by users with different experiences on ontologies in the next chapter.

Figure 6.10: The workflow of producing the literate ontology.
Chapter 7
User Evaluation

Contents

7.1 Introduction .................................................. 146
7.2 Arabic Version of Tawny-OWL ............................... 149
  7.2.1 Distribution Procedure .................................. 150
  7.2.2 Results of Arabic Questionnaire ......................... 150
  7.2.3 Characteristics of Respondents ......................... 150
  7.2.4 Marking Criteria ....................................... 152
  7.2.5 Accuracy and Errors in Answers ....................... 153
  7.2.6 System Usability Scale (SUS) ......................... 155
7.3 Microsoft Excel ............................................. 157
7.4 Wordified Ontologies ....................................... 158
  7.4.1 Stage 1: Empirical Test (Controlled) .................. 159
  7.4.2 Stage 2: Remote Test (Semi-Controlled) ............... 161
7.5 Qualitative Data Analysis of Wordification Experiments .... 163
7.6 Results of Wordification Experiments ..................... 163
  7.6.1 Stage 1 ............................................. 163
  7.6.2 Stage 2 ............................................. 166
7.7 Discussion and Conclusions ................................ 175
7.8 Summary .................................................. 177

*Part of results in this chapter are published in https://drive.google.com/file/d/1Cv46IELah_yCDFoLUke9Xi4qxxernW1L/view*
Chapter 7

7.1 Introduction

Abstract

In this user evaluation stage of the study, the methodologies used to evaluate both the Arabic version of Tawny-OWL and the wordifying mechanism are described here. In addition, the results of the evaluation trials are discussed. First, a survey of Arabic users’ engagement with Tawny-OWL Arabic commands was conducted, with the results documented. Second, users were exposed to the wordified ontology and asked to compare their experience with the wordified presentation to a Protégé version. This novel way of presenting the ontology in a wordified format appears to be competitive with Protégé, most likely because users are already familiar with the tooling.

7.1 Introduction

Evaluation is an essential practice in any research to assess the outcomes. It can be defined as analytical procedures to check the fulfillment of goals to measure the success and answer questions about the overall performance (Twersky et al., 2012). In this thesis, the author has developed a document-centric methodology for ontology construction, incorporating Office tools into the ontology building pipeline. Therefore, the evaluation is focused on ontologies and how users interact with them.

There are plenty of evaluation mechanisms to evaluate the final product of ontologies, depending primarily on the reasons behind the evaluation practice. Three approaches to evaluate ontologies are evolution-based, rule-based, and feature-based (Tartir et al., 2010). Each has its benefits and drawbacks. The choice depends on many criteria, such as the purposes of the evaluation or the existing resources (Hartmann et al., 2005). Another methodology for evaluating ontologies is OQuaRE (Duque-Ramos et al., 2013), which is a framework for evaluating ontologies based on SQuaRE, a standard for software quality requirements.

Here, the user experience of ontology engineering is of primary interest. Therefore, the evaluation mechanisms should involve some kind of user activity in order to assess usability. This type of user involvement has previously been applied to various aspects of ontology engineering, including the use of a foundational ontology in ontology development (Keet, 2011), and finding frequent user activities in Protégé, using eye-tracking analysis (Vigo et al., 2015). Moreover, the tools used for reading and understanding an ontology play
a critical role in determining the usability of that ontology (Tan et al., 2017). Using a verbalized version of the ontology in their evaluation practice, Tan et al. (ibid.) found that this supports the identification of mistakes.

In order to determine the value of the proposed approaches, in this thesis, the evaluation asks the following questions:

**EQ1** Is it possible to translate Tawny-OWL commands into Arabic? (which corresponds to RQ1)

**EQ2** Is it technologically possible to build an ontology using Microsoft Excel? (which corresponds to RQ2)

**EQ3** Is it possible for users to comprehend the ontology when it is presented in wordified form? (which corresponds to RQ3)

**EQ4** Is it easy for users to modify and update the wordified ontology? (which corresponds to RQ4)

The document-centric approach can be evaluated using three mechanisms:

1. Evaluation by the action of construction and translation (**EQ1**, **EQ2**).
2. Evaluation by testing it with users, and getting feedback (**EQ1**, **EQ3**, and **EQ4**).
3. Evaluation by teaching users to use the proposed methodology and assessing their performance.

These mechanisms of evaluation are all applicable. The first two were completed in this thesis. However, the third evaluation mechanism has not been performed due to time limitations, the small number of available participants, and restrictions on their time.

The first question (**EQ1**) is answered by translating all necessary macros of Tawny-OWL into Arabic which has been discussed in Chapter 4, and meant to address **RQ1**. The second question (**EQ2**) can be answered by constructing new ontologies using the Excel-based
workflow (discussed in Chapter 5), which is proven that this is technologically applicable for building ontologies using the proposed methodology, which fulfills RQ2 (see Section 7.3). The rest of the questions (EQ3 and EQ4) are answered during the wordification experiments when users interact with the ontologies to understand their structure and/or update them (see Section 7.6). Those experiments address the RQ3.

The following are criteria to assess and determine the success of using the proposed methodologies:

For the Arabic version of Tawny-OWL, the aim was to test,

- How simple it is to use and understand Arabic commands? How tolerant are Arab users toward the Arabic commands?
- How quickly can Arabic developers learn and write correct Arabic commands?
- Measuring the usability of employing an Arabic environment to develop the ontology.

For the Excel-based workflow, the aim was to test,

- The ability to create an ontology using an Excel spreadsheet.
- That the ontology built using the proposed methodology is correct and consistent.
- The rate at which the ontology is constructed and updated.

For the wordified ontology, the intention was to check how easy it is to,

- Read the new representation of the wordified ontology.
- Understand the flow of ontology construction in this form\(^1\).
- Edit/update the ontology.

\(^1\) In order to assess how easy it is to understand and manipulate the ontology, the task of finding errors was added.
The main goal of the research was to produce ontologies that are the following: 1) closely connected to the domain, where domain experts design the Excel as they prefer, and also can freely update the documentation in Word; 2) easy/flexible in debugging, where users/developers can observe any faults/bugs in the ontology. These criteria have different characteristics; some are straightforward to test, while some are less so, and potentially involve significant user studies.

### 7.2 Arabic Version of Tawny-OWL

To address EQ1, a questionnaire with a mixture of qualitative and quantitative questions was used to seek user experience with the Arabic commands. There were reasons for choosing this method to collect this information; firstly, the questionnaire helps gather responses from many participants reasonably quickly. Additionally, there are time and place restrictions due to the pandemic. However, one of the questionnaire’s drawbacks is that respondents are less likely to provide complete answers, resulting in scarce information and difficulties to follow-up questions with participants for obtaining more information.

The questionnaire is designed to test and evaluate the Arabic version of Tawny-OWL. It has the following four parts:

- **First Part:** Describing and exploring the Arabic Tawny-OWL commands to build a simple Pizza Ontology. Three multiple questions were displayed to check the overall comprehension and motivate the participant to get in touch with the Arabic commands.

- **Second Part:** Seven open-ended questions were asked to build a simple Family Ontology and encourage users to write the Arabic commands. Guidance of using Arabic commands and hints are provided with each question.

- **Third Part:** A quantitative feedback survey is conducted using the System Usability Scale (SUS) to measure the usability of this Arabic version of commands.
• Fourth Part: Collecting demographic data (gender, age,...) for analysis purposes.

Ethical Approval
The questionnaire was reviewed and received approval from the SAgE Faculty Research Ethics Committee at Newcastle University (Ref: 9297/2020).

7.2.1 Distribution Procedure
The invitation link to the questionnaire is created with a friendly greeting message to capture Arabic participants’ attention. This invitation link has been spread via emails and relevant social media using a personal academic network.

7.2.2 Results of Arabic Questionnaire
This section is dedicated to analysis of the results.

7.2.3 Characteristics of Respondents
A total of 71 responses were received. However, the percentage who completed the survey was 28% (which is 21 responses). The reason is that participants are constantly challenged with open-ended questions that require them to write statements. In some cases, participants answered no questions beyond the consent form, which means they are not familiar with the topic.

The following charts show the level of expertise among the respondents who completed the questions. The majority have fundamental knowledge about ontologies and their construction. However, there are about 29% and 24% who have good experience in ontologies and their construction, respectively. This is noteworthy because it suggests that the study has a sufficient number of respondents to reveal the true nature of interaction that motivates domain experts and ontology developers to collaborate, as well as the use of Arabic Script in ontology development in general. Additionally, this removes the problems of discrimination and bias; under such conditions, it may be difficult to effectively distinguish between
the participants who are being studied and measurement problems related to ontology development.

![Bar charts](image)

(a) Background level in ontology.  
(b) Background level in ontology construction.

Figure 7.1: Respondents’ level of ontologies and construction process.

According to the statistics, 23.81% of the participants were male, while 76.19% were female. Those who claimed the highest levels of experience were mainly male. Previous studies can be used to explain why there is such a large gender gap in response rate of this study. Multiple user-based studies explain the reason why there is such a difference in response rates between men and women. Due to this widespreadness of a consistent gender effect in survey participation and nonresponse, which is acknowledged as a social phenomenon (Slauson-Blevins et al., 2016), Becker (2022a) examined the effects of survey participation on men and women. It is well observed that women are more likely than men to respond to surveys, especially those conducted via email or the internet (Becker et al., 2018; Smith, 2008). Furthermore, it appears that women are more likely than men to respond quickly after being invited to participate in an online survey (Becker, 2022b, 2019). Even though response rates are generally declining, the gender effect on survey participation rates has been found to be consistent across a wide range of survey modes and survey topics (Slauson-Blevins et al., 2016). On the other hand, it is still unclear whether or not the gender disparity in participation rates among the surveys that make up a multi-wave panel has changed. However, concerns regarding survey participation and non-participation were made in relation to the demographics issue. Additionally, it was
discovered that, at least in terms of the conventional methods of survey administration, there are indeed patterns in who responds to surveys (Smith, 2008). In general, studies have found that those with higher levels of wealth and education are more likely to participate in surveys than those with lower levels of wealth and education (Goyder et al., 2002). Additionally, it has been conclusively proven that women are more likely than men to participate (Moore et al., 2002).

### 7.2.4 Marking Criteria

The following criteria are considered while marking the responses received from the questionnaire. The main concern of this questionnaire is to check that users can learn and write complete correct Arabic commands, giving descriptions of real examples from Arabic Tawny-OWL commands. As a result, syntax errors, rather than runtime errors are substantially relevant as there is no program running.

Since the test was performed online without using the natural Tawny-OWL environment, users were deprived of the IDE’s autocorrection capabilities. Furthermore, the autocorrection of spellings in browsers is determined by the browsers’ general settings and whether the Arabic dictionary is configured or not. Therefore, spelling mistakes are checked leniently, so the first error is taken into account, and if the same error occurs again, no marks are deducted.

Occasionally, users disregard the instructions and rely on their understanding to answer questions, such as using a different class name or answering part of the question. As long as the correct command and structure are followed, no further deduction was made.

As a result, the answer has three possible states:

1. Correct and rewarded, when the correct command is used with a good structure and correct arguments.

2. Partially correct or awarded half a mark, when the correct command is written, but minor errors (e.g., typographical, spelling), syntax errors (e.g., quotes, parentheses), and logical errors (e.g., domain and range) exist.
3. Incorrect and no mark provided when significant errors occur, such as an incorrect command or an incomplete/incorrect structure.

The following Table 7.1 summarizes the conditions for awarding marks.

<table>
<thead>
<tr>
<th>Answer’s Status</th>
<th>Meaning</th>
<th>Awarded Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct</td>
<td>The answer is written correctly followed the grammar and correct syntax of Arabic Tawny-OWL.</td>
<td>1</td>
</tr>
<tr>
<td>Partially correct</td>
<td>The correct command and structure of the Arabic command is used with minor errors (typos, punctuation).</td>
<td>0.5</td>
</tr>
<tr>
<td>Incorrect</td>
<td>The incorrect command is used or major errors are presented (spelling, command’s structure)</td>
<td>0</td>
</tr>
</tbody>
</table>

### 7.2.5 Accuracy and Errors in Answers

The questionnaire consisted of ten questions. The first three questions were True or False. The remainder were open-ended questions that required participants to write Arabic commands. The overall accuracy of the answers is 66.43%. The questionnaire can be viewed in Appendix B.1.

Figure 7.2 shows that the average of correctness for each True or False question is 69%, 52.5% and 72%, respectively.

Table 7.2 shows the percentage of correct answers for the open-ended questions. The total number of questions is seven, and each question is marked based on the criteria of marking mentioned above. It is shown that the correctness is 60% or above.

Taking into consideration the fact that, on average, 66.43% of the responses are correct. If the percentage is greater than fifty percent, then it is reasonable to conclude that the experiment linked to the accuracy can be verified without issues and that is supported by that accuracy percentage. The response correctness rate was further strengthened by the fact that even the lowest level of correctness was 52.5%. This is true even though each True or False questions and open-ended ones have an overall average level of correctness.
that is greater than 50%. This enables the validity of both the extracted responses and the response rate.

Table 7.2: The percentage of correct answers for the open-ended questions in the survey.

<table>
<thead>
<tr>
<th>Question</th>
<th>Correct answers%</th>
<th>Total answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1- Define the ontology</td>
<td>62.96%</td>
<td>27</td>
</tr>
<tr>
<td>2- Define a class</td>
<td>80%</td>
<td>25</td>
</tr>
<tr>
<td>3- Define two disjoint classes</td>
<td>66.66%</td>
<td>24</td>
</tr>
<tr>
<td>4- Define a group of subclasses</td>
<td>73.91%</td>
<td>23</td>
</tr>
<tr>
<td>5- Define inverse properties</td>
<td>60.86%</td>
<td>23</td>
</tr>
<tr>
<td>6- Define two properties with domain and range</td>
<td>68.18%</td>
<td>22</td>
</tr>
<tr>
<td>7- Define an individual</td>
<td>66.66%</td>
<td>21</td>
</tr>
</tbody>
</table>

The most common errors identified in answering the questions can be described as follows:

- **Syntax Errors**
  - Punctuation: parentheses and colons in Tawny-OWL syntax.
  - Typographical: mistype correct letters/words due to copy/paste from the examples or wrong key hits.
  - Spelling: grammatical incorrect words.

- **Programming Errors**
  - Morphological/Structural: orders of arguments in the commands.
  - Logical/Programmatical: incorrect information supplied such as subclass or domain and range for properties.

In addition, participants were asked to type programming commands which they had just self-learned during the time of the questionnaire, and errors that could occur in a program, that fall into the categories of syntax errors, runtime errors, and logical errors (Attaway, 2019). These are generally common mistakes, misspelling is a normal human behavior when browsing (Hargittai, 2006) and in Arabic, there is plenty of misspelling among Arab users. Due to a lack of implementation, runtime errors are not taken into consideration.
7.2.6 System Usability Scale (SUS)

The SUS developed by (Brooke, 1996) is a valid and reliable method for assessing the usability of a broad range of products and services such as website and user interfaces. In addition, SUS can be used on petite sample sizes and still generate reliable results (Sauro, 2017).

It consists of ten definitive statements with a five-point Likert scale so that participants can quickly complete the assessment of the usability, and the final score can be obtained reasonably quickly (See Table 7.3). The odd are positive statements, and even ones are negative. The SUS yields a single final number which represents the overall usability of the system studied (Brooke, 1996).

To calculate the usability score, each positive statement score is calculated by \textit{statement score minus 1}, and the negative statement score is calculated by \textit{five minus statement score}. The overall SUS score is calculated by adding up the statement scores and
Table 7.3: SUS statements with calculation score of each statement.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. I think that I would like to use this system frequently.</td>
<td>score -1</td>
</tr>
<tr>
<td>2. I found the system unnecessarily complex.</td>
<td>5 - score</td>
</tr>
<tr>
<td>3. I thought the system was easy to use.</td>
<td>score -1</td>
</tr>
<tr>
<td>4. I think that I would need the support of a technical person to</td>
<td>5 - score</td>
</tr>
<tr>
<td>be able to use this system.</td>
<td></td>
</tr>
<tr>
<td>5. I found the various functions in this system were well integrated.</td>
<td>score -1</td>
</tr>
<tr>
<td>6. I thought there was too much inconsistency in this system.</td>
<td>5 - score</td>
</tr>
<tr>
<td>7. I would imagine that most people would learn to use this system very</td>
<td>score -1</td>
</tr>
<tr>
<td>quickly.</td>
<td></td>
</tr>
<tr>
<td>8. I found the system very awkward to use.</td>
<td>5 - score</td>
</tr>
<tr>
<td>9. I felt very confident using the system.</td>
<td>score -1</td>
</tr>
<tr>
<td>10. I needed to learn a lot of things before I could get going with</td>
<td>5 - score</td>
</tr>
<tr>
<td>this system.</td>
<td></td>
</tr>
<tr>
<td>Total Usability Score</td>
<td>Score sum * 2.5</td>
</tr>
</tbody>
</table>

multiplying them by 2.5. This yields a single score ranging from 0 to 100, which is relatively simple to comprehend by a wide variety of people from other fields who serve in project teams (Bangor et al., 2009; Lewis et al., 2009). The higher the score, the more usable the product is.

The reasons to adopt this method to measure the usability are many: the SUS is free and quick to administer and score. Therefore, it has been widely used in assessing the usability of a wide range of technologies (Bangor et al., 2008). Furthermore, it is capable of producing accurate results even with minimal sample sizes (as few as two users) (Sauro, 2017).

In the current Arabic questionnaire, the overall SUS mean is 64.76, which is leaning towards the high score of usability. Figure 7.3 shows the distribution among the responses with $n = 21$. The highest number of answers score between 61-70 (8 responses) with the second highest being between 71-80 (five responses). It is absolutely necessary to assess the usability of the Arabic version of Tawny-OWL in a variety of contexts and circumstances; the System Usability Scale is specifically designed for this purpose. The results of this analysis indicate that the Arabic version of Tawny-OWL maintains a high level of usability
throughout the experiment testing. The majority of participants indicated that the Arabic translation of Tawny-OWL was useful and that it helped them navigate what they perceived to be a difficult environment and scenario.

This concludes that the Arabic version of Tawny-OWL can be considered a usable system for Arabic developers, where they can use it to perform the construction steps to build ontologies.

Figure 7.3: Frequency distribution of SUS scores for 21 participants.

7.3 Microsoft Excel

As mentioned in Chapter 5, the tolerogenic antigen-presenting cells (tolAPC) Ontology has been developed using the spreadsheet catalog as a source of information. This accomplishment confirms the potential in the above criterion 7.1.

It must be pointed out that the author indicated in Chapter 5 that this tolAPC Ontology was the motivating example and was designed beforehand. So there is no restriction on how the Excel spreadsheet should be formatted. After that, the patterns were designed in the programming environment of Tawny-OWL, which uses the Docjure library\(^2\) to extract values from the spreadsheet. By achieving this, the author addressed that ontologies

\(^2\)https://github.com/mjul/docjure
can be built via extracting their values from the Excel spreadsheet using Tawny-OWL programming environment.

Thereafter, the author developed a small Amino Acids ontology using an Excel spreadsheet as a source of information\(^3\), which was also used as a wordified ontology in later user experiments. Then, this Microsoft Excel approach was demonstrated in UKON 2018, showing how the Excel spreadsheet acts as a part of the ontology source code with the ability to update it freely. Subsequently, the ontology changed accordingly by rerunning the Tawny-OWL source code.

This approach can be applied to any spreadsheet source of information, where an ontology developer has to design the program to extract the information as they were stored in the spreadsheet. That is because Tawny-OWL coding is a straightforward programming language based on Clojure, and it is relatively simple to learn and use for developers from any programming background.

However, a generalization of this framework is needed to apply pre-defined patterns in spreadsheets and generate an ontology. This has been addressed by extending the Tawny-OWL tool Bubo\(^4\) to include the mechanism of reading spreadsheets information into an ontology. Chapter 5 described the process of this mechanism and showed its validation.

### 7.4 Wordified Ontologies

The author hypothesized in Chapter 6 that the wordified ontology in this study is readable, understandable, and familiar to domain users. Here, the author seeks to test the readability of the ontology and the ability to understand and extract information from it. Therefore, the author introduces two stages of testing: the first stage is to get the first impression and experience of this new format of the ontology. The second stage is to test the ability of users to work with the ontology document independently and measure their performance in comprehending and extracting information from a wordified ontology. The author compares users’ performance using a wordified ontology with an

---

\(^3\)https://drive.google.com/open?id=1QX5G29hV9N8bwL_s-Tdu4IA7CU-jy3nJ

\(^4\)https://github.com/phillord/tawny-bubo
ontology presented using Protégé and their preferences. The following sections present the evaluation process in detail.

**Ethical Approval**

The study was reviewed and approved by the SAgE Faculty Research Ethics Committee at Newcastle University (18-BLF-009).

**Preparation steps of the test,**

- Used one ontology for each stage; our version of the pizza ontology\(^5\) in Stage 1, and Amino Acid ontology in Stage 2.

- Created five sets of errors, each with two types (logical and non-logical).

- Made five versions of the ontology with all five sets of errors and represented them in Protégé and Microsoft Word.

- Arranged multiple packs/files of the ontologies for every user to use during the session (i.e., Pizza-1-2, Pizza-2-4, Pizza-3-1), where numbers indicate the version of the error set which are different; so, a user will have two different error sets in two different formats.

- Distributed errors across the ontology to make this distribution as even as possible.

The evaluation process has been designed as a series of testing sessions that users complete in two stages which is described in the next section.

### 7.4.1 Stage 1: Empirical Test (Controlled)

All experiment sessions were prepared, controlled, and observed on the Newcastle University campus, which served as the primary location for the study.

\(^5\)[https://drive.google.com/open?id=1uuea_OCJDTCpHq_NK301jSsoED0iHsMv]
7.4.1.1 Recruitment Procedure

Participants were approached using the Newcastle University mailing list. They were recruited from within the University, and none of them had previously worked with a wordified ontology. There were 11 participants in total, all Ph.D. students.

7.4.1.2 Experiment Design and Protocol

This process of evaluation is aimed at letting users read and explore the ontology and find errors. The objective is to discover the readability of the ontology and how users comprehend and interact with different formats of it.

Experiment protocol

Participants started by signing a consent form for participating in the research. They have also been informed that all screen activities, including any spoken words, will be recorded and that all information will be treated confidentially.

The researcher briefly introduced the objectives of the experiment and a brief overview of the software verbally. Then, participants were given a written explanation of the software and the tasks required from them during the test session (script in Appendix A.1).

After that, arbitrary time slots were arranged to reach the participants smoothly. If there were more than one participant in a session, they worked individually, and each participant started the test with a different ontology format. Accordingly, individuals within the same session had two randomly selected error sets: one for each format, with a different error set between formats (i.e., they did not see the same error set twice). The instruction sheets were distributed accordingly to help start the test. Participants were given 15 minutes to be spent on each format; the facilitator monitored this.

The main tasks were to read the ontology in the first format, find the errors, and suggest a fix for each error by writing their thoughts on the sheet provided. The same protocol applied to the other format of the ontology, all within the time specified. After finishing, the participants were asked to fill in a feedback form about their experience and overall opinion
regarding the evaluation session. These experiment paths are illustrated in Figure 7.4 where the participant went through one of the two possible paths described earlier. During the experiment, the facilitator remained silent to avoid any influence on the reactions/responses of participants.

The outcomes of these sessions aided in design the subsequent stage of remote evaluation trials. The results regarding this stage are mentioned implicitly in Section 7.6.

### 7.4.2 Stage 2: Remote Test (Semi-Controlled)

After receiving feedback from Stage 1 experiments, preliminary opinions about the user experience with the ontology were formed. Subsequently, a decision was made to run the same experiments remotely, with some changes. This remote experiment also allows participants to work independently from any direct observation.
7.4.2.1 Recruitment Procedure

For the remote evaluation, some participants who had a background in ontology development were recruited informally through the academic network. Therefore, initially, the participants were asked to determine their expertise in ontology construction and usage. This question helped categorize the participants based on their experience level, enabling the researcher to compare their preferences and performance. There were 25 participants in total.

7.4.2.2 Experiment Protocol

In order to enable participants to conduct the experiments remotely, few adjustments were made to the previous experiment, as follows:

- Developed web pages to be accessible from anywhere with thorough information of experiment purposes and the instructions of test tasks.
- Added some questions, where participants had to extract information from the ontology to find the answer.
- This led to the number of errors being reduced in the ontology (becoming 1 error).
- Uploaded the two formats of the ontology in the cloud storage to be shareable/downloadable with/for the participants.
- Participants used either Protégé (installed beforehand) or WebProtégé, based on their choice, and either local Word or Google Docs to see the wordified version.
- All responses were saved as web form data which could be accessed at any time.

Even though the experiment paths remained the same as the paths illustrated in Figure 7.4, however, in this stage, participants managed their time during the test with no direct interaction with the researcher during the session.
7.5 Qualitative Data Analysis of Wordification Experiments

The qualitative data in the feedback were obtained from either short answer questions or additional comments. This data was analyzed with nothing significant to be considered regarding the experiment. Some of the short answers are mentioned in the next section with some more text included in Appendix A.3.

7.6 Results of Wordification Experiments

7.6.1 Stage 1

As mentioned earlier, all participants in this stage were Ph.D. students who have never worked with ontologies before our experiments. However, they did interact reasonably with the ontology, which resulted in the following reading and error detection results, the difficulties encountered during the experiment, and their general comments about the test.

7.6.1.1 Reading the Ontology

When participants were asked which reading format they preferred, they predominantly preferred Protégé, where 7 out of 11 participants selected this option (see Figure 7.5). This finding implies that the most useful approach to reading ontology is one in which the reader is accompanied and supported by a GUI, that making it easier to read. The format of Protégé which is described in 2.3.3.5, allows users to build and understand the details of any domain’s ontologies. Protégé is an excellent tool for managing massive ontologies and offers a simple graphical user interface for editing, viewing, and validating those ontologies. According to this study, the choice of Protégé is due to its user-friendly interface, which makes it more conducive to reading ontologies. Additionally, this provides more evidence in favor of the claim that Protégé is a suitable choice for reading ontologies. Most of the participants who opted to read the ontology in Protégé, mainly focused on the hierarchical visualization of the ontology classes. Nevertheless, informal observation revealed that they have spent most of their time clicking and expanding the list, and they could not discover
more errors. This is also supported by the fact that Eze et al. (2021) reveals that there was a positive correlation between any software support and its use. Where user opinions regarding the usefulness and simplicity of any software tool, and perceptions of any software tool’s acceptability and likelihood of use acted as positive mediators. This implies that the ease of use of any software product would increase its use, and that particular software would be favored and preferred to use, as in the case of Protégé.

![Figure 7.5: Preference of participants in reading the ontology.](image)

### 7.6.1.2 Finding Errors

As previously stated, the amount of time to work on each format was restricted. Initially, it was ten minutes; but after receiving feedback from the participants, the time was increased to 15 minutes. This change of time did not affect the overall performance in finding the errors. Table 7.4 displays the number of errors discovered in each version; it is noticeable that increasing the time by five minutes did not help to discover more errors, especially in Protégé. However, there was a significant improvement in wordified ontology from only a couple of errors discovered, to a double of four errors in total. This improvement can be interpreted as the participants’ experience using Microsoft Word software is more advanced than using Protégé. Additionally, skimming and scanning text probably provides an easier way to spot mistakes, compared to exploring and clicking on the Protégé screen.
Table 7.4: Error finding performance in both versions of the ontology during the controlled experiment.

<table>
<thead>
<tr>
<th>Participants</th>
<th>Number of discovered errors in wordified ontology</th>
<th>Number of discovered errors in Protégé</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>0</td>
<td>10 minutes</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td>1</td>
<td>0</td>
<td>15 minutes</td>
</tr>
<tr>
<td>P9</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>P10</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>P11</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Total errors</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

### 7.6.1.3 Difficulties and Comments

When the participants were asked about the difficulties they faced in performing the tasks at the experiment, five participants indicated that time was an issue, and they needed more time to perform better in finding errors. This suggested that the experiment’s time should be extended. However, the goal was to avoid consuming too much time by the participants and provide optimum test time that helps in recruitment.

Few others found it a difficult task because they had never seen or used an ontology before. This challenge helped to indicate how many skills are required to interact with the wordified ontology, which seems to be a minimum, as all participants could browse and read the ontology, with some finding a few errors. This is due to the familiarity of dealing with Microsoft Word Document as a popular text application.

In general, there were no complaints about the experiment; furthermore, no one has found anything particularly problematic, which might have biased the results. The only issue was regarding the time, as one participant said, "You can increase the time limit to 20 minutes or add some more hints in the presentation before the test." , while another one
suggested showing the error’s location by expressing, "It would be better if you emphasis at the beginning that the position of the error". This indicates that the temporal constraints imposed by the experiment’s activities are causal rather than coincidental. Participants expressed frustration with their lack of time and stated that they needed it to improve their error-finding abilities. This sparked the idea to extend the experiment time. However, it was crucial to ensure that participants did not have to spend a lot of time in the lab in order to recruit as many people as possible. This is supported by Sauro et al. (2016), who discovered that 'Task time' reveals how much time a user spends on an activity, and that activity is frequently the amount of time it takes users to successfully complete a given task scenario.

Participants who preferred Protégé, mainly pointed out the hierarchical visualization of the ontology, 'In the software the hierarchy was very clear', which is undoubtedly an advantage of Protégé. On the other hand, some found that Tawny-OWL code script is easy to read in the wordified ontology, and also, the documentation attached had helped them to learn about the ontology. This variant demonstrates the participants’ diverse learning characteristics.

For further reviewing those opinions, excerpts of participants’ answers and comments are included in Appendix 1.3.1. Generally, the subject’s unfamiliarity with ontology may be blamed for the resulting misunderstanding. Overall, the trial was well-received, and no major issues were discovered.

7.6.2 Stage 2

Earlier in this chapter, the author mentioned the three criteria which were used to test the wordified ontology: reading, understanding, and editing. In this section, the author categorizes the results and describes the findings based on these criteria: initially, by showing the characteristics of the participants, followed by describing the results using the criteria mentioned above.
7.6 Results of Wordification Experiments

7.6.2.1 Experience Level of Participants

The experience level of the subjects with ontology construction and usage is depicted in Figure 7.6. 32% and 40% of those subjects are 'Somewhat Familiar' with ontology usage and construction, respectively. Approximately, 16% and 20% of the participants consider themselves to be at the 'Mastery' level in ontology usage and construction, respectively. Therefore, the experience was reasonably distributed across a range of different expertise levels, which enabled to obtain a representative sample size that would not have been possible if the test had been restricted to experts only. This is probably reflective of the ontology development community, which also has a range of expertise levels.

This result supports the idea that the levels of expertise of the participants in ontology construction and usage are sufficient to support the generalization of the study. This might be appropriate within the context of ontology application in general. Additionally, it was found that this experience level is distributed fairly across a range of different competence levels, which allowed the research to produce the desired results. Smith (2008) suggests that respondents are more likely to respond to survey requests if they are experts in the field. In a similar vein, Lindstädt et al. (2020) show that it is preferable to aggregate expert responses using the median or modal response rather than the mean in order to correctly understand the frequencies of the expert’s responses. This is true despite the fact that the mean is statistically the more accurate of the two, especially when the topic being researched is complicated. This further illustrates the significance of the findings of the current study and how they can be used to draw valid scientific conclusions. As a result, the findings alleviate any concerns regarding the likelihood of missing less important issues that must be extracted.

7.6.2.2 Reading Ontologies

The term 'reading' the ontology in this thesis refers to the process of exploring and browsing the ontology. Because of the test on the wordified ontology, the term 'reading' is used as it has more text and documentation of the ontology.
Chapter 7

7.6 Results of Wordification Experiments

(a) Experience level in ontology usage.

(b) Experience level in ontology construction.

Figure 7.6: Experience level of the participants.

Generally, over 60% of the participants find that the ontology is easy to read in both representations, as shown in Figure 7.7. Therefore, the conclusion here is that Protégé and wordified ontologies are similar in their level of readability. Based on these findings, it appears that the most effective method of reading ontology is one that simplifies the task of exploring and browsing the ontology so that it can be read more easily. In other words, the ontologies provided for this research provide users with a method by which they can develop and grasp the specific details of the ontologies in a way that is more understandable and easier to read. Since this study employs Protégé and wordified ontologies, it is possible to confirm that both of these types of ontologies are more user-friendly, making them more conducive to reading ontologies. Furthermore, they provide supplementary supports for users because they have determined that it is easier to read; this means that users are entitled to benefits using these representations. The research by Eze et al. (2021), which asserts that a user’s perceptions of the utility of any software product depend on how simple it is to use, can also be used to support this conclusion. This is also supported by the Technology acceptance model (TAM), which shows that "ease of use" influences intent to use any software system (Han et al., 2021). This lends credence to the previously stated assertion. As a result, the findings of this study are quite significant, and they also serve as a criterion to demonstrate that the ease with which ontology can be read will lead to an increase in its use.
7.6 Results of Wordification Experiments

(a) Readability of the wordified ontology.

(b) Readability of the ontology in Protégé.

Figure 7.7: The ease of reading the ontology results.

7.6.2.3 Understanding Ontologies and Finding Errors

Although the wordified ontology was designed with a comprehensive text to explain the ontology construction, when participants were asked about their satisfaction level in understanding the construction flow of the ontology in each format, 40% had a ‘Neutral’ opinion with the wordified ontology where also 40% could understand the construction flow in Protégé. Looking at the satisfaction level with wordified ontology in Figure 7.8a, nearly half of the participants are either ‘Satisfied’ or ‘Strongly Satisfied’, while it is nearly two-thirds with Protégé (see Figure 7.8b). It seems a fair inference that the overall presentation in the wordified ontology provides a clear and easy to understand how the construction of the ontology flow is throughout the document.

(a) Understanding flow in Wordified Ontology

(b) Understanding flow in Protégé

Figure 7.8: Understanding the construction flow of the ontology results.

In this experiment, the participants were asked to find one logical error inserted in the
ontology; this task appears difficult, as seen by the Stage 1 findings, in which participants were merely asked to discover errors and struggled to find all of them. The majority of the participants in this study had only occasionally used an ontology, so they found the task to be quite challenging. Consequently, it is reasonable to assume that some errors could occur. However, the tested ontologies could be browsed and read by all participants, while other did find a few errors.

The results of this task are pretty similar in both parts; looking at Figure 7.9, there is nearly the same number of participants (nine and ten), either ‘Satisfied’ or ‘Strongly Satisfied’, with this task in wordified ontology and Protégé. A quarter is (strongly) dissatisfied when finding errors in Protégé while it is nearly a third in the wordified ontology. The reason could be the time constraints imposed by the experiments. Almost a third of participants were in the ‘Neutral’ condition.

![Graph](image)

(a) Finding errors in the wordified ontology. (b) Finding errors in the ontology using Protégé

Figure 7.9: Results of finding errors in the ontology.

In order to ensure that participants comprehend the ontology correctly, they were asked to extract some information by answering some questions. The total of ten questions were quite similar in each format. These questions are included in Appendix A.2.

The majority of the questions were answered correctly (80% or above) in both formats. A brief outcome of these answers is shown in Table 7.5. One fascinating outcome here is finding the error; 30% of the participants successfully discovered the error in the wordified ontology. In contrast to that, only 4% of the participants detected that error in Protégé.
A possible explanation might be that the Tawny-OWL script is easy to read, which helps detect the error, especially with the explanation text combined. Unlike Protégé, which has a Graphical User Interface (GUI), Tawny-OWL lacks a source code display, and users rely entirely on visual representation and sometimes reasoners to detect logical errors. This finding implies that errors can be successfully identified, but the number of errors is not large enough to cause a serious problem even if they are discovered. Even if using 'wordified' requires more opportunities to locate faults, Protégé has successfully coupled the contracts with the graphical user interface (GUI). This point might be defended by pointing to the elaboration that was given on user satisfaction. In addition, these findings may be explained by the fact that users are able to discover the errors of reading ontology. This shows that errors can be easily detected and substantially minimized through effective interaction. In doing so, reading ontology can be used more effectively and have a greater overall impact.

Table 7.5: The percentages of correct answers in both wordified ontology and Protégé.

<table>
<thead>
<tr>
<th>Questions</th>
<th>Percentage of correct answers in wordified ontology</th>
<th>Percentage of correct answers in Protégé</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ontology name</td>
<td>96%</td>
<td>96%</td>
</tr>
<tr>
<td>First defined class</td>
<td>57%</td>
<td>88%</td>
</tr>
<tr>
<td>Object/Annotation properties</td>
<td>91%</td>
<td>88%</td>
</tr>
<tr>
<td>Total number of classes in the ontology</td>
<td>4%</td>
<td>29%</td>
</tr>
<tr>
<td>Side Chain/Size of an Amino Acid</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Identifying subclasses of X</td>
<td>87%</td>
<td>79%</td>
</tr>
<tr>
<td>Finding an error</td>
<td>30%</td>
<td>4%</td>
</tr>
<tr>
<td>Adding a new Annotation Property</td>
<td>83% (^c)</td>
<td>NA</td>
</tr>
</tbody>
</table>

\(^a\) The correct answer regarding Annotation properties dropped down to 83%.

\(^b\) The correct answer regarding the size of a specific Amino Acid dropped down to 96%.

\(^c\) 78% suggested the correct place for the new property.

An overall view of participants’ performance in answering the questions is shown in Figure 7.10. In the wordified ontology (see Figure 7.10a), most of the questions were reasonably answered correctly, apart from question five and question ten, in which they ask about the
total number of classes and finding the error, respectively. It was expected that these questions are difficult ones.

On the other hand, answers in the Protégé version were similar (see Figure 7.10b). Additionally, the three questions that were mostly answered incorrectly were about the total number of classes, finding the error, and suggesting a correction.

These findings can be interpreted as participants being able to extract information from either version easily; however, error detection is difficult, especially with the time restriction. Regarding the question on the total number of classes, in the wordified ontology, it is clearly difficult to find the correct number since the user needs to count each statement that has a definition of a class. Contrarily, in Protégé, there is a statistic matrix with the information about the ontology entities. However, most of the participants had no experience with the software and the ones who answered correctly, did it with the third question. This is unlike wordified ontology where only one participant answered that question correctly.

(a) The number of correct/incorrect answers in the wordified ontology.

(b) The number of correct/incorrect answers using Protégé.

Figure 7.10: An overview of the correct/incorrect answers.

7.6.2.4 Editing Ontologies

In a wordified ontology, users can add comments and annotate the text using the Track changes facility in Word. The participants were asked to turn on Track changes before
they perform any changes in the ontology. This facility helps save time and effort for the ontology developers when updating the source code of the ontology accordingly.

As shown in Figure 7.11, most participants are either 'Satisfied' or 'Strongly Satisfied' with their performance in the editing tasks during the test. This satisfaction level indicates that modification of a wordified ontology is opted by users as compared to modification in Protégé. To justify that preference, the participant, specifically, the non-experts are more familiar with Microsoft Word software than Protégé. Moreover, mimicking Tawny-OWL is fairly approachable with the source code being visible. According to this finding, the majority of editing task performance is correlated with the ability to interact with the ontology fluidly and to some degree. This finding has been supported by earlier studies that used an experiment in which participants were given a Graphical model of the Family ontology and asked to complete ten editing tasks on it. In that study, it was discovered that users felt about as at ease reading ontologies written in the novel Graphical language as they did reading ontologies written in traditional textual languages (Lembo et al., 2022).

![Edit Wordified Ontology](image1.png)  
![Edit the Ontology in Protégé](image2.png)

(a) Editing the wordified ontology.  
(b) Editing the ontology using Protégé.

Figure 7.11: Results of editing the ontology.

### 7.6.2.5 Overall Results

As mentioned earlier, the aim was to test how users interact and comprehend the wordified ontology as a typical Word document. The results showed that most participants expressed satisfaction with 36% and 20%, respectively, on the clarity of the ontology. On the other
hand, nearly a quarter (24%) struggled with clarity. The rest have no opinion. These findings can be seen in Figure 7.12.

![Graph A](image1.png)  ![Graph B](image2.png)

(a) The clarity of the wordified ontology. 
(b) The clarity of the ontology in Protégé.

Figure 7.12: The clarity of ontology structure results.

In Table 7.6, the preference of users plus their performance in finding errors are shown. Both formats have a favorable readability preference with over 60%. In addition, finding errors is a challenging task, which is clearly shown, with preference being close between the two formats (Protégé being somewhat more preferable). Since Protégé has a graphical interface with a hierarchical presentation of the classes in ontology, it showed preferable results in understanding the structure’s flow of ontology and clarity. Similarly, the editing task has a close result, with wordified being slightly ahead. The results of the experiment show that it was successful, taking into account that it aimed to evaluate the clarity of the ontology in both forms; the wordified form (Word document) and Protégé (A graphical interface with a hierarchical style). On the other hand, roughly one-fourth of the sample struggled to understand what was being said. The remaining members of the group have no point of view. The preference between the two forms is not significantly different, with Protégé being picked slightly more frequently. Despite the fact that both wordified and Protégé have a high readability preference, the process of locating errors remains difficult. It’s possible that Protégé’s exceptional performance is due to the fact that it has a graphical user interface that organizes the ontology classes in a hierarchical manner. Similarly, the results of the editing task are very close, with wordified emerging victorious.
Table 7.6: Overall positive percentage of user preference between the wordified version and Protégé.

<table>
<thead>
<tr>
<th></th>
<th>Wordified Version</th>
<th>Protégé</th>
</tr>
</thead>
<tbody>
<tr>
<td>Readability</td>
<td>60%</td>
<td>68%</td>
</tr>
<tr>
<td>Understanding flow</td>
<td>48%</td>
<td>60%</td>
</tr>
<tr>
<td>Error finding</td>
<td>38%</td>
<td>42%</td>
</tr>
<tr>
<td>Editing</td>
<td>54.2%</td>
<td>52.2%</td>
</tr>
<tr>
<td>Clarity of structure</td>
<td>56%</td>
<td>64%</td>
</tr>
<tr>
<td>Correct answering</td>
<td>71.7%</td>
<td>66.7%</td>
</tr>
</tbody>
</table>

### 7.7 Discussion and Conclusions

This section covers the presentation of the main findings as well as the interpretation of the evaluation of the research. The characteristics of the respondent are considered, which is an important part of the evaluation.

The conclusion drawn from these results is that participants found a wordified ontology to be readable and understandable, similar to a Protégé environment.

Consequently, the overall conclusion is that the wordified ontology, as a format to represent the entire ontology, is helpful and competitive with Protégé. In fact, the user performance and options in wordified format are comparable to the performance in Protégé. This format affirms the benefit of using the wordified version as an integrated version of the ontology with its documentation.

Although reading wordified ontology is pretty convenient, some participants commented on the lack of certain features, e.g., "However, a visualization encompassing the entire structure would have been quicker to understand". While users missed a hierarchy browser in the wordified version, Microsoft Word has the feature of an outline browser, which users could have used, and the wordification process could support better. This would help users repurpose outline viewing as a replacement for a hierarchy browser. This is significant because it indicates that the study has a sufficient number of respondents to reveal not only the prevalence of the use of standard "Office Tools and Arabic Script
in Ontology Development", but also the genuine nature of the interaction that motivates domain specialists and ontology developers to work together, which means that the research community can draw reliable conclusions from these findings. Additionally, various studies confirm the dominance of female respondents and the superiority of specialists over non-experts, which both contribute to this conclusion drawn from the respondent collection. Above all, on average, both formats used for this study, were better for different purposes. Typically, it is noticed that text form is advantageous when users see examples in the source code of Tawny-OWL, because seeing the code with explanation helps to copy it. Furthermore, many users added a new annotation property correctly, whereas with GUI, being able to see an annotation and adding a new one is not necessarily the same. Even when they can visualize and browse easily with icons and menus, they require a good understanding of the software to make correct modifications. This challenge is a common experience using GUIs.

According to the presented outcomes, the wordified version of ontologies is appealing, and other improvements to promote users’ engagement in ontology development should be considered. On the other hand, incorporating some form of narrative documentation within Protégé would certainly advance its features. There should be no issues with the experiment validation for precision in this matter. The proportion of appropriate responses to the open-ended questions further supported the conclusion since both the response rate and the extracted responses are valid. This means that it is safe to trust the data derived from the percentage of respondents. In this manner, the reliability of both the survey respondents and the survey’s raw data can be evaluated. The finding suggested that the System Usability Scale expressed purpose of analyzing Arabic Tawny-OWL’s usability in a range of contexts where Tawny-OWL in Arabic remains highly usable throughout. Most of the participants thought the Arabic version of Tawny-OWL might be helpful, and when they tried it out, they found that it helped them make their way through what they had previously considered to be an intimidating situation. As a result, within the same framework, the findings of this study imply that the method of reading ontologies that is the most effective is one in which the reader is guided and supported by a guide, so
making it simpler to read. This is supported by the research conducted by Jarrar (2021), which included a comprehensive analysis of the content of 150 Arabic-language lexicons. Additionally, these lexicons are being mapped and expanded with the assistance of the ontology as supported by a guide, which makes it simpler to read.

### 7.8 Summary

In this chapter, the evaluation procedures were conducted to evaluate the proposed approaches in developing the ontologies. While incorporating Excel spreadsheets and Word documents to generate new ontologies, the focus was on how various users interact to develop ontologies. Therefore, the evaluation experiments were designed to interact with users, assess their performance, and obtain feedback on the Wordified ontologies.
Chapter 8
Discussion

Contents

8.1 Introduction ...................................................... 180
8.2 Development Approaches ........................................ 181
  8.2.1 Internationalization Environments .......................... 181
  8.2.2 Re-purposing Literate Programming ....................... 182
  8.2.3 Utilizing Office Tools and Ontology Building Environment 183
8.3 Research Outcomes and Contributions .......................... 185
8.4 SWOT Analysis .................................................. 190
  8.4.1 Analysis of Developing Ontology ............................ 191
  8.4.2 Analysis of Readability and Editability ...................... 193
  8.4.3 Analysis of Evaluation ....................................... 194
8.5 Limitations and Restrictions .................................... 196
  8.5.1 Internationalized Ontologies ................................. 196
  8.5.2 Document-centric Ontology Development .................. 196
  8.5.3 Reusing Existing Ontology .................................... 197
  8.5.4 Evaluation Stage ............................................. 198
8.6 Future Work .................................................... 198
8.7 Improving the Collaborative Ontology Engineering Landscape .... 200
Abstract

This chapter discusses the vision of integrating the standard office tooling in the ontology development process and the philosophy behind producing multiple visualizations of the ontology that serve a wider range of users.

8.1 Introduction

An ontology is a formal representation of knowledge, computerized as a set of concepts and relationships. Computational abilities of ontologies aid in the sharing and reusing of knowledge within a domain. This capability enables the understanding and operation of a domain of knowledge based on the ontology’s purpose.

Ontologies are widely used in various domains; they represent heterogeneous data and link various information from different resources, systems, and databases. Therefore, they are valuable models, especially to implement interoperability and knowledge management at different levels 1.

Many methodologies have been applied to build ontologies, depending on the purpose and the services provided by these ontologies. In this thesis, new ways have been investigated to bring domain expertise, aimed chiefly at biologists and medics, into the process of ontology development, by allowing them to use their preferred tools in the process of building ontologies.

It has been discussed in Chapter 2, how the ontology construction process, as a collaborative process involving various users and tools, has its own set of challenges. Furthermore, the interaction between the various parties, with their diverse set of tools and backgrounds, is a fundamental goal in order to facilitate this type of collaboration and, ultimately, the successful development of an ontology, among other things.

Likewise, any software development also contains potential challenges/difficulties, depending on various factors, such as the purpose and complexity of the software. Thus, ontologies are developed using user-centric tools and reusing and re-purposing software development experience; this is referred to as a document-centric approach, using standard office tooling.

1https://www.ontotext.com/knowledgehub/fundamentals/what-are-ontologies/
8.2 Development Approaches

Figure 8.1 depicts the main areas that have been explored in order to contribute to the creation of ontologies. Since the primary emphasis is on user engagement, the development process incorporates the most commonly used office tools (Excel and Word). Users are more likely to read and write ontological knowledge in familiar tools/forms, encouraging collaborative work with ontology developers.

Following that, various strategies for bridging the tooling gap between domain experts and ontologists are discussed.

8.2 Development Approaches

8.2.1 Internationalization Environments

Even in HTML, to understand an ontology, the domain specialist needs to read it. The goal was to investigate if an entire ontology development environment could be internationalized into a foreign language. It has been shown in Chapter 4 that it is possible to translate Tawny-OWL to support an alternative language, specifically Arabic, a right-to-left language. This translation is obviously dependent on the Integrated Development Environment (IDE) working correctly. As a result, the experience with different IDEs varies; some editors worked just well, while others are unusable. This variance is perhaps not surprising as the bias in favor of English in software engineering is noticeable. However, some Clojure editors allow users to edit right-to-left Tawny-OWL files. Still, this leaves users in a programming environment that
is unlikely to be familiar or comfortable.

Previously, in Section 7.2, the survey on using the Tawny-OWL Arabic commands revealed that, after being exposed to simple examples, Arabic users could write accurate commands without any active monitoring. This demonstrates how using a native language can aid in the learning process and speed up the creation process.

The Arabic version is a starting point for internationalizing such sophisticated ontology construction environments. This can reach a broader range of users and can be applied to any other language to serve the same purpose. As a result, if users can build the ontology in their native language, they will be more likely to learn and contribute to ontology development.

### 8.2.2 Re-purposing Literate Programming

One way to bring ontology development into standard office tooling is to improve the presentation of the source, so that it is more readable for domain specialists. This is now a common-place idea in software engineering, with integrated documentation tooling such as JavaDoc being the norm for most languages, and indeed, such tools have been implemented for ontologies with OWLDoc. In terms of motivation, the concept is similar to the concept of literate programming.

In Chapter 6, the literate programming approach has been facilitated in order to produce complete and well-documented ontologies. The Cloud Ontology\(^2\) is an example where the documentation has been integrated within the ontological source code blocks in one final product of the ontology, which can be displayed as HTML pages or as a Word document. In contrast to tools such as JavaDoc or OWLDoc, the end product is a narrative document that can be read from beginning to end, rather than a set of interlinked pages with a navigational structure (more of a story than a manual). This style is perhaps similar to the original concept of literate programming, emphasizing the narrative structure.

According to the findings in Chapter 7, the majority of participants agreed that the ontology wordification format is coherent, simple to interpret, and understand. They can also

\(^{2}\text{Fully described in Chapter 6.}\)
change and update the text as needed since they work in a familiar text environment.

Utilizing familiar tools, particularly wordifying ontologies, helps enhance the readability of ontologies and makes them more accessible to both experts and other users. This improves the ability to read and understand them, plus participate in the creation process through text editing in a familiar environment.

While the Cloud Ontology has been substantially wordified, the same technology of wordification also applied to construct Amino Acid Ontology, which was initially developed as a technology demonstration. This ontology can be explored in Appendix C.2.

Turning the ontology into HTML produces an end-format that is read-only. However, it is also possible to generate other formats, principally Word documents, which can be easily updated and manipulated by domain users. This wordification will be considered in Section 8.2.3 as well.

8.2.3 Utilizing Office Tools and Ontology Building Environment

Chapters 5 and 6 described the proposed work towards integrating Excel spreadsheets into the ontology development process (ODEW) and the wordifying technique of ontologies representing as Word documents, respectively. In both frameworks, unique procedures have been developed, allowing the most popular user applications to cooperate smoothly in the ontology development pipeline, which is illustrated in Figure 8.2.

Specifically, a spreadsheet has been converted into an ontology using highly programmatic and pattern-driven ontology development (see Chapter 5). Critically, the spreadsheet remains a source for the ontology; domain specialists may update it at any time, and changes will propagate to the final ontology. In this approach, a new ontology describing immunological cell lines was built by instantiating ontology design patterns written programmatically, using values from a spreadsheet catalog. This method employs a spreadsheet that domain experts developed. The spreadsheet is unconstrained in its usage and can be freely updated, resulting in a new ontology. This spreadsheet employment provides a general methodology (Ontology Development Excel Workflow (ODEW)) for ontology development,
using data generated by domain specialists in spreadsheet formats.

Chapter 6 described the possibility to wordify ontologies as a readable and editable format from the natural ontological source and its embedded comments. This format had been translated to HTML for reading; the environment provides rich hyperlinking and active features such as hiding the source code in favor of comments. Additionally, the reason behind the wordification of these ontologies is to facilitate and enable editing tasks easily. In addition, wordified ontologies are produced explicitly as a novel instrument for communicating with domain experts. They can easily read and edit the document using the 'Track changes' option, which helps to give the ontology developers their comments and modifications. As a whole, this should provide a significant new route for collaboration between the ontologists and domain specialists.

Now, if any changes were to occur in the ontology through the Excel spreadsheet or the wordified version, ontology engineers need to check the source code to incorporate the changes performed by the experts and then regenerate a new version of the ontology. Automation in the reverse direction is not applied since human manipulation is necessary due to various factors such as the different tools and the modeling decisions.
8.3 Research Outcomes and Contributions

The general and primary findings of this research highlight the importance of achieving the objectives of the research by addressing the questions that were posed during the research, as well as the interpretation of those findings. Data analysis was used to accomplish this. Five research questions have been set in Chapter 1 to investigate the research problem and provide feasible solutions:

**RQ1** is about presenting ontologies globally, specifically in the Arabic language. The Tawny-OWL environment has been converted to Arabic, and commands have been translated and evaluated by Arabic users. This resulted in a programming environment that was suitable for developing ontologies using Arabic terminologies and commands. This achievement has been discussed in Chapters 4 and 7. The findings of the study that was related with this have demonstrated that the Arabic Ontology is a view of the knowledge that the Arabic terminology transmit that is centered on concepts. In the sense of the formal theory of sets, concepts can be thought of as groups of particulars, or people. This finding, which was reached using an operational usability measure, suggests that Arabic ontology is useful for designing user interfaces. That means it indicated that using Arabic ontology, the system usability scale demonstrates that Arabic Tawny-usability OWLs can be used in a variety of settings, and even so, Tawny-OWL in Arabic continues to be highly useful throughout. The results of the experiment also suggest that the Arabic version of Tawny-OWL could be useful. When users used it, they discovered that it assisted them in navigating through a circumstance that they had previously regarded as being daunting.

The findings of this study suggest that the method of reading ontologies that is the most effective is one in which the reader is guided and supported by a guide, making it simpler to read as a result. This is because within the same framework, the findings of this study imply that the most effective method of reading ontologies is one in which the reader is guided and supported by a guide. In addition to this, this has provided a foundation for what the research intends to measure and an answer to the **RQ1**. In a similar vein, this can be justified by the work of Jarrar (2021), which suggests that the effectiveness of read-
ing an ontology ought to be accompanied by a guide in order to make it easy to read and absorb. In addition, it was found that the text in Arabic that was acquired in sentences can be used for an ontology concept in order to extract and weight the semantic domain features by considering their levels in the ontology tree and their frequencies in the dataset to compute the overall polarity of a given textual review based on the importance of each domain feature, in order to make it more clear for analysis. This was done in order to make the text more understandable (Khabour et al., 2022). When users are provided with the right ontology, it enables simpler engagement, permits meaningful communication, and allows users to reason about such information. Finally, organizing Arabic level ontology is a knowledge that is concentrated on extensional concepts. In contrast to other ontologies, such as those that do not contain specifics regarding their representation and that do not define concepts but rather define the meaning of the terms that are used to denote the concepts, this one does describe the meaning of the terms. In other words, concepts in the ontology serve as the primary building blocks, whilst terms are only lexicalizations of concepts; hence, a comprehensive ontology need to be characterized as a unit of knowledge that is produced by an exclusive combination of component parts. The results of this research confirmed the importance of exploring methods for adapting literate ontology to the standards defined by RQ2. Consequently, the goal has been met, and the findings can be understood to suggest that employing Arabic ontology is unique and amenable to incorporation into any ontology tool. In a similar vein, the findings of this study, along with their interpretation regarding the investigation of the transformation of existing semi-structured knowledge, such as Excel, into an ontology from the perspective of answering RQ2, highlight the significance of achieving the goals of reading ontology by a particular ontology platform that answers in RQ3.

The most popular Office applications – Excel spreadsheets and Word documents – are used to address the research questions RQ2 and RQ3 since the vast majority of domain users are familiar with these types of documents. To accomplish this, the ODEW (Chapter 5) was initially proposed by developing ‘The tolAPC Ontology’, which was a success. This accomplishment led to the generalization of the technique, using the ‘Bubo’ tool, by in-
cluding the digestion of Excel spreadsheets into the ontology building framework. Then, as discussed in Chapter 6, the process of wordifying ontologies began with the documentation of the Cloud Ontology using the Tawny-OWL source code, which was a successful implementation that helps in the wordification of other ontologies. Both office software have been checked and evaluated, with positive results, as detailed in Chapters 5 and 7. The finding of this study can be interpreted as a reflection of the reality that ideas and particular thought are both the products of already existing minds and, if they are to be conveyed in form of communication, their presentation should be as close as possible to the ideal of communication goal. In other words, if something is to be communicated, its presentation should be as close as possible to the ideal of communication target goal. The findings of this study can be summed up by stating that it was determined that the implementations of Ontology tools are effective and efficient, which enables the attainment of a communication goal. Because it is impossible for users to know what one another is thinking, it is abundantly clear that essential tools are required to facilitate communication rather than relying on the construction of concepts as thoughts, that is why ontology tools evaluated in this study, are found to be effective and efficient. In addition to this, the results shed light on how users construct and interpret domain-specific ontologies. A notable example of this type of tool is Protégé, which is able to manage massive ontologies in a simplified manner while still providing a user-friendly GUI for efficiently processing content. Protégé is an excellent example of this kind of application. In light of these findings, both RQ2 and RQ3 can be considered answered: the best approach to reading an ontology is one that streamlines the process of exploration and browsing to make it more accessible, and Protégé is the most straightforward method for interpreting ontologies in this regard, where users do not need to complete any complicated steps. Above all else, the utilization of the ontology tool is essential for both representations and comprehension. Based on the findings of this study, it has been possible to deduce that the ontology utilized for various purposes need to originate from a tool that is both effective and efficient. When users investigate Tawny-OWL’s source code, the benefits of textual representations become readily apparent. This is due to the fact that reading the code alongside its ex-
plation makes emulation much simpler. Whereas the vast majority of comments that users were able to successfully make have brought about an understanding that is crystal obvious. In addition, it was seen that even though the ontology representation may be perceived and browsed readily, the provided guidance that comes from the Ontology tool makes understanding more accurate. The interaction that takes place within the ontology domain focuses on the various ways in which it can be put to use to facilitate collaborative work between ontology developers and domain experts. What is the most important fact that has emerged from this research in relation to RQ4 and RQ5 on the topic of interaction relies on the benefits of the usage. Despite drawbacks of iterative enhancement of ontologies to the various groups of people who are involved in the process of developing ontologies, this research has been able to establish. These are the questions that have been answered by this study.

RQ4 relates to the collaboration of domain experts and ontologists which is addressed through the use of methodologies in this thesis. This is possible and could encourage cooperative tasks in the ontology development process by introducing new mechanisms of using familiar environments (Excel or Word) to facilitate the interaction among different users. In these frameworks, domain specialists may communicate and supply semi-structured information to ontology developers to create and build the ontology as they organize the knowledge in spreadsheet format. Similarly, ontologists may produce a wordified ontology, tailored for those domain specialists to read through, and add or modify the information as needed. The most important question is how those who used the technology defined themselves in relation to it, as well as how those who used the technology defined themselves in relation to it. According to the findings of this research, the users and end users should be considered the unit of analysis. The primary concern that was addressed in response to RQ4 was the fact that its findings indicated that participants found a wordified ontology to be readable and understandable, much like a Protégé environment, and that those involved are experts. This was the key issue that was addressed in response to RQ4. As a result, the overarching conclusion is that the wordified ontology, as a format to express the full ontology, is beneficial and competitive with Protégé. The user performance
and options in wordified format, on the other hand, are on level with the performance in Protégé. The use of the wordified version as an integrated version of the ontology with its documentation is validated by the use of this format. Users are able to call attention to the fact that the ease of use of an ontology is the most important component of using an ontology; as a result, their recommendation to the developers lies with the fact that they must make sure the developers understand user needs in terms of less complexity in an ontology tool.

RQ5 refers to the pros and cons of constructing ontologies for various users using the proposed mechanisms. There are some benefits, including, but not limited to, the opportunity to collaborate in the ontology engineering pipeline using user-friendly tools. Furthermore, by incorporating shareable and cloud interaction facilities, which will be easier to use with the famous Office tooling, this will significantly save time for collaborators and speed up ontology creation. Nevertheless, domain users can feel constrained by the spreadsheet’s patterns. Ontologists may also manually implement any adjustments to wordified ontologies by monitoring the changes made by domain specialists.

Looking at the objectives of the thesis from Chapter 1, these objectives are fulfilled successfully, as will be explained:

- The RO1 is completed and described in Chapter 4 and evaluated in Chapter 7.

- The objectives RO2 and RO4 are accomplished in the background research of Chapters 2, 5 and 6.

- The RO3, are described in 4, 5, and 6, and can be fully viewed in Appendix C.2.

- The RO5 is fully described in Chapter 7.

Therefore, it is possible to state that the overall finding has been generalized for the standard Arabic ontology, which is the process of bringing attention to the research activity. This is significant because it indicates that the study has sufficient associated to the interaction domain specialists and ontology developers, as well as the tools for developing
ontology. This is important because it indicates that the study has sufficient associated to the interaction domain specialists. The research community now has access to a vital interactive medium due to the public release of Office Tools and Arabic Script in Ontology Development, from which they may reliably draw conclusions regarding the effectiveness and efficiency of their own work while acknowledging this study.

8.4 SWOT Analysis

SWOT Analysis is a method of assessing the strengths, weaknesses, opportunities, and threats of a desired project or activity. SWOT analyses are often presented in a form of a matrix with four distinct regions, one for each element. This presentation is helpful in that it presents a range of data in an accessible visual way. With this approach, it is possible to distinguish between project internal and external factors that could affect performance.

- **Strength**: Features that distinguish the methodologies in this context from other methods.
- **Weakness**: Aspects that the current methodologies are lack or has limited abilities.
- **Opportunities**: Actions that might be done to improve the outcome and advance the process of the development.
- **Threats**: this would include the risks and limitations of the current version of the process.

The purpose of conducting a SWOT analysis for this study is to conduct an evaluation of the study as a whole. This is after following the methodical flow of the investigation and obtaining the desired results. A secondary analysis of the full study is presented in this part of the report. This is due to the fact that a study that makes use of ontologies should provide a way to describe the finding in a meaningful scheme and manage the knowledge interoperability among many schemes. This has led to the current situation.

SWOT analysis provides more advantages than disadvantages when compared to other Ontologies tools during the course of building and employing an Arabic ontology. It can
efficiently investigate multiple Arabic ontologies simultaneously because of its multidimensional modeling and analysis capabilities, as well as its capacity to integrate data. A reduction in modeling complexity, an increase in connectivity, and improved interoperability do not result in a rise in costs when developing a SWOT analysis (Namugenyi et al., 2019). This study clarifies the interplay between Arabic ontology tools and their application, reducing any confusion that may arise throughout the developing and using process. This holds true despite the fact that quantifying the influence of any SWOT factor on the others can be challenging. Although SWOT analysis produces a great deal of information, some of it may be deceptive. Even though it may be challenging to manage a wide range of options, the present suite of Arabic-specific tools and integrated ontologies is well-equipped to do just that. Prioritizing issues and challenges is crucial for a smooth roll-out of Arabic ontology. This process can be helped and facilitated by solutions. Table 7.1 provided the representation of the SWOT analysis necessary for this study.

### 8.4.1 Analysis of Developing Ontology

#### 8.4.1.1 Strengths and Opportunities

The capacity to utilize an alternative language, specifically Arabic, which is a right-to-left language and which is supplied in the new ontology tool create stand as one of the major strengths of this research. Therefore, as shown in Tawny-OWL Arabic commands, it was
discovered that after being shown a few straightforward examples, users of Arabic were able to create accurate instructions without the need for any kind of active monitoring. Because of this, the construction of an Arabic version brings about the necessity of building and implementing ontologies that are capable of describing new fields of knowledge and incorporating them, along with the associated software tools that are required. In addition, the utilization of an ontology that is formally representing knowledge, in the form of a set of concepts and relationships, contributes significantly to the strength of this research. The ability of ontologies to be computationally processed helps to facilitate the sharing and reuse of knowledge within a particular domain. This capability allows for the comprehension and operation of a domain of knowledge based on the goal of the ontology. In addition, this can be related to the attribute of ontologies, which are extensively utilized in a variety of different areas for the purpose of presenting heterogeneous material in order to link it up with the Arabic version of diverse information drawn from a variety of resources, systems, and databases. Because of this, the power of the development lies in its capacity to create interoperability and knowledge management on several levels. This research paved the way for new methods to investigate and bring domain expertise into the process of ontology development by enabling domain experts to make use of the tools of their choice while constructing ontologies, which was made possible as a result of the successful completion of the system that was being developed.

8.4.1.2 Weakness and Threats

The absence of the appropriate software tools for the development of an ontology is the weakness that this research found in terms of the development of an ontology. These technical specifications need to be prioritized over any other idea. The weakness of the building ontology lies with technical specifications that need to be prioritized over any other idea. Nevertheless, after this, the deficiencies will be resolved, and then there will be an opportunity to create ontologies in order to provide new fields of knowledge. This is a benefit for which a new technology is being used, and it will help develop more effective ontology. In conclusion, the threat of constructing an ontology are dependent on coverage. That is to
say, there is a possibility that it will be unable to cover a large variety of implementations that are associated with different languages. In addition, when new knowledge domains are incorporated into ontology models, it is essential to ensure that the models are correct and consistent; yet, since diverse languages are used, this presents a significant challenge. Another problem is linked with correctness and consistency. In order to successfully design and implement ontologies that accurately reflect and include new areas of knowledge, it is required to have access to the appropriate software tools.

8.4.2 Analysis of Readability and Editability

8.4.2.1 Strengths

The readability of an ontology improves with the presentation and acceptance of more important ontologies, as well as with the increased frequency with which they are used. This indicates that this research has investigated the issue associated with the degree to which readability is enhanced to be proportional to the degree to which the ontology is presented in a manner that is not only simpler but also more capable of achieving high levels of ontological accuracy and consistency. In particular, because it is related with the adoption of the greater scope of the ontology application, and more importantly, because it is essential that the models of the ontology be consistent and correct, this is one of the strength of the study. The current research has also demonstrated that Editing Ontologies enable users to add comments and annotations, which afterwards enables users to make any modifications that are desired. When it comes to upgrading the source code of the ontology in accordance with new information, this facility helps save time and effort for the ontology developers. In addition, users are flexible while carrying out editing content, with the level that shows modification. This finding, which is related to the editing of ontology, suggests that the majority of performance in editing tasks is correlated with the capability of interacting with the ontology easily and having some level of mastery over it. This finding was made possible by the fact that ontology is being edited. In light of this, it can be deduced that this is also one of the research’s strengths.
8.4.2.2 Weakness and Opportunities

The inaccuracies and inconsistencies are the readability and editing flaws that need to be addressed first. This could be due to the fact that when reading ontology and if the sign or presentation is not throughout consistent, or if there are presentations that are not correct, this research found to set that out as a weakness. This could be exerted by the fact that when reading ontology and if the sign or presentation is not throughout consistent. In addition, the opportunities of readability that might be leveraged to overcome the weakness lie in the presenting logical summation of each bit-by-bit details material. This is the case because presenting logical summarization of the information is more easily understood. As a result, there is a possibility that there is still a threat that centers on the "misinterpretation". The fact that the signs utilized in ontology can be misunderstood could have been a contributing factor in the difficulties associated with interpreting the ontology. Because of this investigation, this research is able to draw conclusions about the circumstances described in 7.

8.4.3 Analysis of Evaluation

8.4.3.1 Strengths and Opportunities

The comprehension of the reading format and the provision of the developers with resources, which means that one must have a concept of the pattern for ontology workflow, are the factors that contribute to the strengths of the validation. Based on the evaluation that was carried out and given in chapter 6, this research was able to determine the requirements that must be met in order to receive that support. Because it is accompanied and supported by a guide, the preferred reading format of ontology, which is largely chosen Protégé, also makes it easier to read. This is one of the reasons why it is so popular. This information was obtained through the process of evaluation, which is why it is regarded as one of the most significant strengths of the research. When working with ontology, users should present evidence in the evaluation to support the notion that extra support for the clarification is required. Therefore, based on the user’s perspective, the ease of using
any Ontology product would increase its use, and that particular ontology would be the favored, and predominantly preferred to use like in the case of Protégé in this study.

8.4.3.2 Weakness and Opportunities

One of the limitations of the validation that is related with this study is the fact that, throughout the assessment, if there is an inability to examine inaccuracy, this affects the entire evaluation. This is one of the limitations. This is because there is no method by which an error could be determined when it is unable to be explored, and this is why this statement is true. However, a casual inspection revealed that users spend the most of their time clicking and expanding the list. Although they were unable to find any additional problems, the threat will exist in the areas where the faults cannot be located. Last but not least, this deficiency can be remedied by the utilization of the opportunities for validation. This research employs the users’ perceptions to get an opportunity to determine the users’ perceptions of their level of satisfaction with the reading of the ontology. The users’ perceptions are quite important, and this research leverages them to its advantage. As a result, one of the most significant opportunities that this research use, was to extract user perception in relation to the understanding of their level of satisfaction. It was discovered that the developers’ commitment to the process of constructing ontology is absolutely important in order to derived the satisfaction of the users regarding the use of the ontology. This was the conclusion drawn from the findings described above.

8.4.3.3 Threats

The Experience Level of the users who took part in the evaluation is one of the factors that poses a risk to the validity of the evaluation. The amount of experience that the participant has poses a risk when it comes to reporting the findings of the study. In the event that they did not have the experiences they claimed to have, it would have an impact on the overall findings of the research. However, for the purpose of this investigation, the participant’s experience was fairly dispersed across a range of different expertise levels. This enabled the researchers to obtain a representative sample size, which would not have been possible
8.5 Limitations and Restrictions

There are some limitations of the work in this thesis which are discussed here as follows:

8.5.1 Internationalized Ontologies

This version of the ontology development environment of Tawny-OWL would be narrowed to particular users of that Arabic language, as projected earlier in Chapters 4 and 7. This restraint means that this version of Arabic will be limited to a minor community of Arabic programmers. However, publishing and sharing this version among Arabic communities would encourage the spread of the tool. Moreover, the difficulties of dealing with text direction motivated and encouraged various tools and software to find optimal solutions. Finally, the tool can be used in teaching sessions which would help improve understanding of the ontology development commands for beginners.

8.5.2 Document-centric Ontology Development

Since the Excel spreadsheet in ODEW is unrestricted in design and structure, the ontology development framework here is utterly dependent on the Tawny-OWL and Clojure functionalities. This dependency means that the ontology developers must have strong skills in Tawny-OWL and Clojure to parse any Excel spreadsheet, as they are formatted by domain experts, and design the ontology patterns accordingly.

The implementation of ODEW started by using the tolAPC catalog, which a group of domain researchers have already developed it. The choice of the template was not available at that time because an existing catalog was used; further, another simple spreadsheet was utilized to develop the Amino Acid Ontology for the evaluation phase. Therefore, designing a template would be an excellent addition to be employed in the ODEW. A controlled pre-structured template of an Excel spreadsheet would be ideal for helping with the extraction process and design ontology patterns. However, significant considerations should be made.
when developing such templates to provide a successful template that fulfills most domain needs. Similarly, templating the wordified ontologies will be beneficial in terms of the basic structure for ontologies documentation.

On the other hand, designing the documentation in wordified ontology was quite complex. Given that the intended audience is familiar with Word, an attractive structure for the documentation is required, as reading a plain text block is tedious. Due to the size of the ontology, the wordified ontology lacks navigation and hyperlinks. This is beneficial for large ontologies, and it might be accomplished by utilizing the table of contents to allow users to browse between ontology sections.

Although, in HTML pages, a clicking button can be used to hide the source code and then display it when the user clicks the button, as demonstrated in Cloud Ontology in Chapter 6, to achieve this in wordified ontology, two versions of the wordified ontology must be generated (one with and the other without source code).

### 8.5.3 Reusing Existing Ontology

Reprocessing an existing ontology is a great challenge, and it is still limited in the proposed approaches. The reusing of upper ontologies has been utilized in various ontology engineering methodologies (as described in Chapter 2) and has tremendous impacts on the evolution of the ontology engineering field. However, starting from scratch in this work gives a quick and immediate feel for the proposed approaches. It also helps to develop and evaluate the work within the research life-cycle. This limitation can be overcome by investigating the integration with existing ontologies. In terms of the scalability of ontologies, the scope of ontology is crucial for rapid expansion and heterogeneity, particularly in biology and medicine. In this study, only small ontologies are successfully developed and tested. To sum it up, there is no certainty regarding the scalability of the proposed approaches.
8.5.4 Evaluation Stage

In Chapter 7, the procedures to evaluate the proposed approaches in developing the ontologies have been discussed. Since Excel spreadsheets and Word documents have been incorporated with producing new ontologies, the focus was on how different users interact to generate ontologies. Therefore, the evaluation experiments were designed with users, assisted their performance, and gained their opinion about the wordified ontologies.

The evaluation experiments were finite in the number of participants hired. The recruitment procedure is delicate because the aim was to have people who hold reasonable knowledge of ontologies. Additionally, finding specialized participants is limited by the time and position; therefore, remote experiments were conducted, which helped to increase the overall number of participants. Moreover, the experiment’s time was quite long because of testing two different formalisms of an ontology and asking for an opinion regarding their performance. The sample was not ideal since there were more female participants than males, as well as experts in the ontology field being less presented. In addition, the sample lacked any visually impaired users; therefore, testing the accessibility of the wordified ontology was not possible.

As mentioned above, it is hard to recruit people in general, and targeting particular users makes it even harder; therefore, this process consumed more time than expected. However, the experiments ended up with devoted participants, whose participation contributed heavily to the thesis.

8.6 Future Work

Ontology engineering is a well-investigated field in collaborating and interacting with users and designing and developing robust ontologies that serve their purposes. This thesis proposes frameworks for collaborating with domain experts during the ontology construction process; office tools can be integrated to facilitate this interaction between diverse specialties.

The integration of Microsoft Excel and Word applications can serve as a starting point for
developing a comprehensive user-friendly software suite with graphical user interfaces that can serve a broad range of ontology users and contribute to the development of ontologies through using convenient tools.

Furthermore, to improve what has been done so far, the following questions are set as seeds to motivate the next steps in future work.

For ODEW, the following questions are intended for future work considerations:

- Is it possible to achieve the same results if the size of the Excel spreadsheet is increased to accommodate the creation of a large ontology?

- What about creating a user interface to interact with users? Or designing a general excel template to help with patternization?

- What methods could be used to integrate with existing ontologies? Or reusing existing terms?

- Is it useful to integrate Tawny-OWL with other web platform servers to design a full API for ontology development using excel/word?

For wordified ontology:

- Is it possible to automate the process of generating the Word documents from the ontology source code?

- What about reversing the process? Can a wordified ontology be converted to Tawny-OWL file? Alternatively, how about incorporating a subset of changes made to the wordified version into the source file.

- Can wordified ontologies be used to collaborate with specialists on the development of more complex ontologies?

- What is the ideal format of the ontology documentation? Can we develop model guidelines for ontology documentation?
• Is it possible to apply a similar concept to other types of documentation/code tools, such as Juypter, and what advantages would this environment have compared to word?

For internationalization:

• Is it possible to translate the Tawny-OWL files from English to Arabic, or vice versa? Or even to any other language? Can we automate this process? How?

• What are other languages that would benefit from a Tawny-OWL environment?

8.7 Improving the Collaborative Ontology Engineering Landscape

The primary interest of this research was to engage users and encourage them to use their tools during the collaborative process of building ontologies. However, the ontology development cycle is heavily dependent on community collaboration at many different levels, making it difficult to separate the users from the development process. This compares to the segregation of users and ontology developers, which often happens in software development, simply because of the differences in tooling.

That is why alternative representations were proposed to visualize the ontological source code, such as HTML, wordified ontology, and the internationalized version of the Arabic language. These alternatives are directly related to the ontology script with the extra facilities of those visualizations; for instance, editing, adding sections, and hyperlinks. In addition, the narrative commentary style serves purposes of explanation and description of the ontology with an ability to show or hide ontological statements according to the users'/experts’ preferences.

However, ontology engineering is somehow related to software development, and the most important goal is to represent the knowledge accurately. Those fields have been investigated to help reach a better mechanism to build correct ontologies, and the synthesis of knowledge in this thesis could usefully feed back into these areas.
Appendix A

Evaluation:
Supplementary Material

Contents

A.1 Stage 1 Scripts ......................................................... 202
A.2 Stage 2 Questions .................................................. 205
A.3 Comments .............................................................. 207
  1.3.1 Stage 1 Comments ............................................ 207
  1.3.2 Stage 2 Comments ............................................ 207
A.1 Stage 1 Scripts
**Pizza Ontology Evaluation**

Thank you for participating in this evaluation session. Please note that, there will be a voice/activity recording system during the session. Any information you provide in this experiment will be handled confidentially. Your data will totally be anonymous, no names will be revealed or linked to any results. You are free to withdraw from the test at any time.

If you require any further information please contact me Aisha Blfgeh (a.blfgeh1@newcastle.ac.uk) or supervisor: Dr Phillip Lord (phillip.lord@newcastle.ac.uk).

---

**For researcher only.**

**User # :**

---

In this session, I aim to compare the ability to read an ontology in two different formats. Furthermore, comparing the simplicity of each format by finding the errors in the ontology and suggesting a solution.

You will see two formats of the Pizza ontology, your main task is to find as many errors as you can in each format. (3 – 5 errors)

- You have 15 minutes to spend in each format.
- Please record the errors you found as described below.
- Do not worry if you could not find all errors.
- If you unable to find any errors please ask for help.

Thank you for your participation. 😊

---

**Format A**

- Please open the word document (pizza-test-a#.docx).
- Make sure the Track Changes is activated (Review Ribbon).
- Start reading the document carefully.
- Try and Find any error within the clojure statements in the document.
- If you find an error, please select the words/line corresponding to the error and add a comment (use comment icon) explaining what the error is.
- Please save all you changes.
- If you finish before time, please record your time here [ ]. Please make sure you write all your comments and save your changes before you move to the other format. Because once you have done you cannot go back to the first format.
Format B

- Please open the owl file (pizza-test-b#.owl) using Protege shortcut.
- Start browsing the ontology carefully.
- Try and find any error (classification or relations).
- You are not allowed to run any reasoner.
- If you find an error, please write it down in the table below, and try to explain your view about this error.
- If you finish before time, please record your time here. Please make sure you write all your comments before you move to the other format because once you have done you cannot go back to the first one.

Errors found:

<table>
<thead>
<tr>
<th>Error</th>
<th>Name</th>
<th>Type / Place</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A.2 Stage 2 Questions

In order to ensure that participants comprehend the ontology correctly, we asked them to extract some information by answering some questions. The total of ten questions were quite similar in each format. These questions are included in Appendix.

**Protégé Experiment Questions**

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. What is the name of the ontology?</td>
<td></td>
</tr>
<tr>
<td>2. What is the first class defined in the ontology? (Apart from &quot;Thing&quot; class)</td>
<td></td>
</tr>
<tr>
<td>3. How many object properties do you see? Write them down please.</td>
<td></td>
</tr>
<tr>
<td>4. How many annotation properties are there? Write them down please.</td>
<td></td>
</tr>
<tr>
<td>5. What is the total number of classes?</td>
<td></td>
</tr>
<tr>
<td>6. There is one error in this ontology, can you see it? Where? Why do think this is an error?</td>
<td></td>
</tr>
<tr>
<td>7. Can you correct the error? How?</td>
<td></td>
</tr>
<tr>
<td>8. What is the side chain structure for “Methionine”?</td>
<td></td>
</tr>
<tr>
<td>9. What is the size of “Tryptophan”?</td>
<td></td>
</tr>
<tr>
<td>10. What is the super class of “PhysicoChemicalProperty” class?</td>
<td></td>
</tr>
</tbody>
</table>
### Wordified Ontology Questions

1. What is the name of the ontology?
2. What is the first class defined in the ontology?
3. How many object properties do you see? Write them down please.
4. How many annotation properties are there? Write them down please.
5. What is the total number of classes?
6. Can you add a new annotation property called “hasSingleLetterName”? Please write the corresponding statement.
7. Where do you think you should insert the previous statement?
8. What is the side chain structure for “Cysteine”? 
9. Can you list the subclasses of class “PhysicoChemicalProperty”?
10. There is one error in this ontology, can you see it? Where? Why do think this is an error?
A.3 Comments

Do you have any comments / suggestions for us:

- The questions take more time to read and answer. If there were multiple choice question will be clearer.
- Thank you for making us understand and teach us that.

1.3.1 Stage 1 Comments

Further opinion

1.3.2 Stage 2 Comments

Further opinion
Do you have any comments / suggestions for us:

- It would be better if you emphasize at the beginning that the position of the error.
- You can increase the time limit to 20 minutes or add some more hints in the presentation before the test.
- It is a good experiment, however, if the evaluator has some idea about the purpose of the experiment, you maybe get better result.
Please describe any difficulties you faced during the experiment:

In the software the hierarchy was very clear, but because this was first time use it I found not clear.

1- Explanation wasn't enough to what I should do exactly before the experiment.
2- Time wasn't enough.

I wasn't so sure about how to find the error since I don't have any experience with ontology but at the end I realize that it is so easy to spot some logic error.

The first format needs more than 15 minutes.
Would you please tell us the reason behind your preferences:

- **Protégé**: As stated above, it is easier on the eyes because it was clear so that I can recognize every structure.

- **Both**: I understood the OWL file after reading the word document so now they can both be helpful.

- **Wordified ontolgy**: To understand the code because of its Excel sheet, it's clearly.

- **Wordified ontolgy**: It is easy to understand.
Appendix B

Arabic Questionnaire

Contents

B.1 Arabic Questionnaire with English Translation and Answers  . . . . . . . 212

- 211 -
B.1 Arabic Questionnaire with English Translation and Answers
**Informed Consent Form of Participation**

**Building ontologies using Tawny-OWL Arabic version**

<table>
<thead>
<tr>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>My name is Aisha Blfgeh, IPH student at Newcastle University. I am distributing this questionnaire with the purpose of testing and evaluating an Arabic version of Tawny-OWL as a part of my study.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>The main objective is to test how easy to deal with the Arabic commands to build ontologies and measure the usability of that Arabic environment.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Participant</th>
</tr>
</thead>
<tbody>
<tr>
<td>You are invited to participate in this research project because you are an Arabic speaker with ontology knowledge.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Voluntary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Your participation in this research study is entirely voluntary. You are free to withdraw at any time. If you decide not to participate in this study or if you withdraw from participating at any time, you will not be penalized.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>The procedure involves filling this online survey that will take approximately 40 minutes. Your responses will be confidential, and we do not collect identifying information such as your name, or IP address. The survey will be in Arabic and the questions about using the Arabic commands to build ontologies. It consists of the following parts:</td>
</tr>
<tr>
<td>- <strong>Part1:</strong> Describing the Arabic Tawny-OWL commands to build a simple Pizza Ontology. Then ask about these commands to check the overall comprehension.</td>
</tr>
<tr>
<td>- <strong>Part2:</strong> Build a simple Family Ontology with guidance using Arabic commands.</td>
</tr>
<tr>
<td>- <strong>Part3:</strong> Feedback survey using System Usability Scale to test the usability of the Arabic version of commands.</td>
</tr>
<tr>
<td>- <strong>Part4:</strong> Collect demographic data (gender, age, expertise, affiliation...).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Confidentiality</th>
</tr>
</thead>
<tbody>
<tr>
<td>We will do our best to keep your information confidential. All data is stored in a password protected electronic format accessible by researcher only. To help protect your confidentiality, the surveys will not contain information that will personally identify you. The results of this study will be used for scholarly purposes only.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>If you have any questions about this study, please contact the researcher, Aisha Blfgeh (<a href="mailto:a.blfgeh1@newcastle.ac.uk">a.blfgeh1@newcastle.ac.uk</a>). If you have a complaint, please contact the supervisor Phillip Lord (<a href="mailto:phillip.lord@newcastle.ac.uk">phillip.lord@newcastle.ac.uk</a>) with the details. We have eight weeks to consider your complaint. If we have not resolved it within this time you may complain to School of Computing in Newcastle University. By clicking the &quot;Agree&quot; button below, indicates that:</td>
</tr>
<tr>
<td>- You have read the information above about the research and your participation.</td>
</tr>
<tr>
<td>- You voluntarily agree to participate.</td>
</tr>
<tr>
<td>- You are 18 years or above Arabic speaker.</td>
</tr>
<tr>
<td>If you do not wish to participate in the research, you may click the &quot;Disagree&quot; button.</td>
</tr>
<tr>
<td><strong>Agree</strong></td>
</tr>
<tr>
<td><strong>Disagree</strong></td>
</tr>
</tbody>
</table>

**Form of Participation**

<table>
<thead>
<tr>
<th>Tawny-OWL</th>
<th>نموذج الموافقة المتعلقة للمشاركة في بناء النسخة العربية من Tawny-OWL</th>
</tr>
</thead>
<tbody>
<tr>
<td>المقدمة</td>
<td>اسمك بلغةطقسية ملاطلة تكون مكتوبة في جامعة نيوكاسل. اقترح بتوتيت هذا ± الفرط إن كنت ترغب في المشاركة في هذا الجزء من الدراسة.</td>
</tr>
<tr>
<td>الغرض</td>
<td>الغرض الرئيسي هو اختبار مدى سهولة التعامل مع الأوامر العربية لبناء أنظمة تسمى تاون ي.</td>
</tr>
<tr>
<td>المشاركين</td>
<td>أي مدعٍ للمشاركة في هذا المشروع الباحثي يجب أن يتحدث اللغة العربية.</td>
</tr>
<tr>
<td>التخطيط</td>
<td>ستتم المشاركة في هذه الدراسة البحثية بطريقة منطقية بالكامل. لكل مشارك حرية في الانسحاب في أي وقت. إذا قررت عدم المشاركة في هذه الدراسة أو إذا أبلغت من المشاركة في أي وقت، فإنك مطالب بتقديم شكوى.</td>
</tr>
<tr>
<td>الإجراءات</td>
<td>يتضمن الإجراءات، هذا الاستطلاع عبر الإنترنت الذي سيستغرق حوالي 40 دقيقة، ستكون روتوك مسرة، ولكن لا تتبع معلومات تجارية مثل IP أو اسمك أو عنوان IP. ستكون الاستطلاع باللغة العربية والمنزلة حول استخدام الأوامر العربية لبناء أنظمة تسمى تاون ي.</td>
</tr>
<tr>
<td>الجزء الأول:</td>
<td>الجزء الأول: وضع الأوامر باستخدام نظام تاون ي.</td>
</tr>
<tr>
<td>الجزء الثاني:</td>
<td>الجزء الثاني: بناء أنظمة تسمى تاون ي باستخدام نظام Tawny-OWL.</td>
</tr>
<tr>
<td>الجزء الثالث:</td>
<td>الجزء الثالث: استبيان الاستخدام للمساعدة في تحليل البيانات.</td>
</tr>
<tr>
<td>الجزء الرابع:</td>
<td>الجزء الرابع: استبيان الاستخدام للمساعدة في تحليل البيانات.</td>
</tr>
<tr>
<td>الهيكل</td>
<td>الهيكل: استبيان الاستخدام للمساعدة في تحليل البيانات.</td>
</tr>
<tr>
<td>الهدف</td>
<td>الهدف الرئيسي هو اختبار مدى سهولة التعامل مع الأوامر العربية لبناء أنظمة تسمى تاون ي.</td>
</tr>
<tr>
<td>البحث</td>
<td>البحث: بناء أنظمة تسمى تاون ي باستخدام نظام Tawny-OWL.</td>
</tr>
<tr>
<td>الخريطة</td>
<td>الخريطة: شهادة الاستخدام للمساعدة في تحليل البيانات.</td>
</tr>
<tr>
<td>شهادة</td>
<td>شهادة: استبيان الاستخدام للمساعدة في تحليل البيانات.</td>
</tr>
</tbody>
</table>

**Privacy Statement**

This study aims to test how easy to deal with the Arabic commands to build ontologies and measure the usability of that Arabic environment. The procedure involves filling this online survey that will take approximately 40 minutes. Your responses will be confidential, and we do not collect identifying information such as your name, or IP address. The survey will be in Arabic and the questions about using the Arabic commands to build ontologies. It consists of the following parts:

- **Part1:** Describing the Arabic Tawny-OWL commands to build a simple Pizza Ontology. Then ask about these commands to check the overall comprehension.
- **Part2:** Build a simple Family Ontology with guidance using Arabic commands.
- **Part3:** Feedback survey using System Usability Scale to test the usability of the Arabic version of commands.
- **Part4:** Collect demographic data (gender, age, expertise, affiliation...).

Confidentiality: We will do our best to keep your information confidential. All data is stored in a password protected electronic format accessible by researcher only. To help protect your confidentiality, the surveys will not contain information that will personally identify you. The results of this study will be used for scholarly purposes only.

Contact: If you have any questions about this study, please contact the researcher, Aisha Blfgeh (a.blfgeh1@newcastle.ac.uk). If you have a complaint, please contact the supervisor Phillip Lord (phillip.lord@newcastle.ac.uk) with the details. We have eight weeks to consider your complaint. If we have not resolved it within this time you may complain to School of Computing in Newcastle University. By clicking the "Agree" button below, indicates that:

- You have read the information above about the research and your participation.
- You voluntarily agree to participate.
- You are 18 years or above Arabic speaker.

If you do not wish to participate in the research, you may click the "Disagree" button.

**Agree**

**Disagree**

**Privacy Statement**

This study aims to test how easy to deal with the Arabic commands to build ontologies and measure the usability of that Arabic environment. The procedure involves filling this online survey that will take approximately 40 minutes. Your responses will be confidential, and we do not collect identifying information such as your name, or IP address. The survey will be in Arabic and the questions about using the Arabic commands to build ontologies. It consists of the following parts:

- **Part1:** Describing the Arabic Tawny-OWL commands to build a simple Pizza Ontology. Then ask about these commands to check the overall comprehension.
- **Part2:** Build a simple Family Ontology with guidance using Arabic commands.
- **Part3:** Feedback survey using System Usability Scale to test the usability of the Arabic version of commands.
- **Part4:** Collect demographic data (gender, age, expertise, affiliation...).

Confidentiality: We will do our best to keep your information confidential. All data is stored in a password protected electronic format accessible by researcher only. To help protect your confidentiality, the surveys will not contain information that will personally identify you. The results of this study will be used for scholarly purposes only.

Contact: If you have any questions about this study, please contact the researcher, Aisha Blfgeh (a.blfgeh1@newcastle.ac.uk). If you have a complaint, please contact the supervisor Phillip Lord (phillip.lord@newcastle.ac.uk) with the details. We have eight weeks to consider your complaint. If we have not resolved it within this time you may complain to School of Computing in Newcastle University. By clicking the "Agree" button below, indicates that:

- You have read the information above about the research and your participation.
- You voluntarily agree to participate.
- You are 18 years or above Arabic speaker.

If you do not wish to participate in the research, you may click the "Disagree" button.

**Agree**

**Disagree**

**Privacy Statement**

This study aims to test how easy to deal with the Arabic commands to build ontologies and measure the usability of that Arabic environment. The procedure involves filling this online survey that will take approximately 40 minutes. Your responses will be confidential, and we do not collect identifying information such as your name, or IP address. The survey will be in Arabic and the questions about using the Arabic commands to build ontologies. It consists of the following parts:

- **Part1:** Describing the Arabic Tawny-OWL commands to build a simple Pizza Ontology. Then ask about these commands to check the overall comprehension.
- **Part2:** Build a simple Family Ontology with guidance using Arabic commands.
- **Part3:** Feedback survey using System Usability Scale to test the usability of the Arabic version of commands.
- **Part4:** Collect demographic data (gender, age, expertise, affiliation...).

Confidentiality: We will do our best to keep your information confidential. All data is stored in a password protected electronic format accessible by researcher only. To help protect your confidentiality, the surveys will not contain information that will personally identify you. The results of this study will be used for scholarly purposes only.

Contact: If you have any questions about this study, please contact the researcher, Aisha Blfgeh (a.blfgeh1@newcastle.ac.uk). If you have a complaint, please contact the supervisor Phillip Lord (phillip.lord@newcastle.ac.uk) with the details. We have eight weeks to consider your complaint. If we have not resolved it within this time you may complain to School of Computing in Newcastle University. By clicking the "Agree" button below, indicates that:

- You have read the information above about the research and your participation.
- You voluntarily agree to participate.
- You are 18 years or above Arabic speaker.

If you do not wish to participate in the research, you may click the "Disagree" button.
Part 1:

First of all, I would like to give a summary of Tawny-OWL. It is an integrated scripting interface through which we design and develop an OWL ontology. The commands are executed using the Clojure language and run on the Java Virtual Machine. Most of the writing rules in Tawny-OWL are based on Manchester Syntax rules. More information is available on the web page at the following link:

https://github.com/phillord/tawny-owl

I will now explain Tawny-OWL's commands for building the ontology in a simplified and straightforward way. Before that, I would like to draw attention to the following points:

- Tawny-OWL is built in the programming language Clojure, which is a dialect of the Lisp language, so:
  - Commands are always written inside the parentheses, so any command and its parameters are written inside a general parenthesis.
  - The command operator is always written before any values is to be executed. For example: (+ 1 2) upon execution gives the result (3).
  - Line indentation is important and helps to understand the command and the sequence of any parameters.

First, to define the ontology and give it its own name, we use the command (defontology) with passing the name as follows:

(defontology "name of ontology")

It is also possible to pass a group of arguments with the definition to add (Internationalized Resource Identifier - IRI) or a comment ... etc. These additions are optional and do not affect the execution of orders. Rather, it is preferable to support and record the ontology with sufficient information and data to be clear and correct. See it now with these additions:

(defontology Arabic-pizza
  :iri "http://www.ncl.ac.uk/pizza"
  :prefix "piz"
  :comment "An example Arabic Pizza ontology modelled from Manchester University, written using the tawny-owl library"
  :versioninfo "First version"
  :sealso "Manchester Version")

Now we can define the classes of this ontology using the command (defclass) and no need to pass the name of the ontology because we are in the same file as follows:

(defclass Pizza
  :label "Pizza"
  :defclass PizzaComponent)

Note also that we have added a (:label1) with the first class, and this is optional, of course. As you can see, the second class does not contain a (:label)

The super class can be added when defining the sub class by adding the (:subclass) parameter as follows:

(defclass superclass
  :subclass "subclass")

The class can be added when defining the subclass as follows:

(defclass subclass
  :subclass "subclass")

The super class can be added when defining the subclass as follows:

(defclass superclass
  :subclass "subclass")

Now we can define the classes of this ontology using the command (defontology) with passing the name as follows:

(defontology "name of ontology")

It is also possible to pass a group of arguments with the definition to add (Internationalized Resource Identifier - IRI) or a comment ... etc. These additions are optional and do not affect the execution of orders. Rather, it is preferable to support and record the ontology with sufficient information and data to be clear and correct. See it now with these additions:

(defontology Arabic-pizza
  :iri "http://www.ncl.ac.uk/pizza"
  :prefix "piz"
  :comment "An example Arabic Pizza ontology modelled from Manchester University, written using the tawny-owl library"
  :versioninfo "First version"
  :sealso "Manchester Version")

Now we can define the classes of this ontology using the command (defclass) and no need to pass the name of the ontology because we are in the same file as follows:

(defclass Pizza
  :label "Pizza"
  :defclass PizzaComponent)

Note also that we have added a (:label1) with the first class, and this is optional, of course. As you can see, the second class does not contain a (:label)

The super class can be added when defining the sub class by adding the (:subclass) parameter as follows:

(defclass superclass
  :subclass "subclass")

The class can be added when defining the subclass as follows:

(defclass subclass
  :subclass "subclass")

Now we can define the classes of this ontology using the command (defontology) with passing the name as follows:

(defontology "name of ontology")

It is also possible to pass a group of arguments with the definition to add (Internationalized Resource Identifier - IRI) or a comment ... etc. These additions are optional and do not affect the execution of orders. Rather, it is preferable to support and record the ontology with sufficient information and data to be clear and correct. See it now with these additions:

(defontology Arabic-pizza
  :iri "http://www.ncl.ac.uk/pizza"
  :prefix "piz"
  :comment "An example Arabic Pizza ontology modelled from Manchester University, written using the tawny-owl library"
  :versioninfo "First version"
  :sealso "Manchester Version")

Now we can define the classes of this ontology using the command (defclass) and no need to pass the name of the ontology because we are in the same file as follows:

(defclass Pizza
  :label "Pizza"
  :defclass PizzaComponent)

Note also that we have added a (:label1) with the first class, and this is optional, of course. As you can see, the second class does not contain a (:label)

The super class can be added when defining the sub class by adding the (:subclass) parameter as follows:

(defclass superclass
  :subclass "subclass")

The class can be added when defining the subclass as follows:

(defclass subclass
  :subclass "subclass")

Now we can define the classes of this ontology using the command (defontology) with passing the name as follows:

(defontology "name of ontology")

It is also possible to pass a group of arguments with the definition to add (Internationalized Resource Identifier - IRI) or a comment ... etc. These additions are optional and do not affect the execution of orders. Rather, it is preferable to support and record the ontology with sufficient information and data to be clear and correct. See it now with these additions:

(defontology Arabic-pizza
  :iri "http://www.ncl.ac.uk/pizza"
  :prefix "piz"
  :comment "An example Arabic Pizza ontology modelled from Manchester University, written using the tawny-owl library"
  :versioninfo "First version"
  :sealso "Manchester Version")

Now we can define the classes of this ontology using the command (defclass) and no need to pass the name of the ontology because we are in the same file as follows:

(defclass Pizza
  :label "Pizza"
  :defclass PizzaComponent)

Note also that we have added a (:label1) with the first class, and this is optional, of course. As you can see, the second class does not contain a (:label)

The super class can be added when defining the sub class by adding the (:subclass) parameter as follows:

(defclass superclass
  :subclass "subclass")

The class can be added when defining the subclass as follows:

(defclass subclass
  :subclass "subclass")

Now we can define the classes of this ontology using the command (defontology) with passing the name as follows:

(defontology "name of ontology")

It is also possible to pass a group of arguments with the definition to add (Internationalized Resource Identifier - IRI) or a comment ... etc. These additions are optional and do not affect the execution of orders. Rather, it is preferable to support and record the ontology with sufficient information and data to be clear and correct. See it now with these additions:

(defontology Arabic-pizza
  :iri "http://www.ncl.ac.uk/pizza"
  :prefix "piz"
  :comment "An example Arabic Pizza ontology modelled from Manchester University, written using the tawny-owl library"
  :versioninfo "First version"
  :sealso "Manchester Version")

Now we can define the classes of this ontology using the command (defclass) and no need to pass the name of the ontology because we are in the same file as follows:

(defclass Pizza
  :label "Pizza"
  :defclass PizzaComponent)

Note also that we have added a (:label1) with the first class, and this is optional, of course. As you can see, the second class does not contain a (:label)

The super class can be added when defining the sub class by adding the (:subclass) parameter as follows:
We will now define the property (hasTopping) using the command (defoproperty) and pass super property, range and domain for this property with the definition as follows:

(defunproperty hasTopping
  :subproperty hasIngredient
  :range PizzaTopping
  :domain Pizza)
3. Likewise, please select the correct answer to define the 
(hasBase) property using the command (defoproperty) and 
pass super property, the range and domain for this property as 
previous example:

```
(defoproperty hasBase 
  :subproperty hasIngredient 
  :range PizzaBase 
  :domain Pizza)
```

Now we will define an individual of a pizza which is 
(CheesePizza) which contains (TomatoSauce) and also 
(CheeseTopping) only:

```
(define individual CheesePizza
  :type Pizza
  :fact
  (some hasTopping CheeseTopping)
  (some hasTopping TomatoSauce)
  (only (hasTopping CheeseTopping owl-or hasSauce TomatoSauce)))
```

### Part 2:

Based on what you have already experienced in Part 1. In this 
Part you are going to write the Arabic commands yourself to 
build a simple ontology. I will guide you through this part. You 
can go back and check the information as you desire.

1. Define the ontology (familytree), give it the following info:
   - iri= http://example.com/owl/familytree.
   - comment= An example of family tree ontology.

```
(defontology familytree 
  :iri "http://example.com/owl/familytree" 
  :comment "An example of family tree ontology")
```

2. Define class Person, with the following info:
   - Label= Human.
   - Comment= This is the main class where all people are of 
this kind which also means Human.

```
(defineclass Person
  :label "Human" 
  :comment "This is the main class where all people 
are of this kind which also means Human.")
```

3. Define two disjoint classess Male and Female which are both 
subclasses of Person. Add each class the appropriate labels and 
comments:
   - Label = Male.
   - Comment = This is for every male person.
   - Label = Female.
   - Comment = This is for every female person.

```
(defclass Man 
  :label "Male"

(defclass Woman 
  :label "Female"
```

---

3. Likewise, please select the correct answer to define the 
(hasBase) property using the command (defoproperty) and 
pass super property, the range and domain for this property as 
previous example:

```
(defoproperty hasBase 
  :subproperty hasIngredient 
  :range PizzaBase 
  :domain Pizza)
```

Now we will define an individual of a pizza which is 
(CheesePizza) which contains (TomatoSauce) and also 
(CheeseTopping) only:

```
(define individual CheesePizza
  :type Pizza
  :fact
  (some hasTopping CheeseTopping)
  (some hasTopping TomatoSauce)
  (only (hasTopping CheeseTopping owl-or hasSauce TomatoSauce)))
```

### Part 2:

Based on what you have already experienced in Part 1. In this 
Part you are going to write the Arabic commands yourself to 
build a simple ontology. I will guide you through this part. You 
can go back and check the information as you desire.

1. Define the ontology (familytree), give it the following info:
   - iri= http://example.com/owl/familytree.
   - comment= An example of family tree ontology.

```
(defontology familytree 
  :iri "http://example.com/owl/familytree" 
  :comment "An example of family tree ontology")
```

2. Define class Person, with the following info:
   - Label= Human.
   - Comment= This is the main class where all people are of 
this kind which also means Human.

```
(defineclass Person
  :label "Human" 
  :comment "This is the main class where all people 
are of this kind which also means Human.")
```

3. Define two disjoint classess Male and Female which are both 
subclasses of Person. Add each class the appropriate labels and 
comments:
   - Label = Male.
   - Comment = This is for every male person.
   - Label = Female.
   - Comment = This is for every female person.

```
(defclass Man 
  :label "Male"

(defclass Woman 
  :label "Female"
```

---

3- Likewise, please select the correct answer to define the 
(hasBase) property using the command (defoproperty) and 
pass super property, the range and domain for this property as 
previous example:

```
(defoproperty hasBase 
  :subproperty hasIngredient 
  :range PizzaBase 
  :domain Pizza)
```

Now we will define an individual of a pizza which is 
(CheesePizza) which contains (TomatoSauce) and also 
(CheeseTopping) only:

```
(define individual CheesePizza
  :type Pizza
  :fact
  (some hasTopping CheeseTopping)
  (some hasTopping TomatoSauce)
  (only (hasTopping CheeseTopping owl-or hasSauce TomatoSauce)))
```

### Part 2:

Based on what you have already experienced in Part 1. In this 
Part you are going to write the Arabic commands yourself to 
build a simple ontology. I will guide you through this part. You 
can go back and check the information as you desire.

1. Define the ontology (familytree), give it the following info:
   - iri= http://example.com/owl/familytree.
   - comment= An example of family tree ontology.

```
(defontology familytree 
  :iri "http://example.com/owl/familytree" 
  :comment "An example of family tree ontology")
```

2. Define class Person, with the following info:
   - Label= Human.
   - Comment= This is the main class where all people are of 
this kind which also means Human.

```
(defineclass Person
  :label "Human" 
  :comment "This is the main class where all people 
are of this kind which also means Human.")
```

3. Define two disjoint classess Male and Female which are both 
subclasses of Person. Add each class the appropriate labels and 
comments:
   - Label = Male.
   - Comment = This is for every male person.
   - Label = Female.
   - Comment = This is for every female person.

```
(defclass Man 
  :label "Male"

(defclass Woman 
  :label "Female"
```

---

3- Likewise, please select the correct answer to define the 
(hasBase) property using the command (defoproperty) and 
pass super property, the range and domain for this property as 
previous example:

```
(defoproperty hasBase 
  :subproperty hasIngredient 
  :range PizzaBase 
  :domain Pizza)
```

Now we will define an individual of a pizza which is 
(CheesePizza) which contains (TomatoSauce) and also 
(CheeseTopping) only:

```
(define individual CheesePizza
  :type Pizza
  :fact
  (some hasTopping CheeseTopping)
  (some hasTopping TomatoSauce)
  (only (hasTopping CheeseTopping owl-or hasSauce TomatoSauce)))
```

### Part 2:

Based on what you have already experienced in Part 1. In this 
Part you are going to write the Arabic commands yourself to 
build a simple ontology. I will guide you through this part. You 
can go back and check the information as you desire.

1. Define the ontology (familytree), give it the following info:
   - iri= http://example.com/owl/familytree.
   - comment= An example of family tree ontology.

```
(defontology familytree 
  :iri "http://example.com/owl/familytree" 
  :comment "An example of family tree ontology")
```

2. Define class Person, with the following info:
   - Label= Human.
   - Comment= This is the main class where all people are of 
this kind which also means Human.

```
(defineclass Person
  :label "Human" 
  :comment "This is the main class where all people 
are of this kind which also means Human.")
```

3. Define two disjoint classess Male and Female which are both 
subclasses of Person. Add each class the appropriate labels and 
comments:
   - Label = Male.
   - Comment = This is for every male person.
   - Label = Female.
   - Comment = This is for every female person.

```
(defclass Man 
  :label "Male"

(defclass Woman 
  :label "Female"
```
Similarly: Mother, GradMother, Daughter, Sister and Wife subclasses of Female.

5- Define inverse properties hasParent, hasChild

(as-inverse
(defoproperty hasParent)
(defoproperty hasChild))

6- Define the properties (hasSon), then (hasDaughter) both are subproperties of the property (hasChild) and add the proper domain and range for each one.

(as-disjoint
(defoproperty hasSon
:subproperty hasChild
:domain Person
:range Man)
(defoproperty hasDaughter
:subproperty hasChild
:domain Person
:range Woman))

Or

(as-disjoint-subclasses hasChild
(defoproperty hasSon
:domain Person
:range Man)
(defoproperty hasDaughter
:domain Person
:range Woman))

Similarly; hasBrother, hasSister subproperties hasSibling

7- Consider that the following individuals are defined: Ali, Maryam, Sarah.
Define the individual (Ahmad) with the following info:
Type= Man
hasWife= Maryam
Part 3:
You will now give your opinion about your experience in using the Arabic language to write commands to build the ontology and what is your impression of using this Arabic system.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. I think that I would like to use this system frequently.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2. I found the system unnecessarily complex.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3. I thought the system was easy to use.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4. I think that I would need the support of a technical person to be able to use this system.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5. I found the various functions in this system were well integrated.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6. I thought there was too much inconsistency in this system.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7. I would imagine that most people would learn to use this system very quickly.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8. I found the system very cumbersome to use.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9. I felt very confident using the system.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10. I needed to learn a lot of things before I could get going with this system.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Part 4:
Finally, this is some demographic information that will help in the process of analyzing the results of this questionnaire.

1. Are you: o Male o Female o other:
2. What is your occupation?
3. What is your field of research / work?
4. How do you rate your experience with ontologies (1 = “unfamiliar” and 5 = “very familiar”).
5. How do you rate your experience with ontologies construction (1 = “unfamiliar” and 5 = “very familiar”).
6. If you would like to see the correct answer, please provide email address below:
7. Additional comments:

الجزء الثالث:
ستعطي رأيك الآن حول تجربتك في استخدام اللغة العربية لكتابة أنظمة معرفات. لبناء الأنتولوجيا وما هو انطباعك عن استخدام هذا النظام العربي.

الجزء الرابع:
أخيراً، هذه بعض المعلومات الديموغرافية التي تساعد في عملية تحليل نتائج هذا الاستبيان.
1. هل أنت: o ذكر o أنثى o غير ذلك.
2. ما وظيفتك؟
3. ما هو مجال الدراسة/ البحث؟
4. كيف تقيس خبرتك في الأنتولوجيا (1 = يمتلك فكرة بسيطة و عامة، 5 = يمتلك خبرة دائمة و عميق). 
5. كيف تقيس خبرتك في بناء وتطوير الأنتولوجيا (1 = يمتلك فكرة بسيطة، 5 = يمتلك خبرة دائمة و عميق). 
6. إن أردت معرفة الإجابات على أسئلتي أعلاه، الرجاء كتابة البريد الإلكتروني لأتمكن من إرسالها لك.
7. تعليقات إضافية.
This is an Arabic version of family ontology.
C.2 Testing ontologies

3.2.1 Wordified Pizza Ontology
Building Pizza Ontology Using Tawny-OWL

Aisha Blfgeh

Copyright (C) 2018, Aisha Blfgeh, Newcastle University

This programme is designed in order to test and evaluate a methodology of constructing ontologies using Document-centric approach.

Introduction

We will create a Pizza ontology; we choose pizzas because they are simple, well-understood and compositional (see here for more).

Define the Pizza Ontology

Starting off our definition of the Pizza ontology by defining a name space; and include all required name spaces to be used within the programme.

```
(ns pizza.pizza-test
 (:use [tawny.owl])
 (:require [tawny.read]
 [tawny.reasoner :as r]
 [tawny.pattern :as p]))
```

Now we create a new ontology with the values specified. The ontology will be available from the var "pizzaont" for use within the 'with-ontology' macro. Alternatively, it will be used for all operations inside the current namespace.

```
(defontology pizzaont
 :iri "http://www.ncl.ac.uk/pizzaont"
 :prefix "piz:"
 :comment "An example ontology modelled on the Pizza tutorial ontology from Manchester University, written using the tawny-OWL library"
 :versioninfo "Evaluation Test Version"
 :seealso "Manchester Version"
)
```

Named Classes

Let's add some classes to the ontology in order to define what we believe a pizza to be.

The main two core classes of the Pizza Ontology are; Pizza and PizzaComponent and we define them as follows:
These classes are *disjoint* or in other word *siblings*, so that an individual (or object) cannot be an instance of more than one of these two classes.

**Disjoint Classes**

The following classes are all siblings and should all be disjoint using the `as-disjoint` macro makes things a little easier. Currently we have to use the annotation function with label to pass a language in.

```lisp
(as-disjoint-subclasses
 PizzaComponent)
```

```lisp
(defclass PizzaBase
  ;; the pizza ontology contains some Portuguese labels. The :label keyword
  ;; used above is a shortcut for English
  :annotation (label "BaseDaPizza" "pt")))
```

**OWL Properties**

OWL properties are the relationships between any two individuals. There are different types of properties and we will see them as we go further with our Pizza ontology buildin.

Now that we have our first classes we can specify our properties.

**Inverse property**

Each object property may have a corresponding inverse property; for example the property `(isBaseOf)` is inverse of `(hasBase)`.

**Object properties**

Object properties link an individual to an individual in our ontology. For example: – a pizza has a base (deep pan or thin and crispy) – a pizza has a topping (i.e. cheese and tomato)

Now we define our first object property with its inverse:

```lisp
(as-inverse
 (defoproperty hasIngredient
   :characteristic :transitive)
 (defoproperty isIngredientOf
Property Hierarchy

In OWL, properties may have sub properties, so that it is possible to form hierarchies of properties. Sub properties specialise their super properties (in the same way that subclasses specialise their superclasses).

In our pizza ontology we have the following Property Hierarchy hasBase -> hasIngrediant <- hasTopping

(defoproperty hasTopping
   :subproperty hasIngredient
   :range PizzaTopping
   :domain Pizza
)

(defoproperty hasBase
   :subproperty hasIngredient
   :characteristic :functional
   :range PizzaBase
   :domain Pizza
)

We added a characteristic to our properties such as functional; which means for a given individual, there can be at most one individual that is related to the individual via this property.

Datatype Property

Datatype properties link an individual to an XML Schema Datatype value or an rdf literal. In other words, they describe relationships between an individual and data values.

#+begin_src clojure (defdproperty hasCalorificContentValue :range :XSD INTEGER)
(defn cal [number] (has-value hasCalorificContentValue number))
#+end_src

Annotation properties

Classes, properties, individuals and the ontology itself can be annotated with various pieces of information/meta-data. These pieces of information may take the form of auditing or editorial information. For example, comments, creation date, author, or references to resources such as web pages etc.

We define annotation property here as follows:

(defaproperty myOpinion
   :subproperty owl-comment-property
)
Describing a Pizza

And, now finally, we have the basic concepts that we need to build our pizza. First, we start off with a generic description of a pizza; we have already defined the class above, so we want to extend the definition rather than create a new one. We can achieve this using the class function as follows:

```
(owl-class Pizza
  :super
  (owl-some hasCalorificContentValue :XSD_INTEGER)
  (owl-some hasTopping PizzaTopping)
  (owl-some hasBase PizzaBase))
```

Here we define a set of subclasses which are all mutually disjoint. This section used to reflect the natural hierarchy within the lisp, by embedding multiple `as-disjoint-subclasses` forms.

```
(as-disjoint-subclasses
  PizzaBase
  (defclass ThinAndCrispyBase
    :super (cal 150)
    :annotation (label "BaseFinaEQuebradica" "pt"))
  (defclass DeepPanBase
    :super (cal 250)
    :annotation (label "BaseEspessa" "pt"))

  (as-disjoint-subclasses
    PizzaTopping
    (defclass CheeseTopping)
    (defclass FishTopping)
    (defclass FruitTopping)
    (defclass HerbSpiceTopping)
    (defclass MeatTopping)
    (defclass NutTopping)
    (defclass SauceTopping)
    (defclass VegetableTopping))

  (as-disjoint-subclasses
    CheeseTopping
    (declare-classes
      GoatsCheeseTopping
      GorgonzolaTopping
      ))
```
MozzarellaTopping
ParmesanTopping))
(as-disjoint-subclasses
FishTopping
  (declare-classes AnchoviesTopping
              MixedSeafoodTopping
              PrawnsTopping))
(as-disjoint-subclasses
FruitTopping
  (declare-classes PineappleTopping
              SultanaTopping))
(as-disjoint-subclasses
HerbSpiceTopping
  (declare-classes CajunSpiceTopping
              RosemaryTopping))
(as-disjoint-subclasses
MeatTopping
  (declare-classes ChickenTopping
              HamTopping
              HotSpicedBeefTopping
              PeperoniSausageTopping))
(as-subclasses
NutTopping
  (defclass PineKernels))
(as-subclasses
SauceTopping
  (defclass TobascoPepperSauce))
(as-disjoint-subclasses
VegetableTopping
  (declare-classes PepperTopping
              GarlicTopping
              PetitPoisTopping
              AsparagusTopping
              CaperTopping
              SpinachTopping
              ArtichokeTopping
              OnionTopping
              OliveTopping
              MushroomTopping
              RocketTopping)
(as-disjoint-subclasses
  PepperTopping
  (declare-classes PeperonataTopping
    JalapenoPepperTopping
    SweetPepperTopping
    GreenPepperTopping))

Value Partitions

We use this (design pattern) to refine our descriptions of our class 'Spiciness'. Value Partitions restrict the range of possible values to an exhaustive list, for example, our "Spiciness" will restrict the range to 'Mild', 'Medium', and 'Hot'.

(p/defpartition
  Spiciness
  [Mild
    Medium
    Hot]
)

Equivalent classes

These are the main categories which will be reasoned under.

(defclass CheesyPizza
  :equivalent
  (owl-and Pizza
    (owl-some hasTopping CheeseTopping)))

(defclass InterestingPizza
  :equivalent
  (owl-and Pizza
    (at-least 3 hasTopping PizzaTopping)))

(defclass FourCheesePizza
  :equivalent
  (owl-and Pizza
    (exactly 4 hasTopping CheeseTopping)))

(defclass VegetarianPizza
  :annotation
  (annotation myOpinion "Always a good start.")
  :equivalent
  (owl-and Pizza
    (owl-not
      (owl-some hasTopping MeatTopping))
    (owl-not
      (owl-some hasTopping TomatoTopping))
    (owl-not
      (owl-some hasTopping LeekTopping)))

TomatoTopping
  LeekTopping))
(owl-some hasTopping FishTopping)))

(defclass NonVegetarianPizza
  :annotation
  (annotation myOpinion "Not a good start."
  :equivalent
  (owl-and Pizza (owl-not VegetarianPizza)))

different, but equivalent, definition

(defclass VegetarianPizza2
  :equivalent
  (owl-and Pizza
    (only hasTopping
      (owl-not (owl-or MeatTopping FishTopping))))))

(defclass HighCaloriePizza
  :equivalent
  (owl-some hasCalorificContentValue
    (span >= 700))

(defclass MediumCaloriePizza
  :equivalent
  (owl-some hasCalorificContentValue
    (span >= 400 700))

(defclass LowCaloriePizza
  :equivalent
  (owl-some hasCalorificContentValue
    (span <= 400)))

;; named pizzas
(defclass NamedPizza
  :super Pizza)

Finally, we can generate arbitrarily complex statements. This is a one-off function that is unlikely to be much use for more general purposes.

(defn generate-named-pizza [& pizzalist]
  (doseq
    [[named & toppings] pizzalist]
    (owl-class
      named
      :super (some-only hasTopping toppings))))

We define all the named pizzas. We could get away without doing this, but then we would need to replace generate-named-pizza with a macro, and life is too short. We could also get around this by using a string for the pizza name.
(as-disjoint-subclasses
 NamedPizza
 (declare-classes MargheritaPizza CajunPizza CapricciosaPizza
                  SohoPizza ParmensePizza))

(generate-named-pizza
 [MargheritaPizza MozzarellaTopping TomatoTopping]
 [CajunPizza MozzarellaTopping OnionTopping PeperonataTopping
  PrawnsTopping TobascoPepperSauce TomatoTopping]
 [CapricciosaPizza AnchoviesTopping MozzarellaTopping
  TomatoTopping PeperonataTopping HamTopping CaperTopping
  OliveTopping]
 [ParmensePizza AsparagusTopping
  HamTopping
  MozzarellaTopping
  ParmesanTopping
  TomatoTopping]
 [SohoPizza OliveTopping RocketTopping TomatoTopping ParmesanTopping
  GarlicTopping]
)

Defining two examples of Pizza:

(defindividual ExampleMargheritaPizza
  :type MargheritaPizza
  :fact (is hasCalorificContentValue 300))

(defindividual ExampleParmense
  :type ParmensePizza
  :fact (is hasCalorificContentValue 700))

Adding spiciness to the specific Pizza.

(spiciness
  TobascoPepperSauce Hot
  RocketTopping Mild
)

Save the ontology

Ontologies save into the default directory, which is the top leve of the project. To save the ontology in OWL XML syntax which will be read into protégé We have the following:

(save-ontology "pizza-test.owl" :owl)
3.2.2 Wordified Amino Acid Ontology
This is Amino Acid ontology created using an Excel spreadsheet as a source of information for the ontology values. Author Aisha Blfgeh Newcastle University Purpose: demonstrating ontology workflow using Excel sheet UKON 2018 at Keele University

Introduction

We start the programme by defining the name space

```clojure
(ns amino.amino
 (:use [tawny.owl]
    [tawny.pattern]
    [tawny.repl]
    [tawny.reasoner]
    [dk.ative.docjure.spreadsheet])
(:require [clojure.string :as str]
           [tawny.protocol :as p])
(:import [java.net URLEncoder]))
```

Some Tawny-OWL functions to read and arrange information extracted from the Excel sheet to create classes for the ontology

```clojure
(defn row-inf
 "Extract and save one row information into one lazy sequence.
 SHEET - is the sheet name.
 ROW - is row number."
[sheet row]
(map #(and %
    (str/replace (str/trim %) #"\s+" "))
    (map read-cell
    (cell-seq (take 1 (drop row (row-seq sheet)))))))

(defn create-class
 "Create one class using a name in a string format.
 NAME - name of the class as string."
[name]
(intern-owl-string name (owl-class name)))

(defn create-classes
 "Create classes using a sequence of names (strings).
 NAMES - strings sequence of names of the classes."
[names]
(doall
 (map create-class names)))

(defn create-sub-classes
 "Define subclasses giving super class.
 SUPER - the super class
 ARGS - subclasses in a list"

Amino Acids Ontology

Here the ontology and class definitions

(defontology amino-acids-ont
    :comment "This is Amino Acids ontology"
    :iri "http://ncl.ac.uk/amino-acids-ont")

(defclass PhysicoChemicalProperty)

(defclass NameProperty)

First row of the sheet used to define the classes using the above functions

(def classes (row-inf sheet 0))

(create-classes classes)

Then we adjust the super class for the following classes

(refine Hydrophobicity :super PhysicoChemicalProperty)

(refine Charge :super PhysicoChemicalProperty)

(refine Size :super PhysicoChemicalProperty)

(refine Polarity :super PhysicoChemicalProperty)

(refine SideChainStructure :super NameProperty)

(refine Short :super NameProperty :disjoint LTR)

(refine LTR :super NameProperty :disjoint Short)

Object properties

(defproperty hasHydrophobicity
    :domain AminoAcid
(defoproperty hasCharge  
  :domain AminoAcid  
  :range Charge)

(defoproperty hasPolarity  
  :domain AminoAcid  
  :range Polarity)

(defoproperty hasSize  
  :domain AminoAcid  
  :range Size)

(defoproperty hasSideChainStructure  
  :domain AminoAcid  
  :range SideChainStructure)

Annotation properties
(defaproperty hasLongName  
  :comment "attach the long name to the amino acid")

(defaproperty hasShortName  
  :comment "attach the short name to the amino acid")

This is the wrapping function to fill the patterns of the ontology using extracted information

(defn amino-acid  
  "Define new amino acid based on the row information for the Excel spreadsheet.  
  EXAMPLES - list of the amino acid information as strings extracted  
  from the spreadsheet"  
  [examples]  
  (let [aa (owl-class (first examples)  
                  :super AminoAcid  
                  (owl-some hasCharge  
                              (entity-for-string amino-acids-ont (nth 
examples 4)))))  
       (owl-some hasHydrophobicity  
                  (entity-for-string amino-acids-ont (nth 
examples 3))))  
       (owl-some hasPolarity  
                  (entity-for-string amino-acids-ont (nth 
examples 6))))  
       (owl-some hasSideChainStructure  
                  (entity-for-string amino-acids-ont (nth 
examples 7)))]
All amino acids from the Excel sheet are added using the following statement

```
(doall
  (map amino-acid 
    (for [r (range 1 20)] 
      (row-inf sheet r)))
```

### Amino Acids table

This is the table used as Excel sheet to build Amino acids ontology

<table>
<thead>
<tr>
<th>Amino Acid</th>
<th>Short</th>
<th>LTR</th>
<th>Hydrophobicity</th>
<th>Charge</th>
<th>Size</th>
<th>Polarity</th>
<th>Side Chain Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arginine</td>
<td>Arg</td>
<td>R</td>
<td>Hydrophilic</td>
<td>Positive</td>
<td>Large</td>
<td>Polar</td>
<td>Aliphatic</td>
</tr>
<tr>
<td>Asparagine</td>
<td>Asn</td>
<td>N</td>
<td>Hydrophilic</td>
<td>Neutral</td>
<td>Small</td>
<td>Neutral</td>
<td>Aliphatic</td>
</tr>
<tr>
<td>Aspartate</td>
<td>Asp</td>
<td>D</td>
<td>Hydrophilic</td>
<td>Negative</td>
<td>Small</td>
<td>Polar</td>
<td>Aliphatic</td>
</tr>
<tr>
<td>Cysteine</td>
<td>Cys</td>
<td>C</td>
<td>Hydrophobic</td>
<td>Neutral</td>
<td>Small</td>
<td>Polar</td>
<td>Aliphatic</td>
</tr>
<tr>
<td>Glutamate</td>
<td>Glu</td>
<td>E</td>
<td>Hydrophilic</td>
<td>Negative</td>
<td>Small</td>
<td>Polar</td>
<td>Aliphatic</td>
</tr>
<tr>
<td>Glutamine</td>
<td>Gln</td>
<td>Q</td>
<td>Hydrophilic</td>
<td>Neutral</td>
<td>Large</td>
<td>Polar</td>
<td>Aliphatic</td>
</tr>
<tr>
<td>Glycine</td>
<td>Gly</td>
<td>G</td>
<td>Hydrophobic</td>
<td>Neutral</td>
<td>Tiny</td>
<td>NonPolar</td>
<td>Aliphatic</td>
</tr>
<tr>
<td>Histidine</td>
<td>His</td>
<td>H</td>
<td>Hydrophilic</td>
<td>Positive</td>
<td>Large</td>
<td>Polar</td>
<td>Aromatic</td>
</tr>
<tr>
<td>Isoleucine</td>
<td>Ile</td>
<td>I</td>
<td>Hydrophobic</td>
<td>Neutral</td>
<td>Large</td>
<td>NonPolar</td>
<td>Aliphatic</td>
</tr>
<tr>
<td>Leucine</td>
<td>Leu</td>
<td>L</td>
<td>Hydrophobic</td>
<td>Neutral</td>
<td>Large</td>
<td>NonPolar</td>
<td>Aliphatic</td>
</tr>
<tr>
<td>Lysine</td>
<td>Lys</td>
<td>K</td>
<td>Hydrophilic</td>
<td>Positive</td>
<td>Large</td>
<td>Polar</td>
<td>Aliphatic</td>
</tr>
<tr>
<td>Methionine</td>
<td>Met</td>
<td>M</td>
<td>Hydrophobic</td>
<td>Neutral</td>
<td>Large</td>
<td>NonPolar</td>
<td>Aliphatic</td>
</tr>
<tr>
<td>Phenylalanine</td>
<td>Phe</td>
<td>F</td>
<td>Hydrophobic</td>
<td>Neutral</td>
<td>Large</td>
<td>NonPolar</td>
<td>Aromatic</td>
</tr>
<tr>
<td>Proline</td>
<td>Pro</td>
<td>P</td>
<td>Hydrophobic</td>
<td>Neutral</td>
<td>Small</td>
<td>NonPolar</td>
<td>Aliphatic</td>
</tr>
<tr>
<td>Serine</td>
<td>Ser</td>
<td>S</td>
<td>Hydrophilic</td>
<td>Neutral</td>
<td>Tiny</td>
<td>Polar</td>
<td>Aliphatic</td>
</tr>
<tr>
<td>Theonine</td>
<td>Thr</td>
<td>T</td>
<td>Hydrophilic</td>
<td>Neutral</td>
<td>Tiny</td>
<td>Polar</td>
<td>Aliphatic</td>
</tr>
<tr>
<td>Tryptophan</td>
<td>Trp</td>
<td>W</td>
<td>Hydrophobic</td>
<td>Neutral</td>
<td>Large</td>
<td>NonPolar</td>
<td>Aromatic</td>
</tr>
<tr>
<td>Tyrosine</td>
<td>Try</td>
<td>Y</td>
<td>Hydrophobic</td>
<td>Neutral</td>
<td>Large</td>
<td>Polar</td>
<td>Aromatic</td>
</tr>
<tr>
<td>----------</td>
<td>-----</td>
<td>---</td>
<td>-------------</td>
<td>---------</td>
<td>-------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>Valine</td>
<td>Val</td>
<td>V</td>
<td>Hydrophobic</td>
<td>Neutral</td>
<td>Small</td>
<td>NonPolar</td>
<td>Aliphatic</td>
</tr>
</tbody>
</table>

Last step is to save our ontology as an owl file

```lisp
(reasoner-factory :hermit)
(save-ontology "aminoacid.owl" :owl)
```
References


- 242 -


JavaTpoint (2020). first-order logic in artificial intelligence.


Ontotext (2020). *what are ontologies?*


– (2005). Representing Specified Values in OWL: “value partitions” and “value sets”.


