A foundation for ontology modularisation

by

Zubeida C. Dawood

Submitted to the Department Computer Science
in fulfilment of the requirements for the degree of

Doctor of Philosophy in Computer Science

at the

UNIVERSITY OF CAPE TOWN

November 2017

A foundation for ontology modularisation

by

Zubeida C. Dawood

Submitted to the Department Computer Science
on 06 November 2017, in fulfilment of the
requirements for the degree of
Doctor of Philosophy in Computer Science

Abstract

There has been great interest in realising the Semantic Web. Ontologies are used to define Semantic Web applications. Ontologies have grown to be large and complex to the point where it causes cognitive overload for humans, in understanding and maintaining, and for machines, in processing and reasoning. Furthermore, building ontologies from scratch is time-consuming and not always necessary. Prospective ontology developers could consider using existing ontologies that are of good quality. However, an entire large ontology is not always required for a particular application, but a subset of the knowledge may be relevant. Modularity deals with simplifying an ontology for a particular context or by structure into smaller ontologies, thereby preserving the contextual knowledge. There are a number of benefits in modularising an ontology including simplified maintenance and machine processing, as well as collaborative efforts whereby work can be shared among experts. Modularity has been successfully applied to a number of different ontologies to improve usability and assist with complexity. However, problems exist for modularity that have not been satisfactorily addressed. Currently, modularity tools generate large modules that do not exclusively represent the context. Partitioning tools, which ought to generate disjoint modules, sometimes create overlapping modules. These problems arise from a number of issues: different module types have not been clearly characterised, it is unclear what the properties of a 'good' module are, and it is unclear which evaluation criteria applies to specific module types. In order to successfully solve the problem, a number of theoretical aspects have to be investigated. It is important to determine which ontology module types are the most widely-used and to characterise each such type by distinguishing properties. One must identify properties that a 'good' or 'usable' module meets. In this thesis, we investigate these problems with modularity systematically. We begin by identifying dimensions for modularity to define its foundation: use-case, technique, type, property, and evaluation metric. Each dimension is populated with sub-dimensions as fine-grained values. The dimensions are used to create an empirically-based framework for modularity by classifying a set of ontologies with them, which results in dependencies among the dimensions. The formal framework can be used to guide the user in modularising an ontology and as a starting point in the modularisation process. To solve the problem with module quality, new and existing metrics were implemented into a novel tool TOMM, and an experimental evaluation with a set of modules was performed resulting in dependencies between the metrics

and module types. These dependencies can be used to determine whether a module is of good quality. For the issue with existing modularity techniques, we created five new algorithms to improve the current tools and techniques and experimentally evaluate them. The algorithms of the tool, NOMSA, performs as well as other tools for most performance criteria. For NOMSA's generated modules, two of its algorithms' generated modules are good quality when compared to the expected dependencies of the framework. The remaining three algorithms' modules correspond to some of the expected values for the metrics for the ontology set in question. The success of solving the problems with modularity resulted in a formal foundation for modularity which comprises: an exhaustive set of modularity dimensions with dependencies between them, a framework for guiding the modularisation process and annotating module, a way to measure the quality of modules using the novel TOMM tool which has new and existing evaluation metrics, the SUGOI tool for module management that has been investigated for module interchangeability, and an implementation of new algorithms to fill in the gaps of insufficient tools and techniques.

Acknowledgments

First and foremost, I would like to thank the Almighty, for giving me the strength and wisdom to finish this research.

I would first like to thank my academic supervisor, Dr. Maria Keet. Although we lived in different cities, I would like to say that the door to her office was always open whenever I ran into a problem or had a question (no matter how silly) about my research. I am grateful to Dr. Keet for having the patience of untangling my ideas and steering me in the right direction, and for opening up many exciting research opportunities for me.

I would like to thank my co-authors of publications for the successful and interesting collaboration of work: Claudia D'Amoto and Agnieszka Lawrynowicz on the work on modularising the DMOP ontology for reasoning, Pablo R. Fillottrani and Karina Cenci on the work on inter-model links within conceptual data models, and once again, my supervisor Dr. Maria Keet, who assisted me in learning to write and formulate the work.

Thanks to my friends, Thulani Mashiane and Glenn Masango, for helping me through the stress with much-needed laughs and snacks, and not letting me give up. I would like to express gratitude to my husband, Ismail Dawood, for trying to understand my research, helpful criticism from a different point of view, and encouraging my academic pursuits. Lastly, I want to express my gratitude for my family, who have been a constant source of support and faith throughout this journey.

Published work

The work included in this thesis is supported by several publications:

Chapter 3

1. Zubeida Casmod Khan and C. Maria Keet. Toward a framework for ontology modularity. In *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT'15)*. ACM Conference Proceedings, 2015. 28-30 September 2015, Stellenbosch, South Africa
2. Zubeida Casmod Khan and C. Maria Keet. An empirically-based framework for ontology modularisation. *Applied Ontology*, 10(3-4):171–195, 2015
3. Zubeida Casmod Khan and C. Maria Keet. ROMULUS: the repository of ontologies for multiple uses populated with mediated foundational ontologies. *Journal of Data Semantics*, 5(1):19–36, 2016
4. Zubeida Casmod Khan, C. Maria Keet, Pablo R. Fillottrani, and Karina Cenci. Experimentally motivated transformations for intermodel links between conceptual models. In *20th Conference on Advances in Databases and Information Systems (ADBIS'16)*, volume 9809 of *Lecture Notes in Computer Science LNCS*, pages 104–118. Springer, 2016. August 28-31, Prague, Czech Republic

Chapter 4

5. C. Maria Keet, Claudia d'Amato, Zubeida Casmod Khan, and Agnieszka Lawrynowicz. Exploring reasoning with the DMOP ontology. In *3rd Workshop on Ontology Reasoner Evaluation (ORE'14)*, CEUR Workshop Proceedings, pages 64–70. CEUR-WS.org, 2014. July 1, Vienna, Austria
6. Zubeida Khan and C. Maria Keet. Feasibility of automated foundational ontology interchangeability. In *19th International Conference on Knowledge Engineering and Knowledge Management (EKAW'14)*, volume 8876 of *LNAI*, pages 225–237. Springer, 2014. 24 - 28 November 2014, Linköping, Sweden
7. Zubeida Casmod Khan and C. Maria Keet. SUGOI: automated ontology interchangeability. In Patrick Lambrix, Eero Hyvönen, Eva Blomqvist, Valentina Presutti, Guilin Qi, Uli Sattler, Ying Ding, and Chiara Ghidini, editors, *Knowledge Engineering and Knowledge Management - EKAW 2014 Satellite Events*, volume 8982 of *Lecture Notes in Computer Science*, pages 150–153. Springer, 2014. Linköping, Sweden, November 24-28, 2014. Revised Selected Papers
8. Zubeida Casmod Khan. Evaluation metrics in ontology modules. In *29th International Workshop on Description Logics (DL'16)*, volume 1577 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016. 22-25 April 2016, Cape Town, South Africa

9. Zubeida Casmod Khan and C. Maria Keet. Dependencies between modularity metrics towards improved modules. In *20th International Conference on Knowledge Engineering and Knowledge Management (EKAW'16)*, Lecture Notes in Artificial Intelligence LNAI, pages 19–23. Springer, 2016. 19-23 November 2016, Bologna, Italy

10. Zubeida Casmod Khan and C. Maria Keet. Automated ontology interchangeability towards improved modules. In *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT'17)*. ACM Conference Proceedings, 2017. 26-28 September 2017, Bloemfontein, Free State, South Africa

11. Zubeida Casmod Khan and C. Maria Keet. Automatic modularisation with algorithms for abstraction and expressiveness. (in preparation for submission to an international conference)

Contents

1 Introduction **15**
- 1.1 Background and motivation . . . 15
- 1.2 Problem statement . . . 21
- 1.3 Motivation . . . 21
- 1.4 Research objectives . . . 22
- 1.5 Research methodology . . . 23
- 1.6 Organisation of thesis . . . 23

2 State of the art **25**
- 2.1 What is a module? . . . 25
- 2.2 Analysis of existing modules . . . 26
  - 2.2.1 Ontologies as a set of modules . . . 26
  - 2.2.2 Modules with less detail . . . 27
  - 2.2.3 Conclusions from the analysis . . . 28
- 2.3 Overview of module dimensions . . . 28
- 2.4 Properties of modules . . . 29
- 2.5 Use-cases of modules . . . 29
- 2.6 Techniques of the modularisation process . . . 30
- 2.7 Types of modules . . . 32
- 2.8 Evaluation metrics of modules . . . 33
- 2.9 Discussion . . . 36
- 2.10 Conclusion . . . 37

3 Dimensions of modularisation **38**
- 3.1 Definition . . . 39
- 3.2 Use-cases . . . 39
- 3.3 Types . . . 42
  - 3.3.1 Functional modules . . . 42
  - 3.3.2 Structural modules . . . 43
  - 3.3.3 Abstraction modules . . . 44
  - 3.3.4 Expressiveness modules . . . 45
- 3.4 Properties . . . 45
  - 3.4.1 Properties of a module . . . 45
  - 3.4.2 Properties of a set of related modules . . . 47
- 3.5 Techniques . . . 49
  - 3.5.1 Graph theory approaches . . . 49
  - 3.5.2 Statistical approaches . . . 50
  - 3.5.3 Semantic approaches . . . 51

3.6 Classifying modules: An experimental evaluation . . . 53
3.6.1 Materials and methods . . . 54
3.6.2 Results and Discussion . . . 55
3.7 A framework for ontology modularity . . . 61
3.7.1 Methodology . . . 61
3.7.2 Dependencies between dimensions . . . 61
3.8 Evaluating the framework . . . 66
3.8.1 Ontology case-studies . . . 66
3.8.2 Conceptual data model case-studies . . . 69
3.9 Discussion . . . 71
3.10 Conclusion . . . 73

4 Theories and techniques for modularitsation 75
4.1 Issues with modularisation with existing resources . . . 75
4.1.1 Case-study: ROMULUS's modules . . . 76
4.1.2 Case-study: Modularising the DMOP ontology . . . 78
4.1.3 Problems with existing modularisation resources . . . 79
4.2 Evaluation metrics for modules . . . 80
4.2.1 Structural criteria . . . 81
4.2.2 Logical Criteria . . . 86
4.2.3 Relational criteria . . . 88
4.2.4 Information hiding . . . 93
4.2.5 Richness criteria . . . 94
4.2.6 Tool for ontology module metrics . . . 95
4.2.7 Experimental evaluation . . . 98
4.3 Ontology interchangeability . . . 106
4.3.1 Interchangeability algorithm design . . . 107
4.3.2 SUGOI ontology interchangeability tool . . . 109
4.3.3 Interchangeability with modular ontologies . . . 112
4.3.4 Experimental Evaluation . . . 118
4.4 Ontology modularisation techniques . . . 122
4.4.1 New modularisation algorithms . . . 123
4.4.2 Illustration of the algorithms . . . 129
4.4.3 NOMSA Modularisation tool . . . 132
4.4.4 Experimental evaluation . . . 133
4.5 Discussion . . . 137
4.6 Conclusion . . . 138

5 Conclusion and future research 140
5.1 Conclusion . . . 140
5.2 Future research . . . 145

A Classification of the set of modules 147

B The Burger Ontology 156

List of Figures

1.1 Difficulty using the BioPortal visualisation tool for the large NCI cancer ontology. . . . 17
1.2 Relations between entities which cause difficulty in creating compact modules. . . . 19
1.3 The flow of activities to be performed for the research, the dotted arrows represent backward flows, i.e., returning to a previous activity. 24

2.1 The traversal approach in module extraction for a restaurant ontology. 30

3.1 Overlapping modules in an ontology system. . . . 47
3.2 Mutually exclusive modules in an ontology system. . . . 48
3.3 Union equivalence occurs for modules A, B, and C; the union of them is equivalent to ontology O. . . . 48
3.4 The frequency of each use-case for the set of 189 modules. . . . 56
3.5 The frequency of each type for the set of 189 module. . . . 58
3.6 The frequency of each technique for the set of 189 modules. . . . 59
3.7 The frequency of each property among modules. . . . 60
3.8 A high-level view of the framework for modularity. . . . 62
3.9 The dependencies between use-cases and module types. . . . 63
3.10 The dependencies between module types and technique. . . . 64
3.11 The dependencies between techniques and properties. . . . 65
3.12 The dependencies between the module dimensions for QUDT modules. 67
3.13 A cognitive overload scenario: a conceptual data model on the governance domain with intermodel assertions between modules. . . . 70
3.14 An integration scenario: The intermodel assertions between Flights models in EER and UML. . . . 71

4.1 The metadata pertaining to module details from the DOLCE-Endurants module in ROMULUS. . . . 78
4.2 An ontology's atomic decomposition. . . . 83
4.3 A source ontology and corresponding module for which we calculate intra-module distance. . . . 84
4.4 Two sets of modules S1, S2 with inter-related links. . . . 89
4.5 The interface of TOMM. . . . 97
4.6 A log file for the kisao-partition ontology module generated by TOMM. 97
4.7 The set of metrics that can be measured for each module type. . . . 103

4.8 The terminologies used for the files involved in interchangeability using the SAO ontology. 108
4.9 The interface of the online desktop version of SUGOI. 111
4.10 The interface of the online desktop version of SUGOI-Gen. 112
4.11 Examples of interchanging the dmop:DataType and dmop:Strategy domain entities from ${}^{s}\mathcal{M}_i$ DOLCE to ${}^{t}\mathcal{M}_i$ GFO with SUGOI, using equivalence and subsumption mappings. Source: [81] 113
4.12 The position of the sao:Membrane Surface class in source and target ontologies. Source: [84]. 114
4.13 The mapping file showing alignments for classes between DOLCE and BFO foundational ontologies. 119
4.14 Generating a high-level abstraction module with depth = 2 from the GFO ontology. 125
4.15 The burger ontology to which the algorithms are applied; see text for details. 131
4.16 The interface of NOMSA. 132

5.1 The set of metrics that can be measured for each module type. 143
5.2 A high-level view of the framework for modularity. 144

List of Tables

2.1 A summary of the current techniques of modularisation 33

3.1 The modularisation techniques implemented by each tool. 53
3.2 Classifying the conceptual data model projects using the framework for modularity. 72

4.1 Size metrics for DMOP and its related modules. 80
4.2 The farness values for each entity of the source ontology and corresponding module. 85
4.3 The farness and strength of relation (1/farness) values for each entity of the ontology. 86
4.4 The number of modules, *NM*, that have to be considered to relate two entities in the set of modules (S1). 90
4.5 The number of modules, *NM*, that have to be considered to relate two entities in the set of modules (S2). 90
4.6 The number of external links, *NEL*, that have to be considered to relate M1 to other modules in the system. 91
4.7 A summary of the set of evaluation metrics with their expected value range and values that are considered good. 96
4.8 Averages for the structural metrics of the set of modules. 99
4.9 Medians for the structural metrics of the set of modules. 100
4.10 Average, median, and boolean values for the logical, richness, information hiding, and relational criteria. 101
4.11 The metrics for the QUDT ontology modules generated by TOMM . 104
4.12 The metrics for the Pescado disease ontology generated by TOMM . . 105
4.13 The metrics for the Symptom skin ontology module generated by TOMM. 105
4.14 A comparison of the ${}^{s}\mathcal{O}_m$ and ${}^{t}\mathcal{O}_m$ for the interchangeability and reasoning 120
4.15 The metrics for the ${}^{s}\mathcal{O}_m$ and ${}^{t}\mathcal{O}_m$ ontologies. 121
4.16 The classes of the burger ontology with the number of referencing axioms. 130
4.17 A comparison of three features of modularisation tools against NOMSA 134
4.18 The average values for the metadata for all the generated modules . . 136

A.1 The classification of the set of modules for the use-case, type, property, and technique dimensions. 147

Chapter 1

Introduction

The vision of the Semantic Web is to allow for machines to process web content meaningfully thereby improving the general web experience for users. Ontologies are used as resources for Semantic Web applications because they describe a specific domain such that machines are able to identify, classify and understand the domain thereby improving the web experience for an average user as one usage scenario of ontologies.

In the early 1990s, Gruber [55] defined an ontology, in the context of computer science, as a *formal specification of a conceptualisation.* However, this is not completely accurate because of the intended meaning of the word 'conceptualisation', which refers to a particular state of affairs in the world. Therefore, Guarino [56] has redefined an ontology as *"a logical theory accounting for the intended meaning of a formal vocabulary, i.e. its ontological commitment to a particular conceptualisation of the world. The intended models of a logical language using such a vocabulary are constrained by its ontological commitment. An ontology indirectly reflects this commitment (and the underlying conceptualization) by approximating these intended models."*

Since then, ontologies have been widely applied to various subject domains and applications. Ontologies are commonly used for natural language processing, to realise the goals of the Semantic Web, data integration, ontology-based data access etc. Towards the goal of realising the Semantic Web, ontologies facilitate heterogeneous interoperability. Ontologies that describe large, well-defined domains are consequently large and complex in nature. Examples of such enormous ontologies (those that contain $>$ 100 000 entities) include the BioModels ontology [64], the SNOMED CT ontology [129] and the FMA ontology [133]. These ontologies contain hundreds of thousands of terms.

1.1 Background and motivation

Ontologies are important in the Semantic Web for a number of reasons. They can be used to facilitate common understanding of a domain across many heterogeneous systems and people. For instance, suppose food blogs are built upon a common underlying ontology, then applications can easily extract information from them which

can be automatically used in related websites or blogs such as recipe blogs. The rigorous axiomatisations allowed in an ontology enable it to assist with classifications. In domains such as medicine, a patient's symptoms can easily be stored in an ontology, and by reasoning one can predict health conditions and possible diseases. One of the primary reasons for using ontologies is that parts or modules of other ontologies can be easily imported or used in an ontology without the need for redesigning everything from scratch. One example of this is the use of foundational ontologies for representing higher-level concepts such as processes and events in a domain ontology. Search engines such as Google implement light-weight ontologies in order to improve search results and gain relevant results for a query. For instance, there is the schema.org vocabulary[1] that contains entities and relationships of various domains that are used to mark-up many web pages. There is an enormous amount of information online and ambiguity among the information. The implemented ontology reduces ambiguity, narrows down a search and returns what is required for a particular search query.

We now illustrate the importance of ontologies with a motivating example.

Example 1 *Consider that a bio-medicine website contains textual knowledge describing the Ebola disease, and it states that* Ebola has a symptom of coughing *and,* Ebola is caused by a Virus. *How would other web tools, without human intervention, recognise the fact that this web page describes the Ebola disease symptoms and causes, and not interpret in incorrectly as say, a computer virus? For instance, it could be that a hospital web application may want to automatically integrate this information on the Ebola disease. An ontology could be used to describe the knowledge such that semantic interoperability is achieved. This can be done as follows.*

The knowledge is described within an ontology using the Web Ontology Language (OWL). OWL is serialised in the Resource Description Framework/ eXtensible Markup Language (RDF/XML). RDF provides a graph-like infrastructure for describing this knowledge by using triples. Triples consist of a subject, predicate and object. For describing the knowledge on Ebola, Ebola *would be the subject,* causedBy *would be the predicate, and* Virus *would be the object. RDFS (RDF Schema) then adds semantics to these triples thereby allowing for a high level of expressivity in the ontology. One could state that* Ebola*,* Virus *and,* Cough *are classes and that* causedBy *and* hasSymptom *are object properties. With OWL, cardinality constraints could be added to the data to specify that* Ebola *has at least 3 symptoms. Using OWL, one could also state that the* Ebola *entity on the bio-medicine website is the same as the one on the hospital website thus integrating the data from the bio-medicine website to the hospital web application as required.*

Since ontologies have grown to be extremely large and complex, this brings about great difficulty for both machines and humans. For demonstrating the difficulties faced by machines, we have tried two different ontology processing tools using the large NCI cancer ontology [54]. First, the BioPortal visualisation tool [163] takes several minutes to load large taxonomy branches of an ontology (see Figure 1.1) using a machine with an Intel Core 2 Duo Processor with 4GB of RAM. Next, we

[1] `http://schema.org/` last accessed: 20 June 2017.

used the OWL metrics tool[2] to compute its metrics. It took 12 minutes to process before it returned an ontology parsing error.

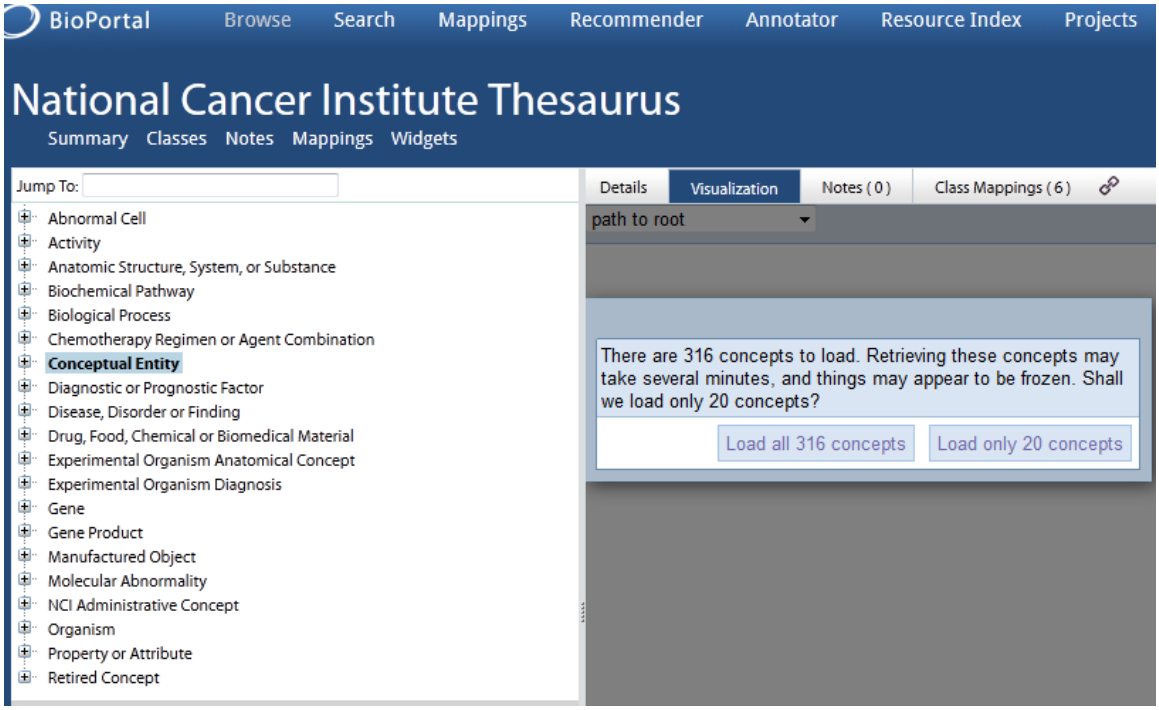


Figure 1.1: Difficulty using the BioPortal visualisation tool for the large NCI cancer ontology.

For humans, ontologies are confusing and not easily understood and maintained by developers and users, and as such, hinder ontology development, usage, reuse and collaborative efforts. On the other hand, due to the excessive data within the ontologies, and having large ontologies in expressive languages, reasoners and tools perform slowly and sometimes malfunction. At times, building an ontology from scratch is not necessary if the domain is well-designed in an existing ontology. While it is important to represent every building block of a domain in an ontology, developers and users sometimes require only specific subsets of an ontology for a particular use-case.

Over the last few years, there has been a growth in using *modularity* to assist with problems in different domains. The general concept of modularity refers to dividing and separating the components of a large system such that modules can be recombined. This concept is present in the biological field, whereby it refers to the fact that organisms contain modules, in mathematics whereby a module is a generalisation of the idea of vector space over a field, and in the cognitive science field, whereby scientists argue whether the mind is composed of modules that are independent of each other and focus on different sub-domains of the mind, to name a few. Closer to the current area of research, modularity has been applied in computer science,

[2] `http://mowl-power.cs.man.ac.uk:8080/metrics/` last accessed: 20 June 2017

particularly in software design and development to allow collaborative work between programmers and promote code reuse. Seeing that modularity has been applied to numerous and heterogeneous fields of study, it is no surprise that ontology developers have considered modularity to aid with large and complex ontologies.

Modularity is used to simplify and downsize an ontology for the task at hand; to modularise a large ontology into smaller manageable ontologies. In terms of ontology modularity, it is defined informally as the possibility to perceive a large knowledge repository as a set of smaller repositories or modules that together compose the entire repository [123]. It is required in ontology development and usage when one needs to hide or remove knowledge that is not required for the use-case. Modularity has been successfully applied to a number of different ontologies to improve usability and assist with complexity. Examples include the myExperiment ontology [111], which is a collaborative environment where scientists publish and share their work-flows and experiment plans among groups, the Semantic Sensor Net ontology where there are various modules to describe sensors and observations [71], and BioTop ontologies for life sciences in which the principle of modularisation have been applied [140]. There are many different types of ontology modules, such as language expressivity modules, domain-specific modules, more/less-detailed modules, to name a few.

In recent years, there has been a proliferation of numerous methods and tools such as Swoop [73], Protégé [110] and OWL module extractor tool [30] which aid in modularity engineering tasks. However, previous work [80] has shown that the modules generated by these tools are biased toward overlapping modules, which results in large ontology modules due to the tight cohesion, or the extent to which entities are related to each other, and underlying logic of ontologies. Overlapping modules are those that share common knowledge. In some cases, smaller disjoint modules are required. An evaluation of these tools indicate that in most cases, the sizes of the resulting modules are quite large and in some cases almost the same as the original ontology [35], hence they are not effective for the intended task. For instance, we tried to modularise the Data Mining OPtimization (DMOP) ontology [75] with several modularisation tools, but all modules were too large to use [78], and extracting content on object properties from DOLCE with the 'copy' feature, their asserted characteristics such as transitivity were not extracted [79]. Most modularisation algorithms are designed to identify relatively separated or isolated parts of an ontology. Therefore, when there are many class expressions and relations in ontology, few classes are isolated, resulting in large modules. This is illustrated in Fig. 1.2, where in the Nature reserve ontology, the Giraffe and Acacia-leaf entities exist and are linked by relations. Let us assume that we wish to use one of the existing tools such as the OWL module extractor [30] to extract a module. The underlying logic of existing ontology modularity tools is aimed at preserving the completeness of the ontology. Local completeness of a module states that every axiom that contains elements of a module local signature M in an ontology O, must be preserved in the module. In this case, if one were to extract a module about Flora, a difficulty arises because of the eats and eaten-by relations that occur between the Giraffe and Acacia-leaf entities. As such, when modularising an ontology of Flora, the tools preserve the Giraffe entity in the resulting module because the Flora entity has relations and axioms that involve the Giraffe entity. This modularisation

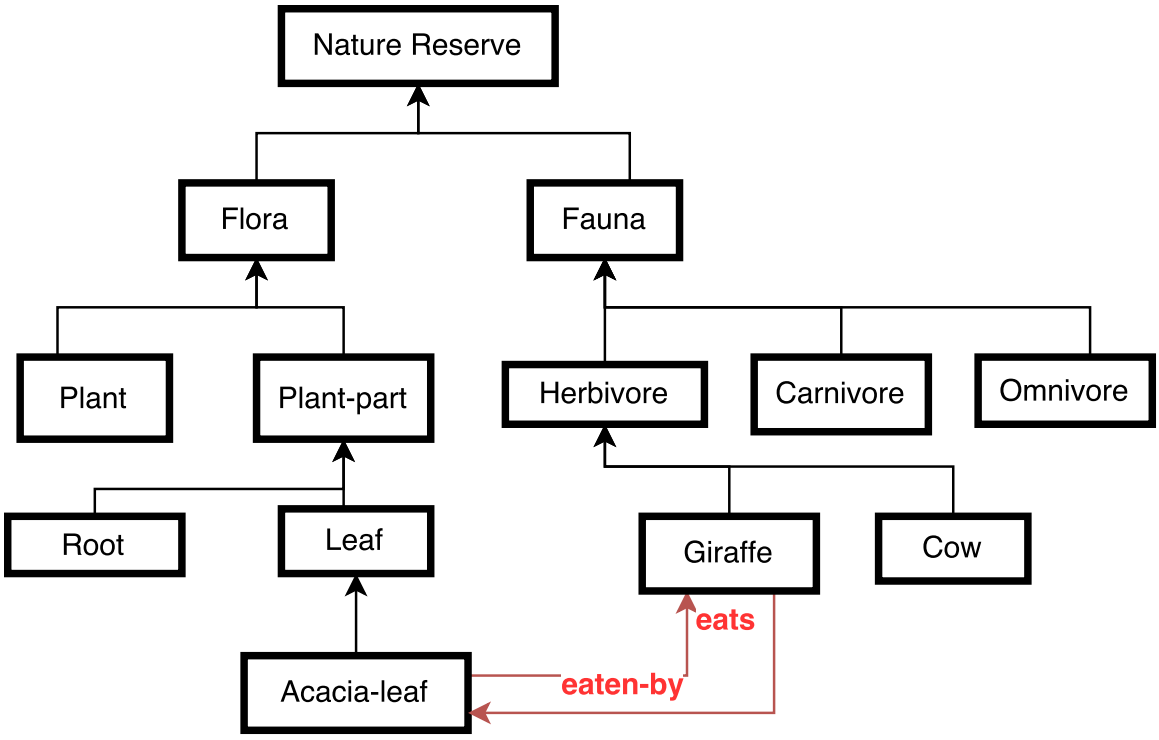


Figure 1.2: Relations between entities which cause difficulty in creating compact modules.

results in modules that are too large with some knowledge that is irrelevant for the domain.

Another issue concerning ontology modules is the lack of evaluation metrics. The few existing works on evaluation metrics focus on only some metrics that suit the modularisation technique [124, 137, 164], and there is not always a quantitative approach to calculate them. Overall, the metrics are not comprehensive enough to apply to a variety of modules, and it is unclear which metrics fare well with particular types of ontology modules. This has made it difficult to determine whether a module is a good module; it is unclear which metrics should be considered.

We illustrate the issues concerning modularity with the following example.

Example 2 *Consider a case where an ontology engineer wishes to develop an enterprise web application for an exciting book store. In order to compete with existing exciting interactive web utilities such as Pottermore which engages and challenges Harry Potter fans*[3]*, a simple website is not enough. The developer needs to create an interactive experience aimed at book fandoms*[4]*. Furthermore, the book store application must be machine readable in order to realise the vision of the Semantic Web. Rather than unnecessarily starting from scratch, the ontology developer decides to reuse existing resources from the web.*

[3] `https://www.pottermore.com/` last accessed: 20 June 2017.

[4] A fandom is a term used to describe fans who share a common interest such as the Harry Potter community, the Pokémon community, the Doctor Who community, and so on.

She has found the following existing resources: a general book ontology containing metadata about books such as authors, edition, etc., a Japanese anime ontology, containing a large amount of data about anime such as genre, protagonists, mangaka, etc., a toy ontology which contains toy product details, and a gazetteer ontology describing different places of geographical interest.

For the Japanese anime ontology, she wishes only to use parts of the ontology that contains information about anime that have originated from manga and novels. The taxonomy of the anime ontology is large, complicated, and too difficult for the developer to manually traverse through the entire ontology and understand it. For the toy ontology, there are two main branches, a Toy *entity with subclasses such as* Book-toy, Game-toy, Movie-toy *etc. , and a* Toy-property *entity containing knowledge about the properties of a toy, e.g., height, weight, colour, etc. The developer wishes to extract only the specific* Book-toy *entity and the generalised* Toy-property *entity. She wishes to use the gazetteer ontology to describe both fictional and real places that appear in books. The gazetteer ontology has a high level of detail and contains knowledge about the population of each city, neighbouring countries, the size of the country, etc. However, the developer does not require the high-level of detail of the ontology, but rather a simplified version.*

The developer decides that the manga and novel aspects of the anime ontology could be imported into the general book ontology. The toy ontology, however, should be separate from the book ontology, to allow toy specialists to update it separately. The gazetteer ontology should also be a separate module. By creating separate modules, the developer hopes to promote collaborative ontology development within a team and ease the validity and maintenance of the system. While the toy and gazetteer ontology are separate modules, they are also related to the book ontology, i.e., there should be axioms that link the modules. For instance, for the toy ontology, it could be that a scaled figure of Katniss Everdeen is based on the character, Katniss Everdeen, from the Hunger games trilogy book series. For the geographical ontology, it could be that Alicante is the city that exists in the Mortal Instruments book series.

This example opens up a number of questions for the developer. How should the developer extract reading material aspects from the anime ontology without traversing through the entire ontology? How should the developer isolate the Book-toy *and* Toy-property *entities from the toy ontology? How should the developer simplify the geographical ontology? It is true that there are existing ontology modularity techniques. However, which technique, if any, will be able to perform the aforementioned tasks. It is clear that the processes of extracting reading material aspects, isolating branches of interest, and taxonomy simplification are all different. Once the developer creates these modules, how should she measure the quality of these modules, considering that these modules have different properties? There are a number of existing modularity evaluation criteria. Again, it is unclear which metric should be used for these differing modules.*

How should the developer ensure that there are separate modules for the different aspects of the application, but communication still exists between the modules? Communication between the different modules implies that there may be overlapping or shared knowledge among the interrelated modules. How should the developer en-

sure that consistency is ensured among the modules, and provide support for module management and maintenance?

1.2 Problem statement

The problem of modularity, in terms of module extraction for satisfying different purposes of modularity such as disjoint modules, modules of context, modules of greater/less details, and in terms of the quality of the generated modules in terms of size and intra-module distance, has not been satisfactorily addressed. There is currently no foundation for modularity, i.e., a user has no guidance on how to initiate modularisation for a large ontology, which type of module to extract, which tool to use, and how to determine if the module is of good quality.

While a number of tools and methods have been developed to assist the user with ontology modularity, it is still unclear which tool is appropriate for a user's modularity task. Users face great difficulty in selecting an appropriate tool to use for modularisation, for the tools differ with regard to techniques and purposes and it is not clear what the appropriate modularity technique is for a purpose. Also, existing techniques are not sufficient in creating compact modules [35, 80]. The evaluation of modularisation techniques reveals that some tools fail to partition large ontologies because they focus on preserving the logical properties of the modules while others lose some of the relational properties of the ontologies, and that most tools generate views instead of module file outputs [116, 117].

Lastly, it is unclear how to determine whether a module is a good or bad module due to the lack of evaluation metrics. While some research has been done on modularity evaluation metrics, it is restrictive and focuses mainly on limited metrics such as size and local completeness. This is not comprehensive enough to apply to different types of modules. Pathak et. al [124] found that, for the criteria that do exist to evaluate modules, certain tools do not satisfy certain criteria. For instance, graph-based tools fail to achieve local completeness of the module.

Based on these unanswered questions, it is clear that the ontology developer faces great difficulty in ontology reuse. These unanswered questions pertain to ontology modularity choices. Currently, there is no theoretical foundation for ontology modularity. Selecting ontology modularity techniques for the different aspects of the enterprise bookstore application is a difficult task because there are different assumptions for each aspect, and there are multiple modularisation approaches. It is unclear which approach should be used for each specific assumption.

1.3 Motivation

Since there are various types of ontology modules, and also a number of modularity techniques, it is unclear on how to generate modules tailored for a particular scenario. A formal foundation for modularity, if exhaustive and clearly defined, would be of great use in solving the problems summarised in Section 1.2.

A multi-dimensional module engineering foundation will aid the ontology developer with ontology modularisation. When such a foundation is implemented, it could assist the user to create useful modules based on a specific application.

In order to solve the problem concerning insufficient modularisation evaluation metrics, the module foundation can include metrics as a dimension to be populated with a variety of metrics to assist with the different types of modules. This will assist the user to determine whether a module is good or bad quality.

1.4 Research objectives

To solve this problem of creating useful and better quality modules, we will look at existing techniques in ontology modularisation, with the aim of devising a foundation for modularity. In order to achieve this, it is necessary to, firstly, study and evaluate existing approaches in modularity evaluation criteria introduced elsewhere [35]. We will then identify relevant dimensions towards creating a modularity foundation. There is one main research question with a number of sub-questions which we wish to address.

1. How can one devise a formal foundation for modularity to improve existing modularity techniques and results? This can be further broken down into sub-questions.

   (a) What are the different types of modules that exist?

   (b) What are the properties with which we can characterise each module type?

   (c) What are the different purposes behind module creation?

   (d) Which techniques have been proposed perform different types of modularisation?

   (e) How can existing techniques for modularity be improved?

   (f) What are the criteria for 'good' or 'usable' ontology modules to meet?

   (g) Is there a way to link the above answers in order to guide the modularity process?

The questions, if answered correctly, should guide the user on how to modularise a particular ontology in the best way according to its properties.Based on these research questions, several tasks are to be performed which are:

- Identify dimensions associated with ontology modules.
- Create a formal foundation for ontology modularity.
- Perform an evaluation of the formal foundation for ontology modularity.
- Create a tool to calculate ontology evaluation metrics for modules.

- Perform an evaluation of the metrics.
- Design and improve on algorithms to create ontology modules of different types.
- Create tool support for the new algorithms.
- Perform an evaluation of the modularisation algorithms.

1.5 Research methodology

We aim to create a foundation to firstly assist the user in creating useful modules depending on the application, secondly to create algorithms, supported by implementations and evaluations, for automating modularisation for different techniques, and lastly to assist in evaluating various types of ontology modules with new and existing metrics that have been experimentally evaluated. The tasks to be performed to achieve the goal of creating and managing modules are outlined in Figure 1.3.

1.6 Organisation of thesis

Following this introductory Chapter, the remainder of the thesis is organised as follows. In Chapter 2, we provide a state of the art into modularity. This Chapter encompasses an analysis of existing modules and literature on formal definitions for modularity, properties, use-cases, techniques, types, and metrics of the modularisation process. The dimensions for modularity were identified from the literature of this Chapter. We have a discussion section and conclude the Chapter.

Chapter 3 focuses on the dimensions for modularisation that were identified in Chapter 2. The Chapter starts with first providing our own definition for modularity to resolve the shortcomings of previous definitions. Thereafter, four dimensions for modularity are defined: use-case, technique, types, and properties. Each dimension is then populated with illustrative examples. Thereafter, we present an empirically-based framework for ontology modularity, aimed at guiding the modularisation process. This follows with a section on classifying ontology modules in an experiment. Thereafter we present the framework for modularity aimed at guiding the modularisation process. An evaluation of the framework with ontology use-cases follows. Thereafter we have a discussion and conclude the Chapter.

Chapter 4 deals with new theories and techniques for modularity. The Chapter begins with an experiment to gain insight into the current state of the art concerning modularisation. Thereafter we investigate the evaluation of module metrics and introduce the Tool for Ontology Module Metrics (TOMM) which is aimed at generating evaluation metrics for modules. TOMM is evaluated experimentally which reveals some dependencies between metrics for modules and other dimensions of the framework. This follows with an investigation on module interchangeability, and we introduce Software Used for Gaining Ontology Interchangeability (SUGOI) which is a tool to swap modules. We perform an experiment using a set of modules for SUGOI

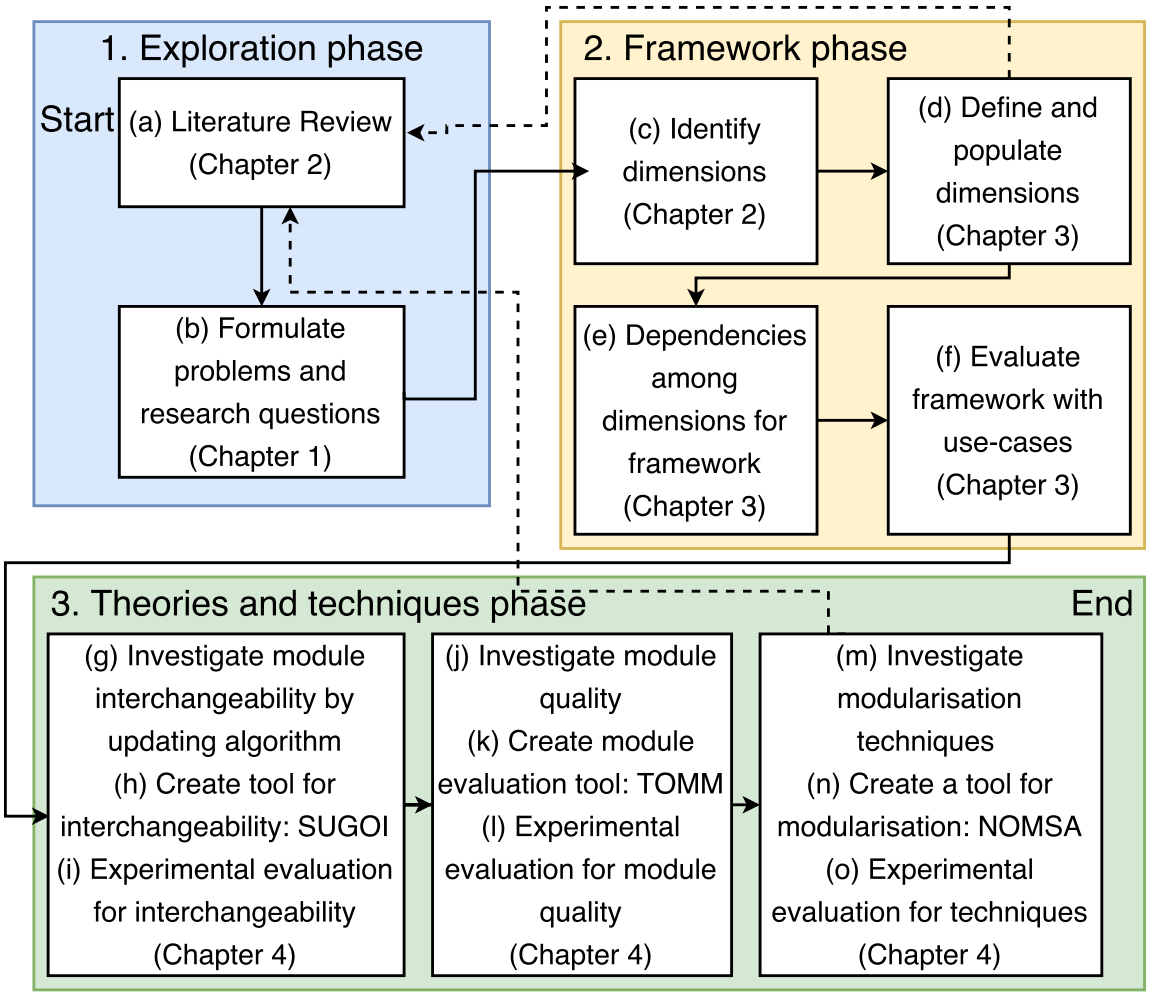


Figure 1.3: The flow of activities to be performed for the research, the dotted arrows represent backward flows, i.e., returning to a previous activity.

to determine whether interchanging modules has an effect on the modules' metrics. We then present five new algorithms for modularisation, an illustration of these algorithms using a toy Burger ontology, and a tool implementation of the algorithms: Novel Ontology Modularisation SoftwAre (NOMSA). An experimental evaluation of NOMSA and a comparison of NOMSA to other modularisation tools follows. The experiment reveals information about the features of NOMSA and the quality of NOMSA's generated modules. We then have a discussion and conclude the Chapter.

In Chapter 5, we summarise our contributions by answering the research questions proposed, discuss avenues for future research, and conclude the thesis.

Finally, there are two appendices. Appendix A is where we provide the classification for each module according to its dimensions from the experimental evaluation in Section 3.6. Appendix B lists the complete Burger ontology from the illustration of the algorithms of Section 4.4.2.

Chapter 2

State of the art

The aim of this chapter is to provide the state of the art concerning modularity in the various disciplines, and discuss current limitations of modularity, hence, to provide direction on the gaps that need to be filled, such as what is lacking in existing techniques and evaluation metrics and to identify components for a foundation for modularity. To begin, we analyse the current definitions for modularity in Section 2.1. We follow with an analysis of existing modules with the aim of identifying dimensions of modularity in Section 2.2. In Section 2.3 we give an overview of all the dimensions for modularity. Thereafter, we have sections 2.4- 2.8 on the literature on each dimension of modularity: properties, use-cases, techniques, types, and metrics. We follow with a discussion in Section 2.9 and conclude in Section 2.10.

2.1 What is a module?

In line with achieving our goal of creating a foundation for modularity, it is necessary to provide a clear definition of a module and to summarise the state of the art with regard to advances in modularisation. In this section, we describe and discuss existing definitions for modularisation to select a definition to use for our research.

Definition 1 *(Modularisation) Parent and Spaccapietra [123]: In its most generic meaning, it denotes the possibility to perceive a large knowledge repository (be it an ontology or a database) as a set of modules, i.e. smaller repositories that, in some way, are parts of and compose the whole thing.*

In definition 1, modularisation is defined at a generic level, as a way of interpreting a large domain as smaller parts such that the parts constitute the whole domain. However, in some cases, modules are parts but are not contained in a set that compose whole things. This is the case for the BioTopLite [140], and GFO-basic [61] ontology modules which are modules containing some of the domain knowledge, without being part of a set of that constitutes the whole domain.

Definition 2 *(Ontology module) Doran et al. [37]: An ontology module is a reusable component of a larger or more complex ontology, which is self-contained but bears a definite association to other ontology modules, including the original ontology.*

Definition 3 *(Ontology module) d'Aquin et al. [34]: We define an ontology module in a very general way as a part of an ontology: a module $\mathcal{M}_i(O)$ of an ontology O is a set of axioms, such that $Sig(\mathcal{M}_i(O)) \subseteq Sig(O)$.*

Definition 4 *(Modules) Del Vescovo et al. [158]: Modules are suitably small subsets of an ontology O that behave for specific purposes like the original ontology over a given signature $\sum$, i.e., a set of terms (classes and properties).*

Definitions 2, 3, 4 make mention of a module having an association to an original ontology. However, not all modules have an original ontology. The myExperiment ontology [111], is a decomposition of the domain into structural modules at the onset of ontology development hence there is no source ontology.

Definition 5 *(Module) Tsarkov [154]: A module is a subset of an ontology that captures all the knowledge the ontology contains about a given set of terms.*

In definition 5, modules are restricted to those that capture all the knowledge of the ontology concerning some given terms. This type of module definition is far too strict to hold for all types of modules that exist and is focussed on modules with the condition of logical completeness. For instance, partitioning tools do not guarantee that all the knowledge concerning some given terms is preserved in each partition; local completeness does not hold [124].

While existing literature does provide some definitions, we note that no definition is universally accepted. It seems the case that the existing definitions are unique to the problem at hand. The above module definitions hold for their respective application areas, but there are some gaps in the existing modularity definitions.

2.2 Analysis of existing modules

In this section, we describe some modules that exist. To do this, we use content analysis as a research method. This involves studying the body of work on modules to identify common components or patterns. We examine 13 different modularisation applications and analyse them according to what they achieve, their purpose, how they are created, and any properties that are apparent. This analysis aids in uncovering various dimensions of a module towards devising a foundation for modularity.

2.2.1 Ontologies as a set of modules

In this section, we describe cases where a large ontology is sub-divided into a set of several modules. We have selected various sets of ontology modules from existing literature and repositories. The LKIF legal ontology [67] is divided into a set of 15 functional modules (modules having different functions), which describe concepts from the legal and foundational domains. It is a set of modules that have been created at the onset of ontology development to describe the domain rather than a single large ontology. The OntoDM ontology [121] is an ontology of data mining

and is modular in structure. It is composed of three different functional aspects of the data mining domain, created at the onset of ontology development as well. The Geopolitical ontology has been designed to facilitate data exchange among systems managing information about different countries; it has been modularised according to different types of territories. A number of ontologies including Koala, Galen, Mereology, miniTambis. OWL-S, People, and Tambis were partitioned using atomic decomposition, in a study of the modular structure of ontologies by decomposition [159].

The PESCaDO ontology[162] is an environmental ontology that has been divided into 10 modules upfront. Each module covers a different aspect of the subject domain such as exceedances, data, geospatial information, etc. The OntoSpace project [11] is aimed at developing ontology-based methods suitable for supporting spatially-aware systems. There are a number of spatial modules used to cover different functions in the system. The Modular Unified Tagging Ontology (MUTO) [99] is an ontology for tagging and folksonomies. It has been designed so that different parts of the ontology are conceptually separate to allow for easy extensibility. Modules can be added to the ontology to describe specific sub-domains or tagging and folksonomies. Gist [103] is a foundational ontology designed to use for the business domain. The latest version, Gist 7.0, is modular in design and is made up of 18 modules that have been created at the onset. For the sets of modules we have analysed, they have been created as a modular ontology upfront, at the onset of ontology development, or they have been created by taking an existing large ontology and applying some modularisation techniques to create a set of modules.

2.2.2 Modules with less detail

In this section we describe cases where an ontology is modularised by removing some detail from the original ontology. We have selected various ontology modules from existing literature and repositories. BioTop [144], a top-domain level ontology, has been modularised into modules that include foundational ontologies, a less detailed module and a module for the chemistry subject domain. There are two modules of the GFO foundational ontology [61]. First, there is a lighter version with fewer theoretical concepts, GFO-Basic. Second, GFO [61] has been updated with a core biological ontology, GFO-Bio [65]. DOLCE [102] has a number of modular extensions that add domain-independent content to the ontology such as temporal and spatial relations. ROMULUS [80], a repository for foundational ontology usage and interoperability uses foundational ontology modules for DOLCE, BFO and GFO. In addition to these existing modules (GFO-Basic, TemporalRelations, etc.), new modules were created, such as OWL profile modules, and branch modules which cover knowledge about specific entities in the ontologies. The ROMULUS modules were created manually since using tools resulted in unsuitable modules; the sizes of the modules were too large [88].

The ChEBI ontology is an ontology of chemical entities in the biological domain. ChEBI has been updated with modular extensions for the purpose of maintenance and validation [60]. There are two extensions, a mapping to the BFO foundational

ontology, and a disjointness module that includes disjointness axioms about entities in ChEBI to assist with validating the ChEBI ontology with error detection. Most classes in the original ChEBI ontology are not disjoint, which could cause classification errors and omissions therefore a disjointness module helps with this. The Dumontier Lab for Biomedical Knowledge Discovery[1] hosts a range of biomedical domain ontologies. Each of their ontologies is separated into non-disjoint primitive taxonomies, and complex class expressions for DL reasoning. The primitive taxonomies are less-expressive modules as they do not contain the expressiveness afforded by the OWL-DL language, such as disjunction, negation etc., and can be expressed in simpler OWL profiles.

2.2.3 Conclusions from the analysis

By analysing the existing modules from Section 2.2.1- 2.2.2 with a content analysis, we have identified a few terms that have been used when describing modules. Firstly, there is the notion of 'use-case': why was the module created? Some of the modules described here were created for maintenance and validation. Then some of the modules were described as functional, taxonomies, etc. indicating that there may be different 'types' of modules. Next, there is mention of how the module is created or its 'technique', whether it was done manually or using an automatic approach such as partitioning. Another aspect is where modules are described as being part of a set of modules, being stand-alone, or having some extensions or refinements. These are some of the 'properties' that modules may exhibit. Lastly, there is some mention of the size of the resulting modules. Size is considered an 'evaluation metric'. The identification of these terms as dimensions of modules brings us a step closer to realising the goal of creating a foundation for modularity.

2.3 Overview of module dimensions

In order to define modularity and create a uniform method for creating and using ontology modules, we identify five dimensions that have an impact on ontology modules. The dimensions each have criteria, and we will demonstrate that approaching ontology modularity using different sets of criteria will result in different interpretations of modules.

1. **Use-cases:** There are reasons for modularity; thus our first dimension is Use-cases. A use-case could be that the module is created to assist with reasoning.

2. **Properties:** By studying existing ontology modules, we can identify properties that modules exhibit, e.g., some modules are disjoint from one another, while others have shared knowledge and are overlapping.

3. **Types:** There are types of modules. A module could be an OWL profile module, or a small branch of information.

[1] `http://dumontierlab.com/?page=ontologies` last accessed: 20 June 2017.

4. **Techniques:** In order to perform modularisation on a system, there are many methods which we refer to as techniques. A modularity technique could be inspired by graph theory, or a logical approach that is based on principles such as logical correctness.

5. **Evaluation metrics:** In order to assess the quality of an ontology module, we need an evaluation dimension. Dimensions such as encapsulation and size are commonly used to assess modules.

In the next section, we review the body of literature for each of these dimensions.

2.4 Properties of modules

Parent and Spaccapietra [123], define strategies that govern the way an ontology is modularised. These strategies include: disjoint or overlapping modules, semantics-driven strategies that involve allowing experts to create modules that 'make sense' and structure-driven strategies whereby ontologies are decomposed based on structural properties.

Other research has been performed on exploring semantic notions of modularisation [91, 92]. These studies are logic-based with a main focus on module inseparability, i.e., when two ontologies (the module, and the source ontology) are deemed to be inseparable if they give the same answers to any query. Wang et. al [161] look at the Food and Agriculture Organisation (FAO) information systems as a case study to improve it using modularity. They propose four requirements for the modular ontologies: encapsulation, re-usability, trust, and security. Stuckenchmidt and Klein bring propose three properties for a modular ontology to fulfil: loose coupling, self-containment, and integrity [148]. For loose coupling, the modular system should prevent unwanted interactions among the modules. The modules have to be self-contained to facilitate re-use of individual modules. For integrity, the modules have to have mechanisms to check whether knowledge across the system has changed to preserve the correctness of the reasoning. The existing work on properties for module also include some overlap on evaluation metrics for modules. For instance, properties such as 'loose coupling', and 'encapsulation' mentioned in this section could be used to evaluate a module.

2.5 Use-cases of modules

In this Section, we group together the goals, reasons, and purposes that have been discussed for modularisation as use-cases.

Parent and Spaccapietra [123] define several goals of modularity. These goals include scalability for reasoning and maintenance, complexity management, understandability and reuse. Stuckenchmidt and Klein's work [148] focuses on modularisation with respect to local containment of terminological reasoning. For this, they describe three reasons for using modularity: to have distributed systems, to deal with large ontologies, and to promote efficient reasoning.

2.6 Techniques of the modularisation process

The two main approaches in modularisation are ontology partitioning and module extraction [35]. Ontology partitioning is the process of splitting up axioms into modules such that the union of all modules is equivalent to the original ontology, while ontology module extraction deals with reducing an ontology to a sub-module to cover a specific subject domain. As such, different techniques are used for the approaches. Commonly, for partitioning, structure-based approaches are used [4, 58, 138], while for module extraction there is usually some semantics and human intervention in techniques [114, 131, 159]. In this section, we describe various approaches that may be applied to generate ontology modules.

A common technique in module extraction is the traversal approach, whereby based on an element of the input vocabulary, relations in the ontology are traversed to gather concepts to be included in a module. An example of this is depicted in Fig. 2.1. Tools that are based on the traversal approach are PROMPT [113] and KMi [33]. Seidenberg and Rector propose the segmentation approach to query-based modularity [141]. This approach exploits semantic links between ontology entities to extract relevant segments of an ontology, given an input entity. The segmentation algorithm produces desirable results for the large GALEN ontology; it reduces the ontology by a factor of 20.

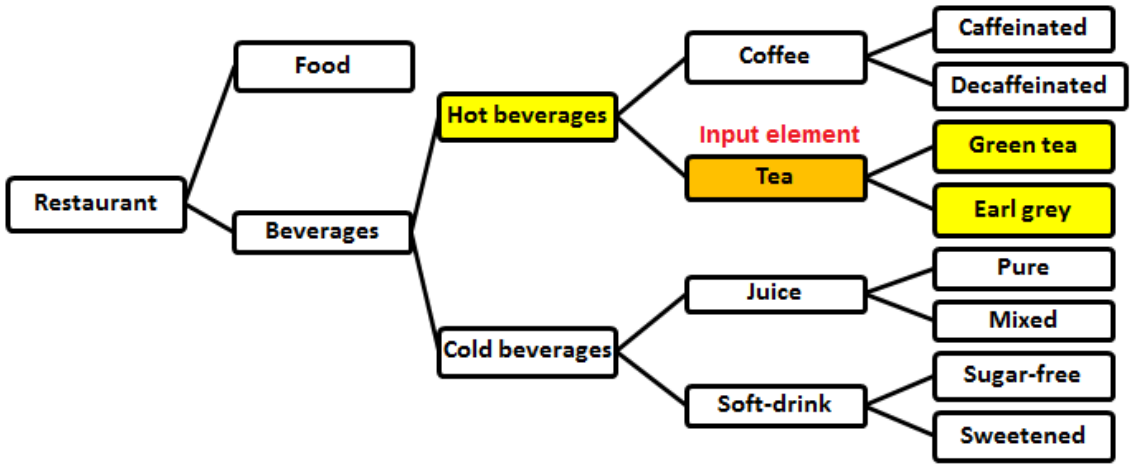


Figure 2.1: The traversal approach in module extraction for a restaurant ontology where the Tea entity is the input element and by traversing through its related entities, the Hot beverages, Green tea and Earl grey entities are included in the module.

All partitioning techniques are based on Graph theory [17]. It has been applied to solving many problems in varying domains. In graphs, communities are clusters of nodes that are fairly independent of each other with weak links between them. Community detection deals with splitting up graphs in order to identify independent components. Most recently, community detection has increasingly been used in social network analysis [26, 122, 126]. In social graphs, vertices represent users, places,

movies, etc., while edges represent the relationships between them, e.g., friend of, likes, visits. Similarly, in ontologies, vertices represent classes while edges represent the relationships between them. As such, there is a strong relationship between the structure of a social graph and an ontology. This is re-enforced by Mika's study in which ontologies are used to extract social networks [104].

Del Vescovo [156] defines the term *atom* as a building block of an ontology or a clump of highly inter-related axioms that cannot be split across two or more ontology modules. The atomic decomposition of an ontology is the set of atoms that the ontology has been split into. Decomposition-based module extraction is a method of partitioning an ontology based on its atomic decomposition. TaxoPart is a partitioning tool with the aim of partitioning large ontologies with the goal of assisting in ontology alignment between ontologies [58]. The approach in TaxoPart is to transform ontologies into sets of blocks of a limited size and thereafter align the blocks from different ontologies.

Partitioning tools for ontologies (PATO) [149] is a tool that uses the structure of the class hierarchy to partition large ontologies automatically. This is performed in three tasks: creating a dependency graph from the ontology file using existing ontology to graph converters, determining the strength of the dependencies between concepts in the ontologies and lastly, determining modules. This is a useful method in that it ensures that the ontology is partitioned into modules with strong internal connections and weaker external connections.

For the SWOOP partitioning tool, [73], ε-connections are used for guiding the partitioning process. The main steps in this approach are a safety check, generating a partitioning graph, and identifying and extracting modules. The partitioning graph's language matches that of an ε-connection. Elsewhere [100], a partition-based reasoning framework is used to deal with large knowledge bases. This framework is composed of graph-based algorithms which assist in automatic partitioning, and message-passing algorithms which are used for reasoning over the partitioned module.

There are a number of logical approaches used in modularisation; these approaches focus on satisfying properties such as the local correctness and completeness of the ontology. These modules are referred to as locality-based modules. Tools based on locality-based modularity such as the OWL module extractor [30], generate modules as follows: given an input signature seed, entities of the ontology that reference the signature seed are preserved in the module. A signature seed is a set of entities used as a basis to extract a module. Another logical approach in module extraction is to reduce it to a reasoning problem based on the rule-based language, datalog [131].

Abstraction, the principle of simplifying complex models by removing some unnecessary details could be applied to ontologies to create simpler modules. Existing approaches include applying the generalisation (isA) techniques, whereby the abstraction function returns the parent entity of a given entity [50, 120]. Similarly, using mereology (the partOf relation) could be applied as a technique where, given an entity, the abstraction function returns the entity that it is part of. Keet [77] proposes three abstraction types; each can be achieved using a list of abstraction functions. There is R-ABS (or generalisation) whereby a more detailed element abstracts into its parent type over a relationship (e.g., a partonomy), F-ABS whereby several entities

and their relations are folded into a single parent type, and D-ABS whereby entities are removed based on their relevance [77]. These types are promising for achieving abstraction but require additional user input since they use general categories from DOLCE's foundational ontology (e.g., Endurant, and perdurant), and to apply this to a range of domain ontologies, the domain entities will have to be aligned according to these categories.

Another approach was designed to simplify large ORM models using a list of rules whereby some roles are deemed less relevant and could be excluded from the model [22]. Not all the rules from Campbell's abstraction method can be applied to ontologies, e.g., unary roles and anchor points are not relevant for ontologies. Furthermore, Keet [76] concluded that Campbell's method is influenced by the model used in the case study; some of the rules become irrelevant when applying them to different models. Keet modified these rules by replacing the rules that were not applicable, with domain-independent abstraction mechanisms which may also be applied to ontologies [76]. All these proposals for abstractions remain theoretical, however, yet their implementation, if adapted towards ontologies, may be used to simplify an ontology such that it assists with the ontology comprehension problem.

The Foundational Model Anatomy (FMA) ontology is one of the largest anatomy ontologies [133]. Difficulty in its reuse has been a motivation for the abstraction of it [105]. The nature of the ontology is that since it defines features of the human body, those that are symmetrical are similar, e.g., left and right, proximal, middle, etc. To create an abstraction of it [105], an ontology design pattern called the selector pattern is applied to it to remove similar parts of the ontology. However, parts are to be manually identified for removal.

OWL profile modules are ontologies represented in a less expressive language than the language that the original ontology is represented in, such that the less expressive language is one of the OWL 2 profiles [93]. Naturally, applying the restrictions to ontologies result in ontologies that are a subset of the original ontology, i.e., a module, by dropping axioms that are outside of a particular language fragment. Protégé v4.3 [110] has a feature for generating modules in some OWL profiles.

To summarise, the existing literature on modularisation focuses on a number of partitioning and modularisation techniques and tools. Most of the tools that exist are partitioning tools, and modularisation tools with techniques to generate locality-based and query-based modules. Techniques have been described for abstraction, but there are no software tools to apply them to ontologies. There is some tool support for generating OWL profile modules using Protégé v4.3 [110] for generating modules in the EL OWL 2 profile. The current techniques in modularisation is shown in Table 2.1.

2.7 Types of modules

Work has been done on characterising modular ontologies in terms of patterns concerning how imports exist in a set of modules [2]. Four types of patterns were found: i) one module importing n modules, ii) n modules importing one module, iii) n mod-

Table 2.1: A summary of the current techniques of modularisation; a tick indicates those techniques that have a tool.

Author	**Technique**	**Tool**
Noy and Musen [113]	Traversal	✓
d'Aquin et. al [33]	Traversal	✓
Hamdi et. al [58]	Partitioning	✓
Stuckenschmidt and Schlicht [149]	Partitioning	✓
Kalyanpur et. al [73]	Partitioning and Locality-based	✓ ✓
Cuenca Grau et. al [30]	Locality-based	✓
Musen [110]	Language simplification	✓
Seidenberg and Rector [141]	Traversal	
Del Vescovo [156]	Partitioning	
MacCartney et. al [100]	Partitioning	
Romero et. al [131]	Locality-based	
Campbell et. al [22]	Abstraction	
Keet [76]	Abstraction	
Mikroyannidi et. al [105]	Abstraction	

ules importing n-1 modules, and iv) pattern mix that combines the previous three patterns. The characterisation is at a starting point; however, it is solely based on three structural criteria (size, cohesion, and coupling) and patterns based on ontology imports [2].

Borgo classifies modules according to their purposes [20] where there are three main types of modules with purposes: 1) modules for a single ontology, those modules which aid with organising and managing domain coverage, 2) modules for several ontologies, basic functionality that when combined lead to better quality ontologies, and 3) modules for everything, which has several different meanings such isolating/developing branches of a taxonomy, collecting categories according to a domain (medicine, engineering,) isolating patterns, separating systems to improve ontology matching, and more.

2.8 Evaluation metrics of modules

A fair amount of work has been conducted on the evaluation metrics of modules. In this section, we first discuss the existing body of work on evaluation metrics and thereafter provide a list of each existing metric.

A study [35] has been conducted whereby d'Aquin et. al review existing tools in terms of modularity criteria. Those criteria for evaluating the generated modules includes logical criteria, e.g., local correctness, structural criteria e.g., size of module, and intra-module distance, software criteria, e.g., encapsulation, and independence,

quality criteria, e.g., module cohesion, and relational criteria, e.g., connectedness, and inter-module distance. The application criteria, for evaluating the modularisation tool is based on criteria such as performance and use of modules. The results of the study [35] show that: 1) no single approach can meet all the criteria, 2) there is no universal way to modularise an ontology, and 3) the modularity technique depends on the scenario for it. In other work, Loebe [98], proposed a number of requirements for logical modules, such as logical correctness and completeness. Loebe also acknowledges that the requirements do not hold for all applications and that specialised methods should be applied for different applications.

Pathak et. al [124] identified main properties that modules need to satisfy, such as size, correctness, completeness, and evaluated these using existing tools. Pathak et. al [124] found that not all the techniques satisfy all the evaluation techniques. For instance, the authors express that there is a risk using graph-based techniques because they cannot satisfy the criteria of localised semantics because the ontology is partitioned into disjoint sets thus the meaning of every concept is not preserved as the original ontology. Tartir et. al [151] propose richness criteria to measure the quality of ontologies. This criteria is based on the amount of relational information in an ontology compared to the number of classes, e.g., the inheritance richness measures the number of subclasses per class and the attribute richness is the number of attributes per class.

Schlicht and Stuckenschmidt created a set of structural criteria for ontology modules [137]. The authors argue that the structural criteria has an effect on efficiency, robustness and maintainability for the application of semantics-based peer-to-peer systems. The structural criteria proposed include connectedness, size, and redundancy of representation. Connectedness of a module describes the interaction of axioms in different modules. Redundancy of representation measures the amount of duplicated axioms in a set of ontology modules. A high value of redundancy entails a large number of shared knowledge on the ontology modules which causes higher maintenance of a set of modules. The authors propose quantitative functions which can be used to measure each criteria value formally. The function to measure the size of a module focuses on the 'appropriate' size of a module where an appropriate module is deemed to have 250 axioms. SWOOP and PATO modularity tools are then evaluated using these structural criteria. Schlicht and Stuckenschmidt [137] found that SWOOP favours modules with a good connectedness, over modules with suitable size values. With PATO, a threshold value could be selected and as it is increased, so is the size suitability of the module, while the connectedness value worsens.

Yao et. al introduce cohesion metrics for ontologies [164]. Cohesion generally refers to the extent to which entities in a module are related. The metrics that they propose for this are: number of root classes, number of leaf classes, and average depth of inheritance tree of all leaf node. These metrics, are not aimed at evaluating the quality of modules, however, but are rather general for all ontologies. In light of this, Oh et. al present new metrics for cohesion to measure the strength of the relations in a module [118]. The work on metrics in [39] focus on semantic dependencies to measure the coupling and cohesion of ontology modules. The coupling and cohesion metrics use the notion of strong and moderate dependencies between entities. However, there

are certain relations in an ontology that are neither strong nor moderate (relations containing the intersections of classes). To measure the coupling of an ontology, researchers propose metrics based on the number of externally defined referenced concepts [119]. This, however, does not take into account external links that different modules share. Oh and Ahn [115] have improved on this to consider the external links between different modules based on whether the link is hierarchical or relational. However, their metric is simply a sum value of the number of each type of links between modules, which does not measure the complete interdependence of a module since it only considers one type of variable in the module. We now summarise a list of evaluation metrics from the literature mentioned in this section together with the definition for each metric.

Correctness: Correctness states that every axiom that exists in the module also exists in the source ontology and that nothing new should be added to the module [98, 31, 35, 124].

Completeness: Completeness is when the meaning of every entity in a module is preserved as in the source ontology [98, 31, 35, 124].

Coupling: A measure of the degree of interdependence of a module [48, 118, 115, 119].

Cohesion: Cohesion refers to the extent to which entities in a module are related to each other [48, 118, 115, 164].

Size: Size is the number of entities in a module (the number of classes, object properties, data properties, and individuals in a module) [34, 35, 118, 137, 124].

Redundancy: Redundancy is the duplication of axioms within a set of ontology modules [137].

Appropriateness: Appropriateness is measured by mapping the size of an ontology module to some appropriateness function value between 0 and 1.

Inter-module distance: The inter-module distance in a set of modules has been described as the number of modules that have to be considered to relate two entities [34, 35].

Intra-module distance: The intra-module distance in a module is the distance between entities in a module [35].

Encapsulation: Encapsulation is a metric that holds when "a module can be easily exchanged for another, or internally modified, without side-effects on the application can be a good indication of the quality of the module" [35].

Independence: Independence evaluates whether a module is self-contained and can be updated and reused separately [35].

Attribute richness: Attribute richness is defined as the average number of attributes per class [151].

Inheritance richness: Inheritance richness is defined as the number of subclasses per class in an ontology [151].

2.9 Discussion

We analysed modules and literature on modules using content analysis as a research method to identify common components at the onset of the Chapter. While content analysis is useful for gathering meaningful information from a large and diverse body of work, it also has an issue. The availability of resources limits the analysis. For modules, this means that if some modules were found in repositories and not in existing literature, this would have been overlooked if the analysis only focussed on ontology modules found in repositories. Similarly, some modules are described in the literature but not available online. To deal with this, we used a variety of resources for our analysis, from both literature and repositories from the web.

The analysis of existing modules revealed that modules can be characterised by a use-case, techniques, type, property, or evaluation metric. Identifying these dimensions, bring us a step closer to achieving the goal of a foundation for modularity aimed at guiding the developer with the modularisation process. The literature presented in this Chapter demonstrated some of our problems presented regarding modularity in Section 1.2. The research about modularisation techniques reveals that about half of the techniques are theory-based only without tool implementations. For the abstraction techniques, the ones that do exist are geared towards conceptual data model or specialised for the FMA ontology and cannot be applied to ontologies in general, nor is there any tool implementation. From the work on evaluating an ontology module, while there are some existing metrics, it is not clear how to apply them to modules to check if a module is of good quality or not.

From this Chapter, it was found that: 1) Current definitions regarding modularity are not sufficient; they are too restrictive, and not exhaustive enough to apply to the various existing modules. 2) Indeed, there do exist dimensions which can be used to classify modules which we identify as the use-case, type, technique, property, and evaluation metric. 3) The current literature on modularity reveals that tool-based support for abstraction techniques of modularity is lacking and that there are some evaluation metrics for modularity, but there is no way to check if a module is of good quality.

2.10 Conclusion

This chapter is a literature review of the research that has been conducted on modularity to date. Various dimensions for modularity were identified from the body of work: use-case, technique, type, property, and evaluation metric. Identifying these dimensions is a first step toward creating a foundation to guide the modularisation process. In addition, we have an idea about what is lacking with regards to modularity. Existing evaluation metrics are not sufficient to apply to different modules, and techniques for modularisation are lacking.

In the next chapter, we use the dimensions for modularity and the body of work discussed here to define and populate the modularity dimensions toward creating a framework for modularisation.

Chapter 3

Dimensions of modularisation

The increasing usage of ontologies in heterogeneous domains has led to the development of large and complex ontologies. Such ontologies cause great difficulties for humans to understand properly and apply them, and for computers to process and reason with them. In order to assist with this, a large amount of research has been conducted on the notion of ontology modules. A number of techniques, tools and metrics for ontology modularity have been published in recent years. However, while there is a plethora of work on ontology modularity, there is no structure or foundation for it. To create a foundation for modularity, the dimensions of modularity need to be identified and defined. There are no well-defined ideas or criteria concerning modularity, causing great difficulty when attempting to create and evaluate an ontology module. This demands for the analysis and guidance of such aspects of modularity. The aim of this Chapter is to define the dimensions of modularity towards the creation of an empirically-based framework. This will serve as a starting point and guide on modularising ontologies since there are a number of varying aspects and approaches to modularity that depend on the use-case at hand.

In order to uncover the essence of ontology modularity, we identify the dimensions associated with it. There are five essential dimensions for ontology modularity which affect each other and can be used as a guide to creating an ontology module for a particular purpose. We begin the Chapter in Section 3.1 where we provide a formal definition for modularity. Thereafter in the following sections, Section 3.2-3.5 we discuss each dimension, with their sub-dimension and some examples to demonstrate their usage. In Section 3.6, we perform an experimental evaluation aimed at classifying a set of ontology modules using the dimensions. We analyse each dimension against the set of modules which revealed values for use-cases, techniques, types, and properties for each module. Next, in Section 3.7, we present the framework for modularity which contains dependencies between the dimensions, aimed at guiding the modularisation process. We evaluate the framework with ontology and conceptual data model case-studies in Sections 3.8.1 and 3.8.2. We follow with a discussion in Section 3.9 and conclude the Chapter in Section 3.10.

3.1 Definition

Following the inconsistencies and gaps from the existing definitions for modularity described in Chapter 2, we now create our own definition for a module. This work in this section has been published in [85]. Our definition for modularity resolves all the shortcomings of definitions 1-5 from Section 2.1 as follows:

1. It does not restrict modules to those that exist in a set and together compose a whole.

2. It allows that modules need not have a source ontology O as in the case where ontologies are created in a modular fashion from the outset.

3. It does not restrict modules to those that capture all the knowledge of an ontology over a given signature (locality modules).

It also introduces modularity dimensions such as use-cases, techniques, properties, and evaluation criteria which have not been previously defined but are important because they guide the modularisation process. The inclusion of these dimensions in the definition contributes to an exhaustive, generic, broad definition for modules. The dimensions introduced in the definition (U, T, P, MT, and EM) are already fixed and will be presented in following sections of the Chapter.

Definition 6 *(Module) A* Module M *is a subset of a source ontology O, $M \subset O$, or M is an ontology existing in a set such that, when combined, make up a larger ontology. M is created for some use-case $u \in U$, number of $u \geq 1$, and is of a particular type $t \in T$, number of $t = 1$. t is classified by a set of distinguishing properties $\{p_1, ..., p_k\} \in P$, number of $p \geq 1$, and is created by using a specific modularisation technique $mt \in MT$, number of $mt = 1$, and has a set of corresponding evaluation metrics $\{em_1, ..., em_k\} \in EM$, number of $em \geq 1$, which is used to assess the quality of M.*

In the next sections, we describe each module dimension from the definition (U, T, P, MT, and EM) and populate them by identifying respective sub-dimensions for them.

3.2 Use-cases

The work in this section until Section 3.5 has been published in [85] and [86]. The type of module that is created greatly depends on the use-case for which modularity was considered; because different purposes can lead to different modules. There are many use-cases for creating and using ontology modules which we identify and discuss in this section. The different types of use-cases, also referred to as purposes, goals, benefits or rationale for modularity, has been mentioned in numerous works [34, 35, 124, 139, 155], and we list and describe all of the known use-cases for modularisation with the aim for the list of use-cases to be exhaustive.

U1: Maintenance Ontologies are constantly evolving. As such, there is a need for constant updates and maintenance. One person can not easily maintain enormous monolithic ontologies. It is a task that is prone to error and omission [23]. Ontology developers often face difficulty in building and populating an ontology with many entities and with sense-making, searching and exploration of an ontology. Such problems are related to the loss of contextual awareness when traversing an ontology [160]. Dividing an ontology into modules can assist with facilitating the maintenance of large and highly complex ontologies. A set of modules could be easily managed, as it is divided into smaller subsets since only specific modules need to be referenced for maintenance rather than the entire ontology. Not all the modules in a system need to be modified if there is a change in the ontology; the knowledge update is localised within the relevant module(s). Maintenance also promotes collaborative efforts, which is discussed as a use-case in a subsequent section.

U2: Automated reasoning Ontology reasoners do not perform well when reasoning over complex and large ontologies of thousands of concepts [160]. Performance decreases as the ontology size and number of axioms and rules increases. Large ontologies cause reasoners to use too much of time and memory and causes the ontology development environment where the reasoner is implemented in to malfunction. Consequently, reasoners will perform better with regard to efficiency if inference has to be performed over less knowledge. In some cases, one will only be required to reason over modules that have been evolved since the last reasoning task. The data mining optimisation ontology (DMOP), for instance, has a classification time of approximately 10 minutes for reasoning [78]. It is worthwhile to investigate whether modularisation could result in improved reasoning time.

U3: Validation It is difficult for a single expert to validate and understand the model as a whole in a large ontology [160, 147, 38]. In order to solve this, the ontology could be modularised to a size that is comprehensible by a human. Naturally, smaller modules are easier for a human to understand which will aid in validation. Identifying errors such as inconsistency and redundancy, and guaranteeing that the ontology meets all the functional requirements in a large, monolithic ontology is a difficult process.

U4: Processing Tools for processing ontologies for different applications such as those used for engineering, mediation, metrics, and editing do not perform well with large ontologies [5, 15, 125]. For instance, with the NCI cancer ontology by [54], the BioPortal visualisation tool [163] took several minutes to load the ontology, and using the OWL metrics tool[1] to compute its metrics took 12 minutes to process before it returned an ontology parsing error, using a machine with an Intel Core 2 Duo Processor with 4GB of RAM. These types of scalability issues are a challenge for developers when using these large ontologies. As demonstrated, the complexity of processing for large ontologies is known to be critical. Since smaller ontologies take

[1] `http://mowl-power.cs.man.ac.uk:8080/metrics/lastaccessed:20June2017.`

a shorter time to open, load, and use with tools, having smaller interrelated modules instead of large and complex ontologies could possibly improve the performance of the processing tools.

U5: Comprehension Gibson et al. define comprehension as follows:"We outline ontology comprehension as the interaction between human agents and the knowledge expressed in an ontology" [52]. It causes a cognitive overload for humans understand and use ontologies with thousands of terms. Keet proposes the use of abstraction by removing some knowledge from an ontology to assist with ontology comprehension [77]. Since ontologies are sometimes designed and created by domain experts without expertise in logic, they rely on visual ontology engineering tools for creating ontologies. Visual ontology engineering tools aid with ontology development but have drawbacks with large ontologies, being that entities cannot be displayed for easy traversal, thereby making it difficult to understand the complete ontology. Some approaches propose model exploration techniques [12, 16] to assist with ontology comprehension whereby the concepts with corresponding relations of an ontology are visually generated in order to understand them. For large ontologies, model exploration techniques could be problematic because of the challenge with ontology processing explained in the previous section.

Comprehension differs from validation for modules as follows. For comprehension, the module is created for cognitive reasons whereas for validation, the creation occurs for suitability. Furthermore, for validation, all components of the ontology need to be considered. However, for comprehension, or for human understanding purposes, simpler views omitting unnecessary components can be considered.

U6: Collaborative efforts Collaborative efforts in ontology development allow a team of experts to combine their knowledge towards the common goal of creating an ontology. Using ontology modules is an approach that enables the division of work tasks. In order to avoid conflict between different versions of the ontology by different developers, it is sensible to divide the ontology to different modules and allow specific people to create and modify specific modules without altering the entire system. This also promotes the parallelisation of ontology development.For instance, there is the set of myExperiment ontology modules [111] which promotes collaboration among scientists for publishing and sharing workflows. On a grander scale, there is the OBO Foundry ontologies [143] aimed at providing interoperable ontologies that are scientifically accurate.

U7: Reuse At times, developers only require a subset of terms from an ontology and not the entire system to reuse in another ontology. For instance, in the Subcellular Anatomy Ontology (SAO) ontology [96], there only exist 3D entities. As such, the BFO-Continuant ontology of the ROMULUS repository [80] can be used rather than the entire BFO ontology. Thus modular ontologies provide infrastructure for ontology development whereby ontology modules can be easily extracted and reused.

The modular components can then be easily adapted for the application at hand.

In this section, we have defined and described seven different purposes or use-cases for ontology modularity. The use-cases form the basis for an approach for creating good and usable modules and will have an effect on the other dimensions of modularity. We can now answer research question 1(c) from Section 1.4 "What are the different purposes behind module creation?" The different purposes behind module creation are presented in this section (U1 - U7).

3.3 Types

We propose that ontology modules can be classified into different types, based on the nature of the module. Module types have been discussed by Borgo [20]. Borgo classifies modules as: 1) modules for a single ontology, those modules which aid with organising and managing domain coverage, 2) modules for several ontologies, basic functionality that when combined lead to better quality ontologies, and 3) modules for everything, which has several different meanings such isolating/developing branches of a taxonomy, collecting categories according to a domain (medicine, engineering, isolating patterns, and more [20]. From existing ontology modules and Borgo's research [20], we have identified and refined modularity type dimensions as subtypes which are classified into four main types of modules: structural, functional, abstraction, and expressiveness modules.

To identify the subtypes of modules, we perform the following tasks. We gather a set of ontology modules and analyse their files. We also review existing literature works about modularisation. Following the analysis of existing modules in Section 2.2 and the review of literature in Chapter 2, we describe each module using keywords such as 'taxonomy', 'functional', etc. Thereafter we categorise the modules based on their keywords into Borgo's existing types and create a description for each of Borgo's existing types. As some of the modules from the literature and the set of ontology modules could not be categorised using Borgo's types, the keywords describing them are then used to create new module subtypes. We also use the keywords to provide a description for each subtype. We then list all the new and existing subtypes and group them together into types.

Besides using and refining Borgo's modularity dimensions as modules subtypes, we have identified new subtypes: locality, privacy, axiom abstraction, type abstraction, high-level abstraction, weighted abstraction, expressiveness sub-language, and expressiveness feature modules. We will describe these module types and subtypes in the remainder of this section. This work was published in [88].

3.3.1 Functional modules

Functional modules are those in which a large ontology is modularised by dividing it into functional components or subject domains. This assists with selective re-use of an ontology. There are subtypes for functional modules which are described here.

T1: Ontology design patterns An ontology is to be modularised by identifying a part of the ontology that can be reused as a best practice for recurring ontology issues; hence, one can isolate a new ontology design pattern (ODP) [44] for general reuse. For instance, the Set ODP [27] can be reused for any domain instead of starting from scratch, and there are several content ODPs. However, not all ODPs are considered modules; e.g., the Lexico-Syntactic ODPs are not. Ontology design patterns differ from other modules in that they are specifically created for overcoming common challenges within ontologies, however, they are still modules according to our definition.

T2: Subject domain modules A large domain must be subdivided according to the subdomains present in the ontology. For instance, the set of architectural design modules [68] such as Spatial ontology, Building construction, among others.

T3: Isolation branch modules A subset of entities from an ontology is extracted. However, entities with weak dependencies to the signature are not to be included in the module. For instance, to isolate the 'endurant' branch of DOLCE [102], the dolce:physical-endurant entity is a direct subclass of dolce:endurant to include in the module, but not the dolce:perdurant because it is linked to dolce:endurant in terms of participation.

T4: Locality modules A locality module is a subset of the axioms in an ontology and is extracted for a set of entities. However, unlike the case of T3: Isolation branch modules, for locality modules, all entities that are dependent on the subset is included in the module. For the previous example, this means that the dolce:perdurant entity is to be included in the module, along with others that are related to dolce:perdurant. T3, and T4 modules are both subset modules but the distinction is that for T4 all entities that are dependent on the subset are included in the module while in T3, only entities that are within the respective branch of a subset are included.

T5: Privacy modules A privacy module is one in which some sensitive information is removed so that privacy is preserved for a particular application.

3.3.2 Structural modules

Structural modules are those ontologies that have been partitioned into modules based on structure and hierarchy. The focus of the modularity is on the syntax of the ontology. In this case, each module is to be separate from one another. There are subtypes for structural modules which are described here.

T6: Domain coverage modules There is an ontology covering a domain, and developers wish to facilitate ontology maintenance by dividing ontologies structurally, without considering the semantics of the ontology. Hence, the modules are divided by their graphical structure and placement of entities in the taxonomy such that

similar size modules could be generated. The set of modules then cover the entire domain. If the ontology modules are to be maintained collaboratively by a team with a specific number of ontology developers, the number of modules to be created could be specified, and the structure of the ontology is exploited to create modules. For instance, the Foundational Model of Anatomy Ontology [133] contains over 100 000 entities describing the exhaustive biomedical informatics domain. This could be modularised structurally for ease of use.

T7: Ontology matching modules An ontology must be modularised to assist with ontology matching by partitioning it into disjoint modules so that there is no repetition of entities when matching occurs. Most matching techniques implement structural or string-matching techniques; hence the semantics of the original ontology need not necessarily be preserved. For instance, the Common Anatomy Reference Ontology (CARO) [57] aims at aligning existing anatomy ontologies. To assist with aligning it to domain ontologies, CARO could be partitioned into smaller modules.

T8: Optimal reasoning modules A large ontology is to be divided into smaller modules to assist with overall reasoning over the ontology and to ensure that reasoners do not malfunction. The DMOP ontology contains over 758 entities and over 4000 axioms and takes almost 10 minutes for the reasoner to perform classification; it would be less time-consuming to maintain and extend if localised reasoning in a module would be possible. This differs from creating modules that are of a less-expressive ontology language which is discussed later.

3.3.3 Abstraction modules

For abstraction modules, some detail must be hidden from the ontology to create a simpler view of the ontology making it 'lightweight' with less detail. There are subtypes for abstraction modules which are described here.

T9: Axiom abstraction modules This is a module having fewer axioms with object properties relating classes, thereby decreasing the horizontal structure of the ontology. For instance, to create taxonomies for classification purposes from ontologies.

T10: Entity type modules This is a module where a certain type of entity is removed from the ontology, e.g., data properties or object properties. For instance, removing the application-specific instance data (individuals) from an ontology to reuse in another application.

T11: High-level abstraction modules This is a module where only higher-level classes of the ontology are required, thereby decreasing the vertical structure of the ontology. For instance, the DMOP-branch-Toplevel module [78] containing only the high-level entities of DMOP.

T12: Weighted modules The developer decides on entities in the ontology that are more important than others. For instance, using abstraction rules to assign a higher weighting to entities that are deemed more important than others [22, 76].

3.3.4 Expressiveness modules

For expressiveness modules, an ontology is modularised according to a specific ontology sub-language by removing some of its expressive power, where it is not required for certain applications. There are subtypes for expressiveness modules which are described here.

T13: Expressiveness sub-language modules These modules contain limited language features that are captured in a sub-language of a core ontology language.For instance, the OpenGalen [128] module in OWL 2 EL [109] was created to test the lightweight ELK [74] reasoner for EL ontologies.

T14: Expressiveness feature modules These modules contain limited language features that are not necessarily defined by any sub language and consider modelling alternatives to preserve the meaning of the ontology. For instance, the DMOP-WithoutInverseRoles modules was created by removing the OWL InverseObject Properties language feature and replaced with the OWL ObjectInverseOf axiom [78].

By refining Borgo's modularity dimensions and analysing existing modules, in this section, we have identified four main module types: structural, functional, abstraction, and expressiveness. We have classified and described the subtypes for each of these main module types. We can now answer research question 1(a) from Section 1.4 "What are the different types of modules that exist?" The different types of modules that exist are presented in this section (T1 - T14).

3.4 Properties

Modules have properties that are associated with them. In this section, we identify and describe the properties that modules exhibit. Properties exist in isolation in a single module, and also in sets of related modules. In order to identify properties that modules exhibit, our approach is as follows. We gather a set of ontology modules. We review existing works pertaining to ontology modularisation and inspect the ontology module files to draw out the properties which modules exhibit. We define each property with use of the existing literature where applicable. Thereafter, we separate these properties into those that are found in a single module, and those that are found in a set of related modules.

3.4.1 Properties of a module

In this section, we describe properties that a module exhibits by itself.

P1: Seed signature A seed signature occurs when the user specifies some input entity to base the resulting module on [30, 156, 157]. All entities related in some way to the entity chosen as seed signature are included in the module. For instance, for modularising the DOLCE ontology for a module with only wholly-present objects, the 'endurant' entity is selected as a seed signature [80].

P2: Information removal Information removal is when parts of the ontology are selected to be removed from the ontology, resulting in a module without all the detail of the original ontology. For information removal, an input entity need not be selected as a basis for modularisation as in the case for the seed signature characterisation above. For instance, the NCS ontology on Bantu noun classes [24] reuses only part of the GOLD ontology, as it has no need for, among others, phonetic properties.

P3: Abstraction Abstraction is the property of hiding undesirable information from an ontology at different levels [53, 76]. Abstraction is used as a principal to give the user a simplified view of the ontology for comprehension purposes. For modules with this property, there exists information with more or less detail in a set of modules. However, the source ontology with all the original information still exists in the system as a related module.

> **P3.1: Breadth abstraction** The type of abstraction that occurs in an ontology where some relational properties of entities in the module are removed in order to provide a simpler view of the structure of the ontology, therefore the 'breadth' of the ontology is reduced.

> **P3.2: Depth abstraction** The type of abstraction that occurs in an ontology where high-level classes from the original ontology exist and lower-level classes are removed; therefore the 'depth' of the ontology is reduced.

P4: Refinement Refinement occurs in ontology modules where some new axioms are added to the module, to assist with inter-module links, or when computationally-expensive ontology language features are modified resulting in new axioms. For instance, to reduce reasoning time for the DMOP ontology, the InverseObjectProperties axiom was removed and replaced with the OWL ObjectInverseOf axiom [78].

P5: Stand-alone This describes a module that has no external links or imports with other ontologies and can exist on its own. It is self-contained and can be modified without having dependencies on other modules. For instance, the BioTopLite module [140], a top-domain level ontology for the life sciences domain, does not contain any inter-module relations with other ontologies nor does it have any import statements linking other ontologies to it.

P6: Source ontology A source ontology is the original ontology which has been modularised in some way resulting in the module. For instance, the DMOP-profile-EL module has the source ontology DMOP [78].

P7: Proper subset This describes a module that contains a subset of entities that are contained in another source ontology. The module has fewer entities than the source ontology. For instance, the FMA_subset ontology module omits all relationships other than is_a, part_of, and has_part and thus has fewer entities than the original FMA ontology [133].

P8: Imports This describes a module that contains other ontology components, by using the owl:import statement declared for importing another ontology. For instance, the Spatial Ontology module [68] from the set of architectural design modules that imports the DOLCE ontology [102].

3.4.2 Properties of a set of related modules

In this section, we describe the properties that a set of modules exhibit altogether, and in relation to one another.

P9: Overlapping Overlapping in modules refers to cases where entities in an ontology system can be found in more than one module of the system [123]. These modules partially cover the same concepts. In overlapping modules, entities in different modules may have dependencies on one another. Thus changes to one module in a system affect some other modules. This is shown in Figure 3.1.

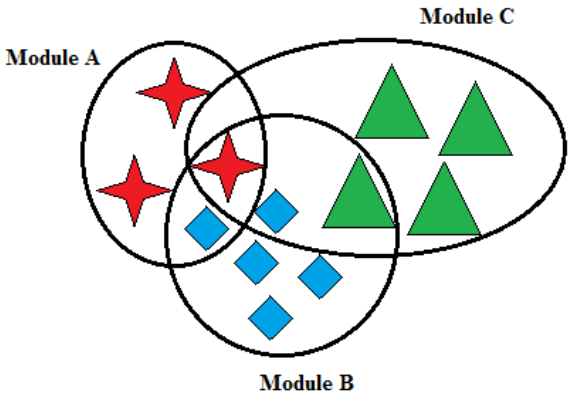


Figure 3.1: Overlapping modules in an ontology system. Modules A, B, and C are three ontology modules that are overlapping. Module A contains knowledge from Module B, Module B contains knowledge from Module A and C, and Module C contains knowledge from Module A and B.

P10: Mutual exclusion Mutual exclusion (or disjointness) in modules refer to cases where entities in an ontology system are not found in more than one module of the system; they have no entities in common [123]. This is shown in Figure 3.2. The advantage of this type of module is that the modules in the system can be managed easier as there is less chance of conflicts among modules since the entities are not common. For instance, if there were entities that were common among modules, and a user altered the axioms defining the entity in one of the modules, the entity definition would be different in the other modules. It is difficult to create mutually exclusive modules for most ontologies because entities in ontologies have referencing axioms to each other therefore there is great difficulty ensuring that entities are not common among modules.

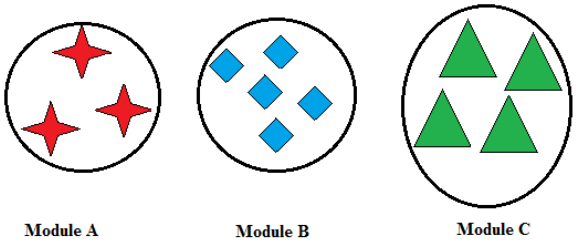


Figure 3.2: Mutually exclusive modules in an ontology system. Modules A, B, and C are three ontologies that are mutually exclusive. Module A, contains no knowledge of modules B, and C; Module B contains no knowledge of modules A, and C; and Module C contains no knowledge of modules A, and B.

P11: Union Equivalence Union equivalence occurs when the union of a set of modules is semantically equivalent to the original ontology. This is shown in Figure 3.3.

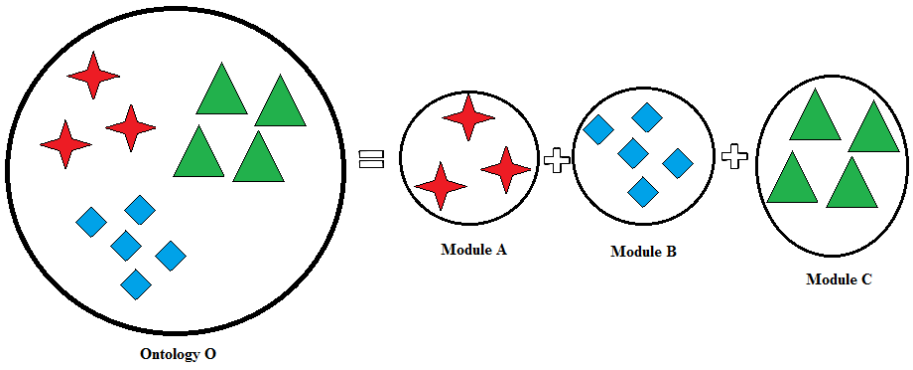


Figure 3.3: Union equivalence occurs for modules A, B, and C; the union of them is equivalent to ontology O.

P12: Partitioning Partitioning occurs in large ontology whereby it is structurally divided into a set of independent modules, thereby allowing concurrent reuse in distributed systems [35]. Independence is meant that the modules cover sufficiently different knowledge of the domain; e.g., representing the anatomical knowledge about the limbs and eyes of an animal. Theoretically, a module having the partitioning property should also have the P10 (mutual exclusion) and P11 (union equivalence) properties but this might not always be the case in practice. This will be demonstrated later in a classification experiment in Appendix A, specifically in Table A.1.

P13: Inter-module interaction This describes modules that have links to other modules to relate entities in a similar way to their existence in the original ontology to ensure that the knowledge is preserved. Inter-module interaction among modules exist if there are either bridge ontologies in the set to link together modules, or linking languages are explicitly used within the modules. For instance, in the EDAM bioinformatics ontology [70], the object property is_format_of has as domain the class Format and range Data. When it was partitioned with SWOOP, the Data class was present in the first partition while Format and is_format_of were present in the third partition, having used ε-connections to create interaction among these entities that existed in different modules.

P14: Pre-assigned number of modules This occurs when the number of modules to be created or generated in a system is known prior to development. For instance, the modularisation tool requires one to state the number of to-be-generated modules upfront, or an ontology is to be divided into a number of modules based on the developers that will collaboratively create and maintain the ontology. This property differs in module sets and can be used to annotate a set of ontology modules.

The properties that we have identified in this section are used to characterise and distinguish modules. We can now answer research question 1(b) from Section 1.4 "What are the properties with which we can characterise each module type?" The different properties with which we can characterise modules are presented in this section (P1 - P14).

3.5 Techniques

In the previous section, we introduced module properties. Here we identify possible techniques from existing approaches that may be used to create modules and classify them into categories. Such techniques are not only restricted to those from the ontology field; there are also approaches from the graph and network theory fields.

3.5.1 Graph theory approaches

Graph theory approaches are those that have been designed to be applied to the general problem of community detection. In graphs, communities are clusters of nodes that are fairly independent of each other with weak links between them. Applying

graph theory techniques to ontologies deals with modularising them according to structure in order to identify modules together with the inter-module links. The advantage of such an approach would be that if there is already a set of modules, algorithms can be applied to improve the modules until the best set of modules has been generated. The disadvantage of such an approach is that the final ontology strongly depends on the initial partitioning and if there is no information about how the initial partitioning ought to be performed, the method performs poorly. The various graph theory approaches are discussed in the remainder of the section.

MT1: Graph partitioning Graph partitioning is when a large graph is divided into partitions with the condition that vertices are not shared across different partitions, and the number of partitions is known. In terms of ontologies, graph partitioning algorithms would be most useful in cases where structural division of the ontology modules is a driving force.

Several graph partitioning algorithms have been proposed for ontology modularisation [3, 4, 73, 139]. In the PATO partitioning tool [139], graph partitioning is performed by using maximal line islands [10] to compute partitions in the graph. A maximal line island checks that for a set of nodes, the strength of the connection between the nodes inside the set is higher than the strength of any connection to nodes outside the set [10]. Unlike traditional graph partitioning, in PATO, the number of partitions to be created is unknown prior to the modularisation.

MT2: Modularity maximisation Modularity maximisation methods aim at optimising the connection between nodes in graphs. To perform this technique, the modularity function Q measures the concentration of edges within modules compared with the random distribution of links between all nodes regardless of modules. In terms of ontologies, this means that regardless of the location of the concepts in the hierarchy, modules will be created based on concepts that have strong axiomatic relations with others.

3.5.2 Statistical approaches

Statistical approaches emphasise on using statistical equations to create ontology modules. In order to do this, the entities in the ontology are converted to data. Thereafter, statistical methods and functions are applied onto the data with the aim of creating modules. Thus, the ontology is viewed as a data set in order to modularise.

MT3: Hierarchical clustering Hierarchical clustering [130] is used to group together data, when little is known about it, such as the number of partitions it should be split to. It is a method aimed at building a hierarchy of clusters, either by an agglomerative or divisive strategy. An agglomerative strategy is one in which each data point is placed in separate clusters, and clusters are merged based on a given distance function between data points in clusters. A divisive strategy is one in which the data is divided recursively as one moves down the hierarchy.

Hierarchical clustering is a good approach to use for data with a hierarchical structure. A hierarchical structure is one in which every entity in a system, except the top-level entity, is a subordinate to at least one entity. Hierarchical clustering is a good approach to use for data with a hierarchical structure [42]. Ontologies have a hierarchical structure; thus this approach would be ideal for modularising ontologies. Furthermore, since hierarchical partitioning does not require initial knowledge about the structure and number of partitions [42], it would serve as an ideal automatic approach for ontology modularity.

To date, there has been one application of using hierarchical clustering to perform ontology modularity where Garcia et. al [47] found that the two hierarchical clustering algorithms obtained similar results when compared to other graph theory approaches. However, Garcia et. al [47] also concluded that semantically other approaches worked better, at least for the case of modularising a version of the pizza ontology, because their modules grouped together vegetarian, non-vegetarian, and general pizza entities while the modules of the hierarchical approach did not.

3.5.3 Semantic approaches

In the previous section, the structure of the ontology was the driving force for modularisation. In this section, the entities and axioms of the ontology are used for the modularity approach. The common aspect in each of these approaches is that it is user driven, i.e., a user provides some initial information about entities to drive the modularisation process.

MT4: Locality modularity Locality is used to generate modules based on a given signature with the condition that conservation extension holds for the given module. Conservation extension is the notion that every axiom's meaning from the original ontology is preserved in the module. Conservation extension is greatly influenced by the atomic structure of the ontology. In an ontology, an atom is defined as: "a maximal disjoint subset of an ontology such that their axioms either appear always together in modules, or none of them does" [156]. For instance, if one were to generate a locality module of endurant entities (an object that is wholly present at all times) from the DOLCE ontology, a number of perdurant entities (entities unfolding in time, e.g., processes) would also be contained in the module, because there exists an axiom $\mathsf{endurant} \sqsubseteq \exists\ \mathsf{participant\text{-}in.perdurant}$, in the DOLCE ontology. Therefore the module would not be restricted only to endurant entities because the conservation extension of the original ontology is guaranteed. Clearly, the atoms in the DOLCE ontology are large and therefore not suitable for locality approaches, as shown by [80]. On the other hand, biological ontologies from the BioPortal repository [163] have been shown to modularise well using locality methods [157], thanks to them having on average just two axioms per atom.

MT5: Query-based modularity Query-based approaches require that the user initially creates a query with certain conditions in a query language such as SPARQL

and a module is automatically created based on that query. Noy and Musen [114] use queries to allows the user to create a view of an ontology by selecting an input entity and traverse through the ontology, to select other relevant entities to be included in the module until a particular depth is reached. Similarly, given an input entity, the KMI tool [33] recursively inspects the ontology to include the other relevant elements found in the definition of the entity. This type of modularity depends heavily on user input as the user decides, at every step, which entities to be included in the ontology.

The segmentation approach to query-based modularity exploits semantic links between ontology entities to extract relevant segments of an ontology, based on an input entity [141]. It works as follows, given an input entity, first traverse upward through the hierarchy, gathering all its superclasses and ancestor classes to include in the module. Thereafter, travel downwards from the input entity and gather all its subclasses and descendant classes to include in the module. It does not include sibling classes in the module. An evaluation of this segmentation approach reveals that the large GALEN ontology of medical terminology was reduced by a factor of 20 [141].

MT6: Semantic-based abstraction Abstraction, the principle of simplifying complex models by removing some unnecessary details, is applied to ontologies to create simpler modules. Semantic-based abstraction is an approach whereby the semantics of the model is analysed using a set of pre-defined rules to determine key entities, where the key entities are deemed more important than others [22, 76]. For instance, for the Blood and Bacteriocins ORM models [76], mandatory roles are weighted with 10 points while single-role set constraints are weighted with 5 points. Similarly, this could be applied to ontologies by designing a set of weighted rules to guide the ontological abstraction process.

MT7: *A priori* modularity An *a priori* modularity method [152] is one in which the modular structure of the domain is decided, and the modules are created at the onset.

MT8: Manual modularity For manual modularity, the ontology developer decides which entities and axioms should be removed from an ontology, and manually creates a 'custom' module based on this. Unlike the *a priori* modularity method, here the modules are not created at the onset of development, but created later on, based on some existing ontology. For instance, for the DMOP-WithoutInverseProperties module, some language features of the ontology were manually removed to improve the reasoning [78].

MT9: Language simplification Language simplification techniques are those that focus on simplifying the ontology language of an ontology by removing some language features present in the ontology. This results in a simplified module of an ontology with limited expressivity, e.g., a module where the cardinality constraints have been removed.

In Table 3.1, the techniques that are implemented by existing modularisation tools are displayed. The existing tools implement graph partitioning, query-based methods, semantic-based abstraction, and locality-based methods; other techniques are lacking in tools. We can now answer research question 1(d) from Section 1.4 "Which techniques have been proposed to perform different types of modularisation?" The different techniques that are proposed to perform modularisation are presented in this section (MT1 - MT9).

Table 3.1: The modularisation techniques implemented by each tool.

Modularity Tool	**Modularisation Technique**	**Created**	**Updated**
SWOOP [73]	MT1: Graph partitioning	2004	2007
TaxoPart [58]	MT1: Graph partitioning	2009	unknown
OWL module extractor [30]	MT4: Locality-based	2008	2011
Protégé copy/move/ delete axioms [110]	MT4: Locality-based MT9: Language simplification	2008	2016
PATO [110]	MT1: Graph partitioning	2008	unknown
PROMPT [113]	MT5: Query-based	2004	2006

The dimensions in Section 3.2- 3.5 now need to be linked to ontology modules towards creating a framework for ontology modularity. To do this, we perform an experimental evaluation in the next section by classifying a set of modules with the dimensions.

3.6 Classifying modules: An experimental evaluation

There is a lack of a foundational theory for ontology modularity; e.g., it is unclear which evaluation metrics are to be considered for different module types and what type of modules different techniques produce. In [35], it was found that the evaluation of a modularisation depends on an application's requirements, that there is no universal modularisation, and that a formal well-defined framework for modularity is lacking. This opens up a number of issues and questions; e.g., difficulty in selecting the appropriate modularity technique, insufficient modularity tools for applications, and it is unclear which one should be applied for which scenario. For instance, we tried to modularise the Data Mining OPtimization (DMOP) ontology [75] with several modularisation tools, but all modules were too large to use [78], and extracting content on object properties from DOLCE with the 'copy' feature, their asserted characteristics such as transitivity were not extracted [79]. Also, existing techniques are not sufficient in creating compact modules [35, 80]. Evaluation of modularisation techniques reveals that some tools fail to partition large ontologies because they focus on preserving the logical properties of the modules while others lose some of the

relational properties of the ontologies and that most tools generate views instead of module file outputs [116, 117].

In praxis, it also remains largely unclear how to manage ontology modules once they are generated and start leading their own life, being merged with or imported into another ontology. For instance, one may have slimmed a highly axiomatised module into an OWL 2 EL fragment of it or extracted only two main branches of the class hierarchy and their relations: in the former case, it would not matter for one's project if the original ontology was augmented with axioms whose expressiveness is beyond OWL 2 EL, in the latter case, one may have to re-generate the module to reflect the changes made to the original one. Currently, there is no way of knowing this automatically and modules are typically not even annotated with such type of information (unless they were created for certain experiments), let alone annotated in a structured way across modules.

These issues raise a plethora of questions not only from an engineering viewpoint to create tools, but also, still, from a conceptual and ontology engineering viewpoint. We now raise a few 'mini questions' regarding modularity, in addition to the main research questions of the thesis:

1. How do module types differ with respect to certain use-cases?

2. Which techniques can we use to create modules of a certain type?

3. Which techniques result in modules with certain annotation features?

The purpose of this experimental evaluation is to classify a set of ontology modules with the dimensions for modularity from this Chapter: use-cases, types, techniques, and properties. This will lead to the development of a framework for modularity. The classification in the section was published in [85]. We apply the grounded theory research method for this classification. This research method is used for "developing theory that is grounded in data systematically gathered and analyzed" [146]. Grounded theory allows us to construct theories or finding through analysing real-life data. Typically, a grounded theory study begins with some questions and data We have three research questions identified for this study, and the data we will use is discussed in the next section. We have selected this approach for a number of reasons: 1) It allows us to keep an open mind; since grounded theory is not linked to any pre-existing data or hypothesis, it has the potential to generate new and innovative theories, and 2) Validity: it will accurately represent real-world settings since we will be using real-life data.

3.6.1 Materials and methods

The method for the experiment is as follows:

1. We collect ontology modules from ontology repositories and existing literature on modules.

2. Classify each ontology module according to its use-cases, techniques, properties, and types. This classification is performed manually for each module by reading through the documentation or published works of an ontology module and looking at the annotation of the ontology module file. We also inspect the ontology module file to determine its properties and type.

3. Conduct a statistical analysis to determine the frequency of dimensions occurring in each module.

The materials used for the experiment were as follows: Protégé v4.3 [110], SWOOP v2.3 [73], OWL Module Extractor [30], TaxoPart [58], and a set of ontology modules. The sample size was 189 ontology modules of varying domains, such as architectural, data mining, biological, chemical, linguistic, among others.

Of these 189 modules, 146 belonged to 11 sets of inter-related modules. A set of inter-related modules is when a large subject domain is represented by a set of modules rather than a large ontology. For instance, one of the 11 sets is the 10 modules of the myExperiment [111] ontology. The full list of ontologies used for this experiment is shown in the classification table in Appendix A.

Our tests were obtained on a 3.00 GHz Intel Core 2 Duo PC with 4 GB of memory running Windows 7 Enterprise. All the test files used for this evaluation can be downloaded from `www.thezfiles.co.za/Modules/testfiles.zip`.

3.6.2 Results and Discussion

The first step for module classification was to determine the use-cases for an ontology module. In most cases, the documentation for a module would describe some rationale for creating the module which we used as a use-case, e.g., modularising DMOP for automated reasoning (U2) [78]. In some cases, we had to assume the use-case, e.g., that the BFO Occurrents module was created for reusing a specific subset of the BFO ontology (U7).

Modules that were found on the web include those of type T1 (ontology design pattern modules), T2 (subject domain modules), T3 (isolation branch modules), T8 (optimal reasoning modules), T11 (high-level abstraction modules), T12 (weighted modules), T13 (expressiveness sub-language modules), and T14 (expressiveness feature modules). We could not find any modules of type T4 (locality modules), T5 (privacy modules), T6 (domain coverage modules), T7 (ontology matching modules), T9 (axiom abstraction modules), and T10 (entity type modules), so we generated them by using tools (SWOOP, OWL Module Extractor, and TaxoPart) or manually. 74 of the modules were publicly available, and the rest were generated for this study. Thereafter, each of the modules was classified according to the dimensions: its type, use-case(s), property(s), and technique.

Given the amount of publicly available modules that were found for the study (n=74), it appears that many modules were created specifically for this study and would thus affect the results of the framework. However, a large amount of the generated modules, 78.3%, (n=90) of the modules were created as T7 (ontology matching

modules), each with fewer than 5 entities, from just 2 source ontologies using the TaxoPart tool. It appears that the nature of ontology matching modules is to have many tiny modules to promote processing for the ontology matching tools. For the rest of the generated modules, there was 10.4%, (n=12) generated with SWOOP, 8.7%, (n=10) manually created, and 2.6%, (n=3) generated with OWL Module extractor. The classification for each module is shown in the table in Appendix A.

3.6.2.1 Frequency of use-case

The frequency of each use-case among the set of ontology modules is shown in Figure 3.4. The dominant use-case or purpose among the modules was U6 (collaborative efforts) which accounted for over 70% of the use-cases. Modules of U6 in the sample set include the myExperiment [111] and Gist [103] ontologies. Indeed, ontology modularisation has been motivated by the need for collaboration among multiple ontology developers in a number of publications [124, 29, 152]. One of the success factors of the SNOMED ontology project [97] is collaboration, which is unsurprising because it is large, containing over 300 000 entities and thus requires a team of experts for development.

Thereafter, U4 (processing) followed with 49.74% of the modules. There were many such modules in this set because the Spatial [32] and Common Anatomy Reference Ontology (CARO) [57] ontologies were automatically split with a specialised ontology alignment partitioning tool, TaxoPart. This resulted in a large number of modules containing, in most cases, fewer than 5 entities, that would allow for easy processing for use with automatic alignment tools.

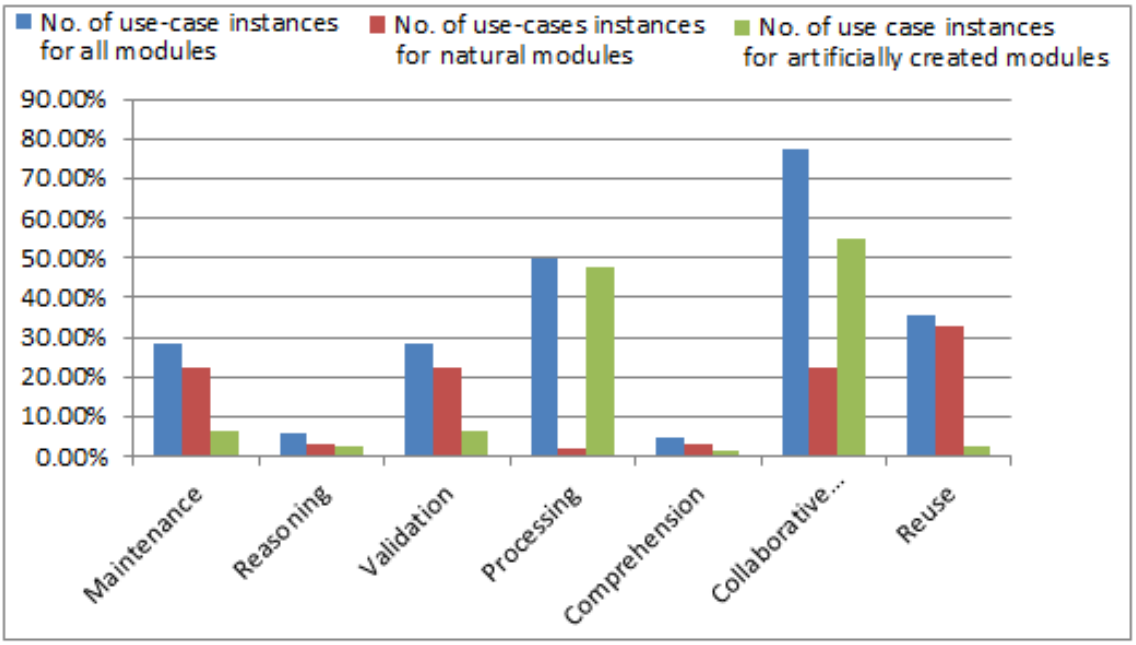


Figure 3.4: The frequency of each use-case for the set of 189 modules.

Next, U7 (reuse), U1 (maintenance), and U3 (validation) use-cases account for 35.45%, 28.57%, and 28.57% of the modules, respectively. Modules motivated by all three of these use-cases include the data mining OntoDM [121], myExperiment [111], and OntoSpace [11] ontology modules whereby a large domain was divided according to subject domains.

U2 (reasoning) and U5 (comprehension) were the least popular use-cases, with 5.82% and 4.76% of the set, respectively. For reasoning, there was some split domain DMOP ontology modules motivated for divide-and-conquer reasoning as well as a less-expressive EL language module for DMOP [78]. For comprehension, there were lighter versions of ontologies with less knowledge, such as BioToplite based on BioTop [140], and GFO-Basic based on GFO [61].

From the set of modules, all the use-cases are present for both the natural module types and the artificially created module types for the study.

3.6.2.2 Frequency of type

Figure 3.5 shows the frequency of each type for the set of 189 modules is skewed toward module type T7 (ontology matching modules). From the 189 modules, almost half of them were ontology matching modules; this is because the TaxoPart tool generated a large number of ontology matching modules for the two source ontologies, CARO and Spatial ontologies, where each generated module contained less than 5 entities in most cases. Second, there was a considerable number of T2 modules (subject domain modules), 22.22% (n= 42 modules); some of them were freely available on BioPortal ontology repository [163] and others were available on their respective project pages.

For the remaining types of modules in the set, there were very few of each type, ranging from 6.88% to 0.53%. These module types were difficult to find in existing repositories, and in cases where publications described such modules, URLs to the test files were not provided, or if they were, the URLs were no longer working. Many of these modules were generated for this experiment, and in the next paragraph, we provide a breakdown of which modules were found from existing resource.

The natural modules were T1 (Ontology design pattern), T2 (Subject domain), T3 (Isolation branch), T8 (Optimal reasoning), T11 (High-level abstraction), T12 (Weighted abstraction), T13 (Expressiveness sub-language), and T14 (Expressiveness feature). The module types that could not be found naturally, hence generated for this study, were types T4 (Locality), T5 (Privacy), T6 (Domain coverage), T7 (Ontology matching), T9 (Axiom abstraction), and T10 (Entity type abstraction).

3.6.2.3 Frequency of technique

For the frequency of techniques among modules, as displayed in Figure 3.6, it is apparent that MT1 (graph partitioning) is the most popular of the techniques, with 54% of the modules. This is because graph partitioning techniques were used by the TaxoPart tool for the large portion of Spatial and CARO ontology matching modules, discussed in the previous section. Furthermore, graph partitioning approaches were applied in the SWOOP partitioning algorithm for splitting up the large domain

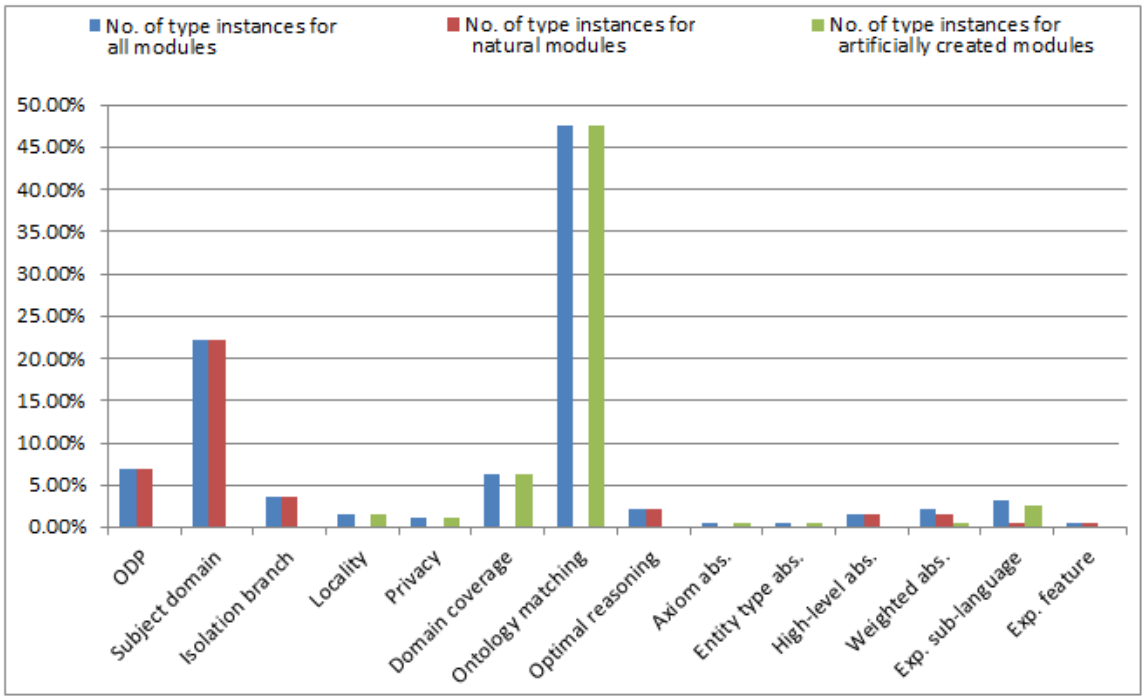


Figure 3.5: The frequency of each type for the set of 189 modules; exp. = expressiveness, abs. = abstraction.

modules for the Amino Acid [145], Edam bioinformatics [70], and MEO Metagenome and Microbes Environmental[2] domain ontologies.

Next, MT7 (*a priori*) modularity techniques were used for 27% of the modules. These sets of modules include the set of aforementioned Architectural, Gist, and OntoDM modules. MT8 (manual methods), accounted for 15% of the modules, including, the BioToplite ontology, and the Set ontology design pattern [27]. Lastly, MT4 (locality-based) techniques accounted for the smallest number of modules, 4%. These included a module with the seed signature seizure_types, based on the Epilepsy ontology [134]. Indeed, the locality-based modularity technique and principles have been discussed in a number of existing works [30, 156, 158, 135] but evidence of such modules in applications is scarce.

The techniques that were used for the natural module types include MT8 manual, MT4 locality, and MT7 *a priori* techniques. The techniques that were used for generating the artificial module types for the study were MT1 graph partitioning, MT4 locality, and MT8 manual techniques.

From the data, we observe that there is a heavy reliance on using manual methods for module creation. For 9 out of the 14 module types, manual methods were used for module creation since tools could not be applied, for instance, the BFO Continuants module which will be discussed in Section 4.1.1. The remaining 5 were generated with tools.

[2] `http://metadb.riken.jp/metadb/ontology/MEO` last accessed: 20 June 2017.

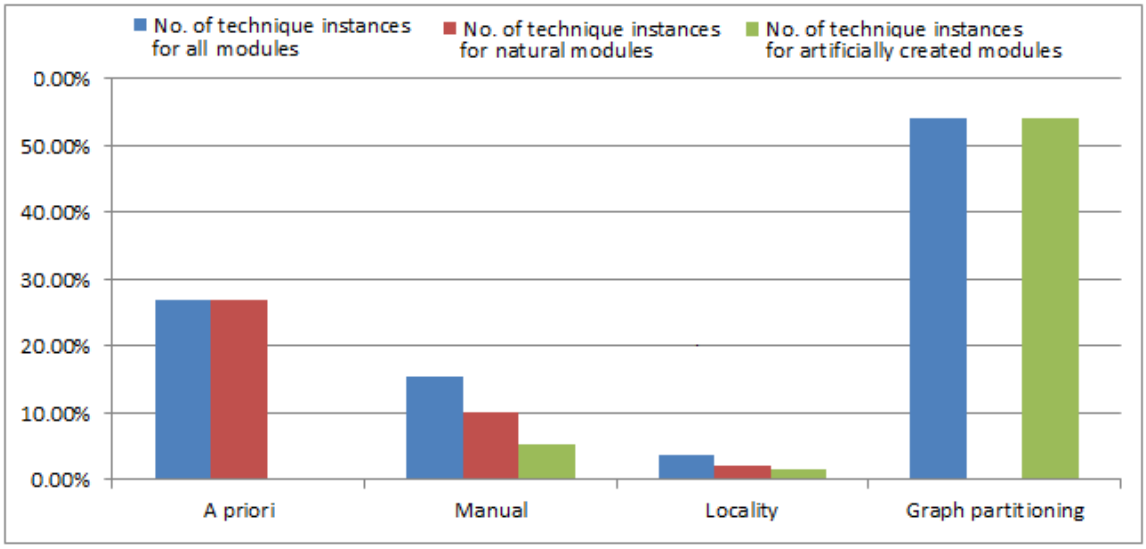


Figure 3.6: The frequency of each technique for the set of 189 modules.

3.6.2.4 Frequency of property

For the frequency of properties of modules, as displayed in Figure 3.7, property P5 (stand-alone) and P6 (source ontology) are exhibited in most of the modules (73.02%). Indeed, a large number of modules in the set contain no links or imports to other modules and are thus stand-alone, and most modules in the set are based on an original ontology.

Property P2 (information removal) is present in 68.25% of the modules, meaning that some detail is removed resulting in a smaller module with less knowledge. Property P7 (proper subset) is also present in 68.25% of the modules. The remaining properties, P1, P3, P4, P5, P6, and P10, are present in only a few of the modules ranging from 19.04% to 0.53%. P3.1 (breadth abstraction) is only exhibited in 1 module, in the FGA_taxonomy module, that was generated for this study for creating a bare taxonomy from the original Fungal Gross Anatomy (FGA) ontology[3]. For the refinement property, P4, its low presence (4.23%) in the set is no surprise because refinement is concerned with adding more detail to a module, thus refining it, and going against the basic definition of modularity in which a module is a smaller subset of a source ontology. Refinement existed in the DMOP-WithoutInverseProperties module because some OWL language features of the ontology were removed to improve the reasoning but new axioms were added to preserve the semantics of the ontology. Refinement also occurred in most of the modules that were created by partitioning using SWOOP because new axioms were introduced to enable linking among them.

There were 11 sets of inter-related modules. 72.73% of them exhibit P9 (overlapping), whereby entities in a set exist in more than 1 module of a set. The overlapping property in a module ensures the knowledge preservation within a set of modules but

[3] `http://www.yeastgenome.org/fungi/fungal_anatomy_ontology/` last accessed: 20 June 2017.

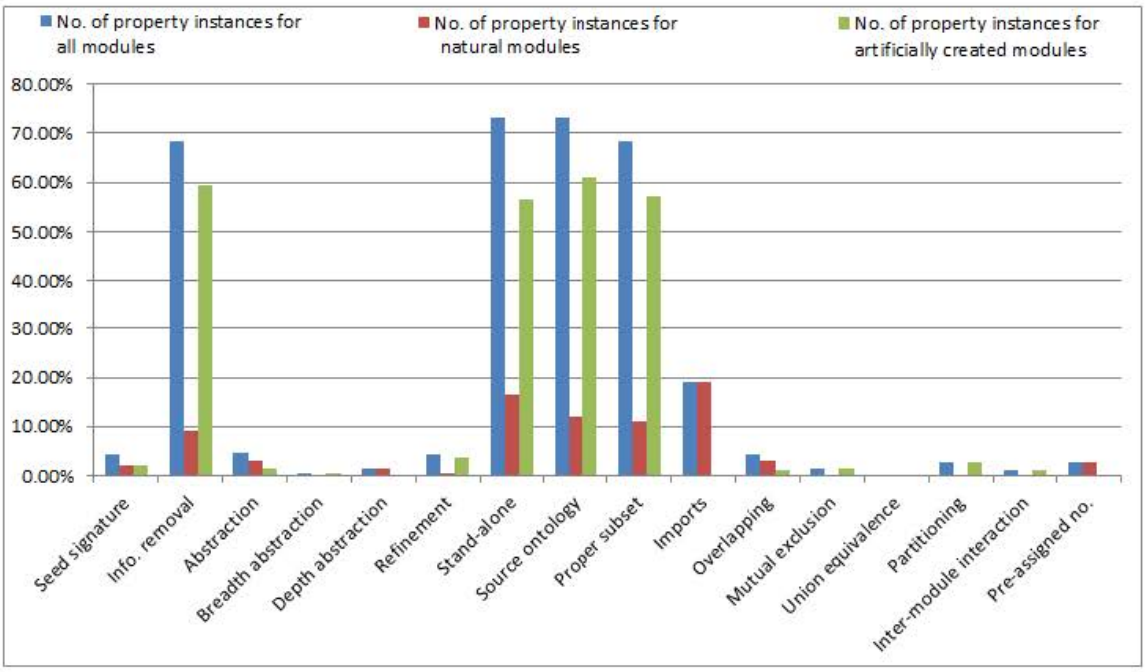


Figure 3.7: The frequency of each property among modules.

poses other challenges such as module maintenance and consistency. Property P12 (partitioning) was also a prevalent property among sets of inter-related modules, consisting of 45.45%. Since most of the modules were generated using graph partitioning techniques (recall the CARO and Spatial alignment modules), this is no surprise. Property P14 (pre-assigned number of modules), was present in 45.45% of the modules in the set, whereby the number of modules to be created for the system is known. Such modules include the set of Gist modules, and the set of myExperiment modules. Property P10 (mutual exclusion) was present in 27.27% of the module set, whereby entities were not shared across modules; the MEO, CARO and Spatial ontology module sets exhibited this property. Property P13 (inter-module interaction) is exhibited in 18.18% of the ontology module sets; it is present in the set of Amino acid modules [145], and the set of EDAM bioinformatics modules, thanks to the ε-connections links generated by the SWOOP partitioning tool to allow interaction.

Property P11 (union equivalence), was not present in any of the sets of modules as existing tools failed to ensure such a property. When the module sets were merged to check for the union equivalence, there were two reasons for the lack of union equivalence. Firstly, there were extra axioms added to the modules using ε-connections for inter-module interaction thus for some sets, the union of the modules were larger than the original ontology. Secondly, SWOOP did not preserve the annotation axioms of the original ontology; thus, for some sets, the union of the modules were smaller than the original ontology.

For the natural modules, all the properties except P10 (Mutual exclusion), P11 (Union equivalence), P12 (Inter-module interaction), and P13 (Pre-assigned number of modules) exist. For the artificially created modules, all the properties except P3.2

(depth abstraction), P8 (imports), and P11 (Union equivalence) exist.

3.7 A framework for ontology modularity

In this section, we present the framework for ontology modularity. The framework was published in [85] and [86]. We begin with the methodology for creating the framework and follow with the dependencies between the ontology modularity dimensions.

3.7.1 Methodology

We have performed the following steps towards creating a framework for ontology modularity:

1. Identify the issues and questions concerning modularity. These issues and questions were introduced in Section 1.2 and new specific questions were raised in Section 3.6.

2. Identify the relevant dimensions concerning ontology modularity. Five main dimensions were identified from an analysis of existing modules in Section 2.2: use-cases, types, techniques, properties, and evaluation metrics.

3. Populate each dimension with criteria. For the framework, we define and populate four of the dimensions with criteria: the use-cases, types, techniques, and properties. This is done in Sections 3.2- 3.5. The evaluation metrics dimension is not included in the framework as it requires additional investigation in a different direction. For evaluation metrics, an investigation needs to be performed on the quality of ontology modules. Developers need the following: 1) a tool which includes all the metrics for modules to compute them automatically for an ontology, and 2) to determine which metric values correspond to which module types, i.e., how to measure if a module is of good quality. We investigate this in Section 4.2.

4. Perform an experimental evaluation using modules whereby modules are characterised according to the dimensions. This experimental evaluation was performed in Section 3.6.

5. Identify relationships between the dimensions of the framework. This follows in Section 3.7.2.

3.7.2 Dependencies between dimensions

Given the insights obtained in Section 2.8 to elucidate modularity dimensions and properties, and with the assessment of actual usage of ontology modules and modularisation by ontology developers (Section 3.6), we go one step further with this survey by elucidating observed dependencies between the properties. This leads to a basic framework for modularity, of which the high-level view is shown in Figure 3.8.

Based on our comprehensive review of the literature, analysis of ontologies and ontology modules, and usage of modularisation tools, to the best of our knowledge, our framework is indeed exhaustive.

Figure 3.8: A high-level view of the framework for modularity.

The dependencies are used to refine and answer the earlier proposed questions:

1. Given that we wish to create an ontology module with a certain purpose or *use-case* in mind, which modularity *type* of module could this result in? (How do module types differ with respect to certain use-cases?)

2. If we wish to create a module of a certain *type*, which is the best *technique* to use? (Which techniques can we use to create modules of a certain type?)

3. By using a particular *technique*, which *annotation features* will the resultant module exhibit? (Which techniques result in modules with certain annotation features?)

The dimensions of the framework are related as follows. A module's *use-case* results in modules of a certain *type*. A module of a certain *type* is created by a modularisation *technique*. Modularisation *techniques* result in modules with certain *properties*.

Answers to these questions are mentioned in following diagrams. For instance, regarding question a, if we wish to create an ontology for the use-case of U5 comprehension, this could result in a T9-T12 abstraction type module (see Figure 3.9). Thereafter, for question b, if the module type is either one of T9-T12 abstraction, the technique for modularisation is MT8 (manual methods); see Figure 3.10. Lastly, for question c, when MT8 manual methods are used, the resulting modules exhibit the following annotation features: P1 (seed signature), P2 (information removal), P3 (abstraction), P3.1 (breadth abstraction), P3.2 (depth abstraction), P4 (refinement), P5 (stand-alone), P6 (source ontology), P7 (proper subset), or P8 (imports) (see Figure 3.11).

The dependencies between dimensions were identified by analysing the classification table for modules (Table A in Appendix A), and drawing out dependencies between dimensions. For instance, for the first entry in Table A, the use-case is U7: Reuse and the type of module is M1: Ontology design pattern. This tells us that ontology design pattern modules (type) are created for reuse (use-case). The ontology design pattern modules (type) are created by MT8: manual modularity (technique). Manual modularity (technique) could result in modules with P2: information removal, P6: source ontology, P7: proper subset, and P8: imports (properties). The links between the various dimensions are discussed in the following sections.

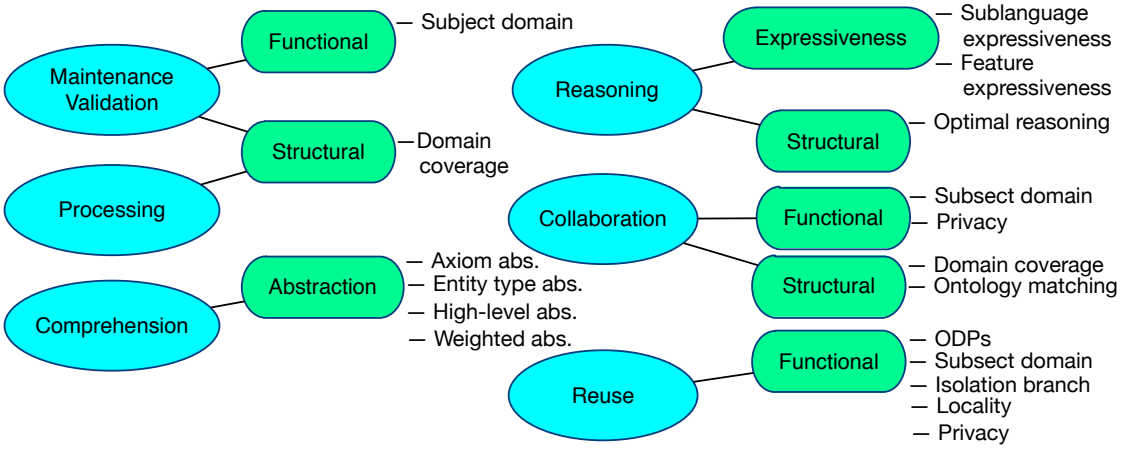


Figure 3.9: The dependencies between use-cases and module types; abs = abstraction, ODP = ontology design pattern.

3.7.2.1 Dependencies between use-case and type

We examine the relationship between use-cases and types of module to determine which types of modules the use-cases drives. The dependency relationship between the use-cases and module types are displayed in Figure 3.9. When U1 (maintenance) is the use-case, the resultant modules are T2 (subject-domain modules) or T6 (domain coverage modules), hence, maintenance results in a type of functional or structural module. For U2 (reasoning) the resultant module is T8 (optimal reasoning modules) which is a type of structural module, or T13 (expressiveness sub-language) or T14 (expressiveness feature modules) which are both expressiveness modules.

The U3 (validation) use-case results in the same modules as the maintenance use-case, i.e., T2 (subject-domain modules) or T6 (domain coverage modules). Hence, validation results in a type of functional or structural module. When the use-case is U4 (processing) the resultant module is T7 (ontology matching) or T8 (optimal reasoning modules) which are both structural modules. For U5 (comprehension), the resultant modules are either T9 (axiom abstraction), T10 (entity type abstraction), T11 (high-level abstraction) or T12 (weighted abstraction modules) which are expressiveness type modules

For the U6 (collaboration) use-case, the resultant module is T2 (subject domain), T5 (privacy), T6 (domain coverage), or T7 (ontology matching modules). T2 (subject domain), and T5 (privacy modules) are a type of structural module. T6 (Domain coverage), and T7 (ontology matching) are a type of functional module. For the U7 (reuse) use-case, the resultant module is T1 (ontology design pattern), T2 (subject domain), T3 (isolation branch), T4 (locality), and T5 (privacy), which are all functional type modules.

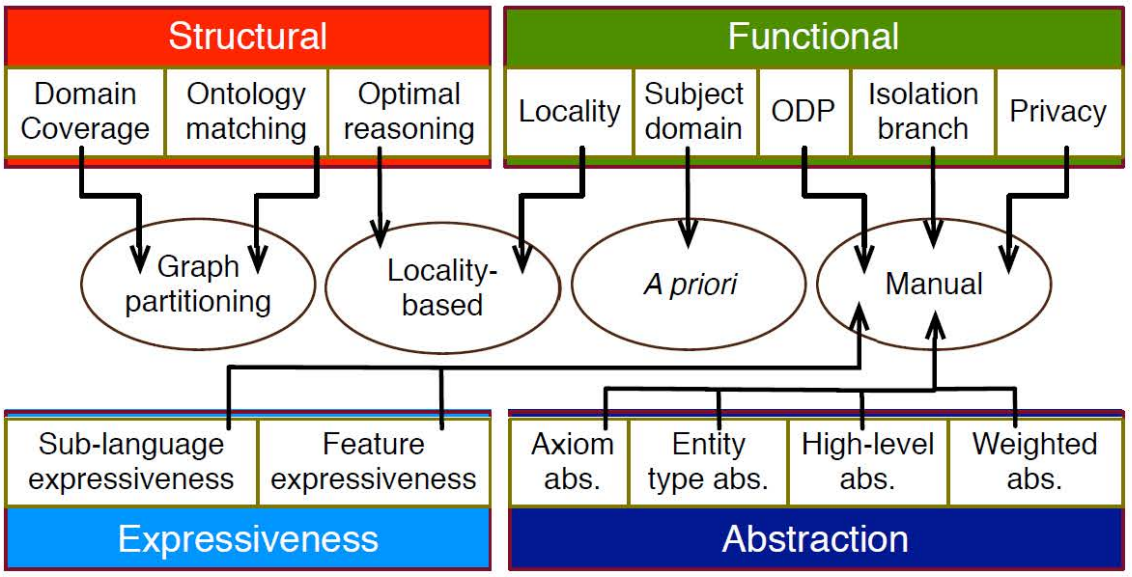


Figure 3.10: The dependencies between module types and techniques; abs = abstraction, ODP = ontology design pattern.

3.7.2.2 Dependencies between type and technique

Next, an ontology developer needs to know which technique to use for creating a type of ontology module. The dependencies between the module types and techniques are displayed in Figure 3.10. For T1 (an ontology design pattern module), both MT8 (manual) and MT7 (*a priori*) methods are used. To create T3 (isolation branch), and T5 (privacy modules), thus far there is only evidence of MT8 (manual methods) being used. To create T2 (subject domain modules), when a large domain must be divided according to specific subject domains, MT7 (*a priori* modularity) techniques are used. For T4 (locality modules), naturally, MT4 (locality-based modularisation) approaches are used. Therefore the three types of techniques used for creating structural modules. in general are MT4 (locality), MT7 (*a priori*), and MT8 (manual) methods.

To create modules for T6 (domain coverage), or T7 (ontology matching), then MT1 (graph partitioning) techniques are used. For T8 (optimal reasoning modules), i.e., when an ontology is large and must be divided to assist with reasoning, then MT4 (locality-based methods) is used. Thus, MT1 (graph partitioning) or MT4 (locality-based) methods are used to create structural modules. For all abstraction and expressiveness modules, only manual methods are used.

3.7.2.3 Dependencies between technique and property

Next, we examine the properties exhibited by modules created with different techniques. The dependency relationship between all these techniques and properties are displayed in Figure 3.11. When MT1 (graph partitioning) techniques are employed, the modules have the following properties: P2 (information removal), P5 (standalone), P6 (source ontology), or P7 (proper subset). Since MT1 (graph partitioning)

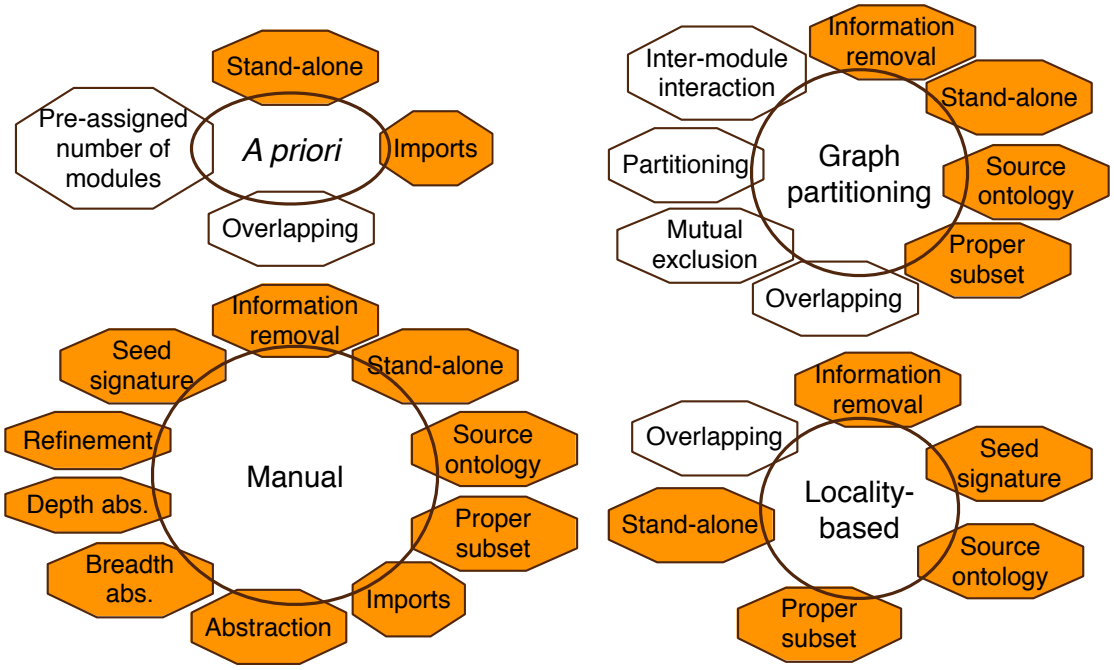


Figure 3.11: The dependencies between techniques and properties. The shaded hexagons represent the modification and relational properties, the unshaded represent the set properties.

always results in a set of modules, there are set properties that exist, which are P9 (overlapping), P10 (mutual exclusion), P12 (partitioning), or P13 (inter-module interaction). Using MT4 (locality methods) techniques results in modules with the following properties: P1 (seed signature), P2 (information removal), P5 (stand-alone), P6 (source ontology), or P7 (proper subset). Since MT4 (locality methods) have also been used to create a set of modules, its modules exhibit the set property P9 (overlapping).

Considering the results of the dimension, the experimental evaluation, and its limitations, we now return to the answers for the questions posed in Section 3.6.

1. How do module types differ with respect to certain use-cases?
   The manner in which a module use-case affects the type of module that will be created is shown by the dependencies between the use-cases and types in Figure 3.9.

2. Which techniques can we use to create modules of a certain type?
   The manner in which a module type affects the technique that should be used is shown by the dependencies between the type and technique in Figure 3.10.

3. Which techniques result in modules with certain annotation features?
   The manner in which the module technique affects the annotation features that it exhibits is shown by the dependencies between the module technique and annotation features in Figure 3.11.

Overall, this is, to the best of our knowledge, the, thus far, most comprehensive list of aspects of ontology modules and a first insight into the dependencies between all those dimensions and criteria.

3.8 Evaluating the framework

We now evaluate the framework using existing ontologies and conceptual data models as case-studies to demonstrate the usability of the framework.

3.8.1 Ontology case-studies

In this section, we test the framework to guide the modularisation process. We randomly select four case studies of ontology modules. These modules were not from the 'training' set of modules that were used in the experimental evaluation to create the framework. We consider these modules as the 'testing' set of modules. Examples 3 to 5 were published in [86] and example 6 in [85]

Example 3 (QUDT ontology modules) *The Quantities, Units, Dimensions and Data Types (QUDT) ontologies are a set of ontology modules focussed on terminology used in science and engineering for representing physical quantities, units of measure, and their dimensions [63].*

Use-case *Identify the use-case for the modularity of the set of ontologies. In the set of seven QUDT modules, there is a total of 4067 entities; hence it is a large domain and is divided into several modules to facilitate maintenance and validation. One of the goals outlined in the QUDT specification states that parts of the vocabulary will be of interest to certain users or applications depending on the use-case. Hence the individual modules in the set could be used for reuse. Lastly, the modular approach of the subject domain means that a team of experts could work with specific modules thereby enabling collaborative efforts. Hence the four use-cases for the QUDT ontology modules are maintenance, validation, reuse, and collaborative efforts.*

Type *Since the use-case(s) have been identified, we can now refer to the framework to check for the next step of the modularisation process. The framework states that use-cases result in module types. According to the dependencies, when maintenance or validation is the use-case, this results in subject-domain or domain coverage modules. Collaborative efforts result in subject domain, privacy, domain coverage, or ontology matching modules. For reuse, the resulting module(s) is ontology design pattern, subject domain, isolation branch, locality or privacy modules. Across all four use-case to type dependencies, the common module type that maintenance, validation, collaborative efforts, and reuse result in is the subject domain modules. Hence the set of QUDT modules is a set of subject domain modules, whereby a large ontology is divided according to the subject domains within the ontology.*

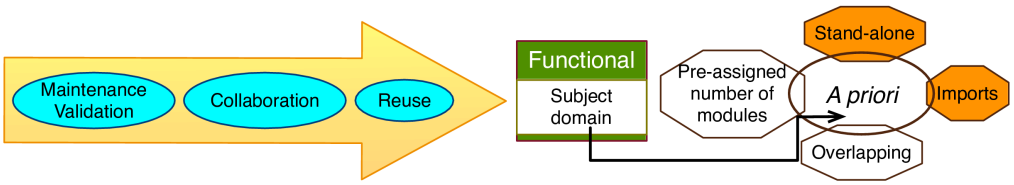


Figure 3.12: The dependencies between the module dimensions for QUDT modules.

Technique *The type of module drives the modularisation technique. Given that the module is a subject domain module, according to the framework, such modules are created using the a priori technique. Hence, the modules to be created are decided at the onset of ontology development. There is no source ontology for QUDT containing the entire domain, there is only a set of modules hence we can assume that a modular approach was decided at the onset of ontology development.*

Property *By using an a priori technique, which module properties can we expect of the QUDT set of modules? The framework states that a priori techniques result in modules each with the stand-alone or imports properties, and as a set of modules, they exhibit the overlapping or pre-assigned number of modules properties. Two of the QUDT modules exhibit the stand-alone property only. The remaining six modules each exhibit the imports property only. As a set of modules, the modules exhibit the following set properties: overlapping and pre-assigned number of modules.*

A summary of the case-study for the QUDT modules is shown in Fig. 3.12.

Example 4 (Foundational Model of Anatomy module) *The Foundational Model of Anatomy Ontology (FMA) is a reference ontology for the domain of human anatomy [133].*

Use-case *We wish to create a module with a small selection of knowledge from the FMA ontology, particularly to be reused in the creation of an ontology about infection. For our proposed ontology about infection, we require all the information about body substances from the FMA ontology to be reused.*

Type *Given that the identified use-case for the module is reuse, we refer to the framework to check which type of modules result from reuse. The reuse technique results in all the functional module types: ontology design pattern, subject domain modules, isolation branch, locality, and privacy modules. Since we wish to extract a subset of the FMA ontology, we consider the creation of either an isolation branch module or a locality module. Looking at the entities about body substances in the FMA ontology, we realise that we do not wish to preserve entities with weak dependencies or relations to the body substances entities. Hence, we decide to create an isolation branch type module.*

Technique *For the isolation branch type module, the only technique that can be used to create such a module is manual methods. To create a branch module of body substances from the FMA ontology, we delete all entities besides the taxonomic branches referring to body substances entities.*

Property *The framework states that when manual methods are used, the resulting module could have the following properties: seed signature, information removal, abstraction, breadth abstraction, depth abstraction, refinement, stand-alone, source ontology, proper subset, or imports. The FMA body substances module exhibits the following properties: seed signature, information removal, stand-alone, source ontology, and proper subset.*

Example 5 (OpenGalen EL module) *The OpenGalen ontology [128] is a common reference module for application-independent and language-independent model of medical concepts.*

Use-case *ELK [74] is a reasoner for the lightweight ontology language OWL EL created for improved reasoning for large ontologies. A study on the evaluation of the ELK reasoner requires modules for the use-case of reasoning.*

Type *Since the use-case is reasoning, this could result in the following types: optimal reasoning modules, expressiveness sub-language or expressiveness feature modules. In order to test out the ELK reasoner, the development team created an EL version of the OpenGalen ontology*[4]*. Hence the module type is an expressiveness sub-language module. The EL version of the OpenGalen ontology was created by removing all inverse role, functional role, and role chain axiom of the OpenGalen ontology.*

Technique *Thus far, the technique used for creating expressiveness sub-language modules is manual methods. The study on the evaluation of the ELK reasoner states that an EL version of the OpenGalen ontology was created by removing* InverseObjectProperties *and* FunctionalObjectProperties *axioms, hence we can assume that manual methods were used for this.*

Property *The framework states that when manual methods are used, the resulting module could have the following properties: seed signature, information removal, abstraction, breadth abstraction, depth abstraction, refinement, stand-alone, source ontology, proper subset, or imports. The OpenGalen EL module exhibits the following properties: information removal, stand-alone, source ontology, and proper subset.*

Example 6 (Symptom ontology) *Let us assume that we wish to reuse the Symptom ontology [6], a domain ontology about symptoms and signs of diseases.*

[4] `http://code.google.com/p/elk-reasoner/wiki/TestOntologies` last accessed: 20 June 2017.

Use-case *We wish to create a module with knowledge about symptoms that exist on the skin in order to reuse it in a larger domain ontology about dermatology.*

Type *We now refer to the framework to check which type of modules result from reuse. Reuse results in all subtypes of functional modules. Since we wish to extract a subset of the Symptom ontology, we consider an isolation branch or locality module. We wish to preserve all entities with dependencies to the skin entities; hence we create a locality module.*

Technique *For the locality module, a locality-based technique is selected, using the OWL Module extractor tool to extract a module containing knowledge about the skin symptoms, with a seed signature* skin and integumentary tissue symptom.

Property *Modules created with locality-based techniques could have these properties: information removal, seed signature, source ontology, proper subset, stand-alone, and overlapping. The generated module exhibits all these properties, except overlapping (since it is not a set).*

3.8.2 Conceptual data model case-studies

We now assess whether the framework has any transferability, for the usage of conceptual data models where a modular approach is taken. This work was published in [90]. We analysed 15 conceptual data models that exist as a set of modules to uncover information about them using the framework for modularity from Section 3.7. Six of these data models were from existing student projects at the Universidad Nacional del Sur (UNS) where for each model, a few modules were linked using the CASE tool, ICOM [40]. For the six ICOM projects, some of them are cognitive overload scenarios as they are large models that have been designed as modules with links among them to deal with managing a large amount of knowledge and others are integration scenarios where heterogeneous models have been combined with inter-model links. These models cover domains about telecommunications, college, governance, etc. The remaining nine data models were cases where we used publicly available data models in different conceptual data models (UML, ER, and ORM) and integrated them; they are all classified as integration scenarios. These models cover domains about banking, car sales, flights, etc. The test files for the models and analysis are available at `http://www.meteck.org/SAAR.html`.

An example of a cognitive overload scenario is shown in Fig. 3.13, where a module with information about governance has links to a module about memorandums. An example of an integration scenario where we manually aligned models is shown in Fig. 3.14. The solid lines link entities of the same type, e.g., the object types er:Airplane and uml:Aircraft, the long-dashes dashed lines links entities that are semantically very similar (e.g., a full attribute, as in uml:Airport.name and an attribute without data type, er:Airport.name), and the short-dashes dashed line required some transformation, such as between er:Airplane.Type (an attribute) and uml:Aircraft_Type (a class) and between er:Airport.Code (an identifier, without data type) and uml:Airport.ID (a plain attribute, with data type).

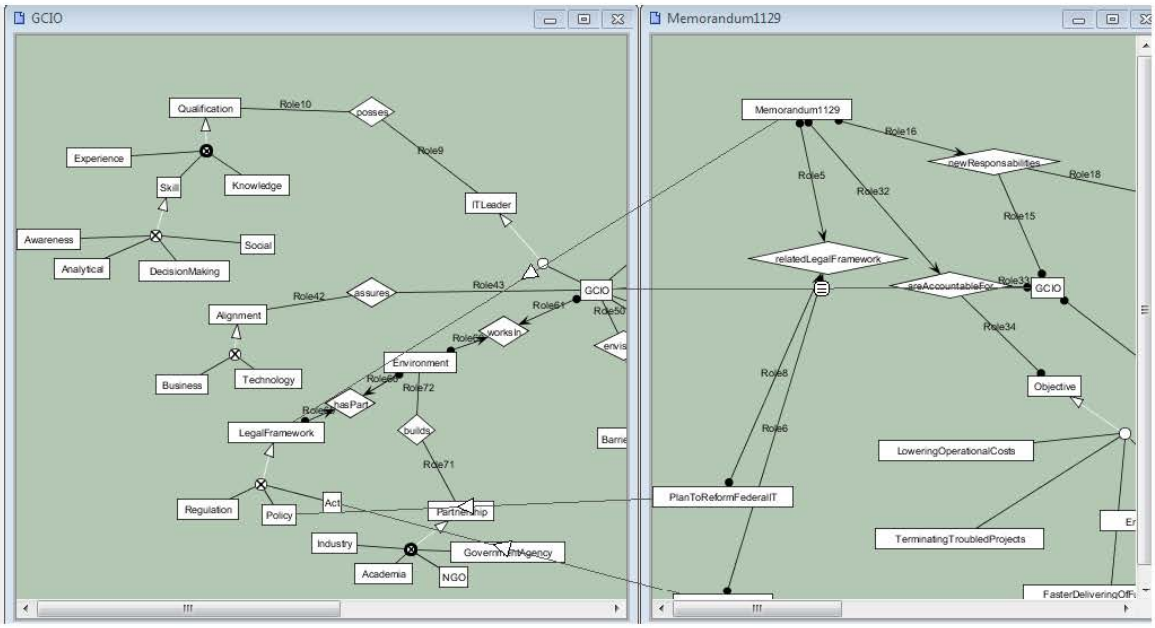


Figure 3.13: A cognitive overload scenario: a conceptual data model on the governance domain with intermodel assertions between modules.

We used the framework for modularity from Section 3.7 to classify the projects. Little was known about the use-cases for the conceptual data models, so we began by mapping each model to a type, by definition. For this, all the models of the cognitive overload scenario are T2: subject domain modules, i.e., they are created when an ontology is subdivided according to the subject domains present in the ontology. According to the framework, the use-cases for such modules are maintenance, validation, collaborative efforts, and reuse (see Figure 3.9). As for the technique for subject domain modules, an *a priori* technique is usually used, one in which the modules of the domain is decided at the onset of development (see Figure 3.10). Lastly, the properties that are linked to the set of modules created by the *a priori* technique could be pre-assigned number of modules or overlapping. The properties linked to a single module created by the *a priori* technique could be stand-alone or imports (see Figure 3.11).

For the modules on system integration, they correspond to T11: high-level abstraction modules of the framework, i.e., for when there is a module in the system where only higher-level classes of the ontology are required, and this decreases the vertical structure of the ontology. According to the framework, such modules have a use-case of comprehension and a technique of manual modularity. The properties for modules created by manual modularity techniques are: source ontology, proper subset, and depth abstraction. The full classification according to the use-case, type, technique, and property is shown in Table 3.2.

By using the framework, the conceptual data model developer can gain more insight about their conceptual data models. One can check which type of module a data model is classified as, the use-cases for creating such modules, and the techniques that

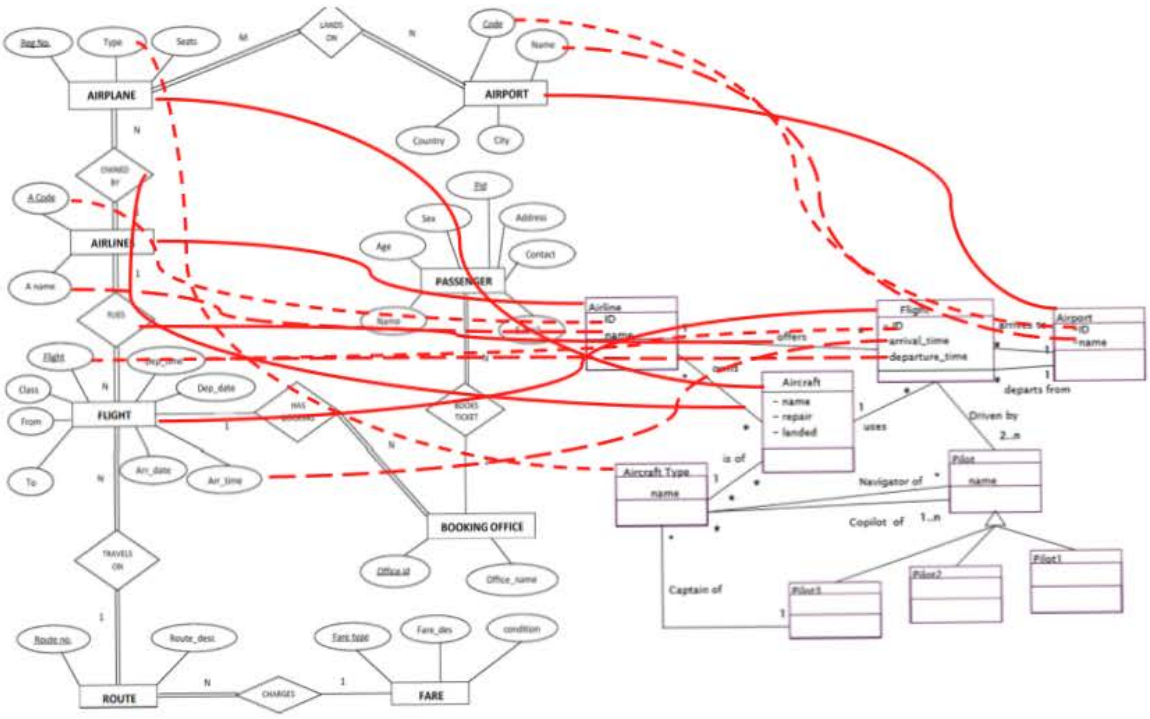


Figure 3.14: An integration scenario: The intermodel assertions between Flights models in EER and UML. The solid lines link entities of the same type, the long-dashes dashed lines links entities that are semantically very similar, the short-dashes dashed line required some transformation.

are used to create such conceptual data model modules. Furthermore, the framework provides the developer with information about the properties that their data models ought to exhibit. For future cases where a conceptual data model needs to be created, the framework can be used to guide the modularisation process for the conceptual data models. This can be achieved by referring to the framework to check which combination of technique results from the use-case of the module (Figures 3.9) to give a developer insight on how to modularise. Thereafter, the developer can check the other dependency diagrams of the framework (Figures 3.10- 3.11) to uncover information about the type and properties of the modules.

3.9 Discussion

Following the inconsistencies and gaps from current modularity definitions first introduced in Chapter 2, we created our own definition for a module. The dimensions that were defined and populated in this Chapter are used towards creating a framework for modularity. To create a framework, we had to perform an experiment whereby we collected and classified a set of ontology modules using the dimensions. A limitation of the method used in the classification experiment is that more subdimensions may appear in the future as ontology engineering progresses. For instance, new types of

Table 3.2: Classifying the conceptual data model projects using the framework for modularity.

	Use-case	**Type**	**Technique**	**Property**
Cognitive overload projects	Maintenance	Subject domain	*A priori*	Pre-assigned no. of modules
	Validation			
	Collaboration			Overlapping
	Reuse			
Integration projects	Comprehension	High-level abstraction	Manual modularity	Source model
				Proper subset
				Depth abstraction

modules are found to be needed as the use of ontologies expands into more application areas. To address this, we propose that the dimensions be periodically updated as required.

A framework for modularity assists the ontology developer in providing guidance for the modularisation process. Questions that an ontology developer might have before attempting the modularisation process include, "which tool could I use to perform modularity", or "what properties should my module exhibit". This can now be looked up using the novel, empirically-based, evaluated framework for modularisation. The framework comprises four of the five dimensions: use-case, technique, type, and properties.

The framework for modularity was created using grounded theory as a research method. While grounded theory is beneficial as it allows us to tackle the problem with an open-mind and use real-life data, there is also a drawback. It relies heavily on empirical data, without having an initial hypothesis to test. This opens up the possibility for biased data. To resolve this issue, we have included diverse data: various modules from different sources representing different domains, and some that were generated with modularisation tools.

In addition to assisting with guiding the entire modularisation process, the properties of modules can be used for ontology annotation towards improved metadata. Metadata promotes ontology discovery and reuse, and repositories such as BioPortal [163], Ontohub [107], and ROMULUS [80] use metadata models. There is limited metadata concerning modular ontologies [80], which now can be refined and improved further. If a module is not annotated with some properties, it will indeed be difficult to figure out its properties, but, in theory, at least, it may be possible to determine them when either the source ontology or the other modules in the set are known. Thus the resultant framework of the module dimensions and dependencies can be used to steer the modularisation process, and form the basis for metadata for ontology modules, which promotes ontology reuse.

The classification of modules according to techniques reveals that there is a heavy reliance on using manual methods for module creation. Thus, the problem, that there are insufficient tools for modularisation still exists, but the classification of modules using the dimensions from this Chapter has refined it as follows. The clas-

sification reveals that for 9 out of the 14 module types, manual methods were used for module creation. For creating ontology design patterns, isolation branch, privacy, sub-language expressiveness, feature expressiveness, axiom abstraction, entity type abstraction, high-level abstraction, and weighted abstraction modules, manual methods are used. We now have a list of the type of modules that rely heavily on manual methods. The implementation of tool-based methods as a technique for some of the abstraction and expressiveness type modules is within reach, given the recent advancements in ontology API libraries such as the OWL API [69].

For the tools that are available, they are not sufficient. They are hardly maintained and sometimes not usable. We had hoped to generate modules from partitioning large ontologies. However, the SWOOP partitioning tool could not be applied for large ontologies such as the FMA ontology [133] as it could not open it, despite manually changing the java heap space parameters. We had also hoped use PROMPT traversal views with Protégé [114] for query-based modularity, but it malfunctioned and returned a null pointer exception. OWL module extractor was considered for extracting DMOP modules. It extracts modules by using an input set of terms as a signature while ensuring the logical completeness of the module. This means that for every axiom of the original ontology, the meaning of the axiom is preserved in the module. Due to dependencies between entities in the DMOP ontology and the logical completeness constraint, OWL module extractor generated too large a module to use to improve reasoning.

An evaluation of the framework with ontology scenarios proves that the framework is indeed useful and worthwhile for module development. The example of the application of the framework to the QUDT, FMA, OpenGalen, and Symptom ontology module extraction demonstrate that the framework is promising for guiding the modularisation process. The framework provided guidance in classifying the module according to its type, which technique to use for modularity, and in addition, which annotation features the module should exhibit.

Using conceptual data models for the evaluation demonstrate the transferability of the use-case from ontologies to conceptual data models, and how important information about conceptual data models can be uncovered.

3.10 Conclusion

In this chapter we identified, discussed, and populated dimensions to demonstrate that modularity is not a straightforward, solitary concept but rather a methodological approach with specific conditions resulting in different ontology modules.

We have identified issues and questions concerning modularity. The issues are that there is difficulty in selecting the appropriate modularity technique, insufficient modularity tools for applications, and it is not clear which technique should be applied for which scenario. To address them, we identified and populated dimensions concerning modularity which was used in an experimental evaluation with a set of 189 ontology modules resulting in dependencies among the modularity dimensions. The classification of the modules using the dimensions led to the creation of a framework

for ontology modularity which can be used to solve the developer's issue concerning modularity technique selection, to refine the issue concerning insufficient tools for modularisation, and to systematically guide the entire modularisation process. The examples of the application of the framework to the various ontology module extraction and conceptual data model use-cases demonstrate that the framework is promising for guiding the modularisation process.

Chapter 4

Theories and techniques for modularitsation

In this Chapter, we present theories and techniques for modularity. We begin the chapter with an experiment to gain insight into the current state of the art concerning modularisation in Section 4.1. The problems uncovered in Section 4.1 led us to an investigation of ontology metrics, and we present work on evaluation metrics and the Tool for Ontology Module Metrics (TOMM) in Section 4.2. We identified existing metrics and created new ones which are then implemented into the TOMM tool to measure the quality of a module. We performed an experimental evaluation with TOMM to gain insight about evaluation metrics for ontology modules. The experiment, with ontology modules, reveals for which type of module, which metrics are relevant and their expected values, therefore, it can solve the problem that it is unclear whether a module is of good or bad quality. In Section 4.3, we investigate ontology interchangeability and introduce SUGOI, a tool that has been designed to interchange modules for promoting semantic interoperability and improving modularisation metrics. Thereafter we investigate techniques for automatically performing ontology modularisation in Section 4.4. We present five new algorithms for performing modularisation and show how these algorithms work with illustrative examples. The algorithms were implemented in the Novel Ontology Modularisation SoftwAre (NOMSA) tool and compared to other modularisation tools where it was found that for most of the features, NOMSA performs as well as or better than the other tools, with the benefit of full automation of the process. The NOMSA tool was also experimentally evaluated with a set of ontologies. We discuss our contributions in Section 4.5 and conclude the Chapter in Section 4.6.

4.1 Issues with modularisation with existing resources

The aim of this exploratory experiment is to gain better insight into the current state of the art practically and in particular, the problems that an ontology developer encounters when attempting to perform modularisation using existing resources. We

assess this by means of two case-studies.

4.1.1 Case-study: ROMULUS's modules

The intent behind the Repository of Ontologies for MULtiple USes (ROMULUS), was to have an online library of machine-processable, aligned, merged, and modularised, systematically related foundational ontologies [88]. In earlier work [88], modules were created within ROMULUS to facilitate foundational ontology reuse. We now revisit this modularisation process to investigate the problems with existing resources. The following modules have been created in ROMULUS:

- DOLCE modules:
  - DOLCE-Endurants, DOLCE-Perdurants and DOLCENoQualityAndQualia: These modules contain only a 'branch' of existing entities from the original ontologies; e.g., the DOLCE endurant entity branch and all its subclasses.
  - DOLCE-EL and DOLCE-QL: These modules are trimmed to what can be represented in OWL 2 EL and OWL 2 QL.
- BFO modules:
  - BFO-Continuants and BFO-Occurrents: These modules contain only a 'branch' of existing entities from the original ontologies; e.g., the BFO Continuant entity branch and all its subclasses.
  - BFO-EL-QL-RL: These modules are trimmed to what can be represented in OWL 2 EL, OWL 2 QL and OWL 2 RL.
- GFO modules:
  - GFO-NoOccurrents and GFO-NoPersistantsAndPresentials: These modules contain only a 'branch' of existing entities from the original ontologies; e.g., the GFO ontology with the Occurents and subclass entity branches removed.
  - GFO-Basic-EL and GFO-Basic-QL: These modules are trimmed to what can be represented in OWL 2 EL and OWL 2 QL.
  - GFO-ATO (based on the Abstract Top Level layer) and GFO-ACO (based on the Abstract Core Level): These modules contain the high-level meta-categories of GFO.

The following tools were considered for modularisation: Swoop v2.3 [73], OWL module extractor [30], and Protégé v4.3 [110]. They were considered to create the branch type taxonomic modules, the trimmed down language versions, and the abstract level modules. It was found that all three tools created modules that were too large; they contained too much data and were too similar to the original ontologies; this is explained in the remainder of this section. For BFO, in an attempt to create

a module with only an Occurrent entity branch, both Swoop and OWL module extractor generated modules that were 92% of the size of the original BFO ontology, while Protégé generated a module that was 97% of the original BFO ontology. For modularising DOLCE and GFO, it could not be done because all three tools created modules with entities that were not to be included in the branch. For instance, the DOLCE-endurants module was supposed to only contain a taxonomy of endurant entities, but it still contained entities from the perdurant branch. This is due to the fact that the tools preserve the local completeness of the modules thanks to the relationships between the endurant and perdurant branch entities in DOLCE. The consequences of these sub-optimal results with existing tools meant that the resultant modules of ROMULUS have been created manually. The relative percentage reduction of a module that was identified for these modules could be an interesting metric for measuring the quality of a module and deems further investigation in the following section.

Additional data pertaining to foundational ontologies are required to assist ontology developers with reusing an ontology effectively. Metadata values for all ontologies, including the modules are provided in ROMULUS. ROMULUS uses a few entities from existing metadata models, OMV [59] and OM^{2}R [153], but extends this considerably with its own metadata for ontology modules. The retrieval of some of the metadata for ROMULUS's modules is a tedious task that requires manual calculations. There is currently no software tool available to assist with the automatic generation of such metadata for modules. The list of metadata pertaining to modules is provided here:

- ModuleType: This is used to classify a module into a broad type.
- ModuleSubtype: This is used to classify a module into a more specific subtype.
- ModuleCoverage: This represents the value of the ontology that is covered by the module; the percentage of the original ontology that is found in the module e.g., a module that covers 50% of the original ontology.
- ModuleCorrectness: This states whether the module is logically correct, i.e., if all the axioms from only the original ontology are found in the modules, and nothing new has been added to the module.
- ModuleCompleteness: This states whether, for every axiom in the original ontology, the meaning of the axiom is preserved in the module.
- ModuleClassSize: This represents the value of the classes of the original ontology has been contained in the module.
- ModulePropertiesSize: This represents the value of the properties of the original ontology has been contained in the module.
- ModularisedAxiomSize: This is used to describe the value of the axioms of the original ontology has been contained in the module.

Metadata: DOLCE-Endurants

Entity	Value
Module details	
Module Type	Reusable component
Module Subtype	Branch
Module Type Description	This module covers a single branch of the original ontology
Module Coverage	91.69%
Module Correctness (ModuleCorrectness states whether a module is logically correct, i.e., if all the axioms from only the original ontology are found in the modules and nothing new has been added to the module.)	Yes
Module Completeness (ModuleCompleteness states whether for every axiom in the original ontology, the meaning of the axiom is persevered in the module.)	No
Module Class Size (ModularisedClassSize represents the amount of classes of the original ontology that remains in the module.)	31.08%
Module Property Size (ModularisedPropertySize represents the amount of properties of the original ontology that remains in the module.)	100.00%
Module Method	Manual
Original ontology	A Descriptive Ontology for Linguistic and Cognitive Engineering (Lite)

Figure 4.1: The metadata pertaining to module details from the DOLCE-Endurants module in ROMULUS.

The metadata with corresponding values for the module details for the DOLCE-endurants module is shown in Figure 4.1. For each module in ROMULUS, the ontology developer needs to identify metadata values from each ontology file. This task requires computations from each ontology file which is a manual process. For instance, a user needs to check whether, for every axiom in the original ontology, its meaning is preserved in the resultant module, in order to get the metadata value for the ModuleCompleteness metadata. We also note that other metadata values, such as the ModuleType and ModuleSubtype correspond to the type dimension of the ontology modularity framework presented in Section 3.7 and can be easily extracted from the framework for a particular module.

4.1.2 Case-study: Modularising the DMOP ontology

The work in this section has already been published in [78]. In an attempt to improve reasoning performance for the Data Mining and Optimization Ontology (DMOP), we attempted to modularise DMOP ontology using the following automated tools: SWOOP v2.3 [73], OWL Module Extractor [30], and Protégé v4.3 [110]. The locality-based algorithms of SWOOP and OWL Module extractor were experimented with, but they created modules that were too large because entities within the DMOP ontology have many dependencies between them; there were no isolated branches of the ontology that could be modularised. Recall in Section 4.1.1, the DOLCE

ontology could not be modularised for branch modules because of relations between the endurant and perdurant entities. This occurs in the DMOP ontology because it is merged with the DOLCE ontology. The partitioning feature of SWOOP could not be used either as it does not support ontologies that have imports to other ontologies that exist in the DMOP ontology.

We use the 'axioms by reference' method in Protégé to select entities from an ontology to copy, move, or delete. The copy and move function returned an error, therefore we used the delete option to remove unwanted entities from the module. We used the 'axioms by profile' method in Protégé to create an OWL EL profile of the DMOP ontology and we also merged the axioms of the branch modules to create a merged module. The resultant modules are as follows:

- DMOP-Branch-Endurant: A module of wholly-present entities.
- DMOP-Branch-Perdurant: A module of entities that unfold in time.
- DMOP-Branch-Abstract-Quality: A module of entities that exist in neither space nor time and property related entities.
- DMOP-Branch-Toplevel: A module that has only DMOP's top-level entities.
- DMOP-Branch-Merge: A merged module of the branch modules of DMOP.
- DMOP-Profile-EL: An OWL EL profile module of the DMOP ontology.

Table 4.1 displays a comparison of the size metrics for the DMOP modules. Note that merging the branches does not result in the original ontology, as can be observed from the difference in number of axioms in the merged ontology compared to the original DMOP. So if ontology developers were to independently work on the different modules for a collaboration with the hope of re-merging the modules to achieve the original ontology afterwards, it is not possible. It is apparent that the module extraction feature in Protégé extracts classes in isolation and includes the object properties, data properties, and individuals of the original ontology, including those that do not relate to the classes in the module. For instance, in the DMOP-Branch-Perdurant.owl module, the object property, solves exists. This has no dependencies to the classes in the module and should not be present.

4.1.3 Problems with existing modularisation resources

Attempting to modularise the set of ROMULUS foundational ontologies and the DMOP ontology with existing tools resulted in a number of problems. Existing tools such as Swoop and OWL module extractor resulted in modules that were too large with unnecessary entities that were preserved in the modules due to the underlying logical principals that the tools met such as local completeness and correctness. Protégé generated suitable modules, but there were slight operational problems with the tool which could be overcome by using the 'delete' option instead of the 'copy' option. While the sizes of the modules were suitable in Protégé, it did not consider

Table 4.1: Size metrics for DMOP and its related modules; OP = object property, dp = data property, and ind = individual.

Ontology	**|Class|**	**|OP|**	**|DP|**	**|Ind|**	**|Axiom|**
DMOP.owl	758	169	15	459	4584
DMOP-profile-EL.owl	758	169	15	459	4214
DMOP-branch-Endurant.owl	512	169	15	459	3409
DMOP-branch-Perdurant.owl	19	169	15	459	1376
DMOP-branch-Abstract-Quality.owl	231	169	15	459	2131
DMOP-branch-Toplevel.owl	44	169	15	459	1428
DMOP-branch-Merge.owl	758	169	15	459	4359

the entities in relation to each other, i.e., it extracts a class in isolation without considering its related object or data properties. Next, there is the problem of identifying useful metadata for modules. Modules are to be annotated with useful metadata to promote module reuse. However, for the modules in ROMULUS repository, values such as ModuleCompleteness and ModuleCoverage have to be manually calculated from each ontology module file. This is a time-consuming and tedious process. Furthermore, there is no systematic way of classifying these modules into a ModuleType and ModuleSubtype for those metadata values.

The modularity tools could be improved by taking into consideration the following:

- Considering all the different types of entities (classes, object properties, data properties, and individuals) that are dependent on a selected entity, and not just the selected entity in isolation.
- By relaxing on logical principles such as completeness and correctness to allow for the creation of smaller modules.
- By employing other modularisation techniques besides locality-based, graph partitioning, and language-based techniques.
- By having functionality for calculating values for annotating modules with useful metadata.

4.2 Evaluation metrics for modules

A number of techniques for ontology modularisation have been proposed in recent years, such as traversal methods [113], locality-based extraction [30], and partitioning [31, 35]. There also have been attempts at analysing which types of modules exist [20], and in Section 3.7.2, we determined which types of modules are useful for which purpose, such as that high-level abstraction modules are used for comprehension. There is, however, a disconnect between the two. For instance, if an ontology developer wants to reuse, say, only the branch of the 'Space' entities from the GFO module for another ontology of Spatial objects, then how does the developer know

that the module extracted from GFO is a good module? Therefore it is unclear how the quality of an ontology module could be measured. While there are few studies on evaluation ontology modules, they focus on a few of the metrics, such as size, cohesion, coupling, correctness, and completeness [124, 137, 164], and these metrics are not comprehensive enough to apply to the variety of types of modules that exist. For instance, for most of the modules that were to be generated for our two case-studies in Section 4.1, they cannot be evaluated with logical criteria such as completeness and correctness. Another problem concerning the evaluation metrics is that while many of them are described in several works, there is no formula designed to measure them. For instance, intra-module distance, to measure the distance between entities in a module.

One of the dimensions for modularity identified in Section 2.2 was the evaluation metrics, or how to measure the module. It is necessary to evaluate the ontology modularisation techniques, in terms of the quality of the generated module to determine whether a module is of good or bad quality and appropriate for a use-case. Existing studies [34, 35, 137] mention a number of techniques such as size, logical correctness, and cohesion. The evaluation metrics dimension was not included with the other dimensions in the creation of the framework for modularity as it required additional work in another direction in addition to the classification of modules performed in Section 3.6. For evaluation metrics, an investigation needs to be performed on the quality of ontology modules. For this we require a software tool which includes all the metrics for modules to automatically compute them for an ontology module as some cannot be manually computed, and we need to perform an experimental evaluation to determine which metric values correspond to which module types, i.e., how to measure if a module is of good quality.

In this section, we review and formulate existing evaluation criteria, identify new evaluation criteria, and categorise all the evaluation criteria. The evaluation metrics for modularity was compiled by studying existing literature on modularity. This resulted in 13 metrics from the literature, of which seven were short of a metric for quantitative evaluation that have now been devised (indicated with an asterisk), and three new ones have been added (indicated with a double asterisk). Earlier versions of this work on evaluation metrics has been published [82] and [87].

4.2.1 Structural criteria

Structural criteria are calculated based on the structural and hierarchical properties of the module. These criteria are calculated by inspecting the syntax of the ontology. It is usually based on counting components of the ontology such as axioms, entities, etc., and is thus a numerical value. Calculating structural criteria involves evaluating the size, relations, and placement of entities within a module. It has an impact on the maintenance, reasoning, and efficiency of the module. For maintenance, criteria such as size and relative size, which is discussed in the remainder of the section, can be used to give ontology developers an idea of the amount of maintenance that will need to be performed on the module in comparison to the original ontology. Smaller ontology modules might require less maintenance. For reasoning, criteria such as

atomic size, which measures the interdependent axioms in an ontology could give an indication of whether reasoning could be improved thanks to modularisation.

EM1: Size Size is a fairly common metric as a modularity evaluation criterion mentioned by several existing works [118, 34, 137, 35, 124]. Size refers to the number of entities in a module, $|M|$. This can be further subdivided into the number of classes $|C|$, number of object properties $|OP|$, number of data properties $|DP|$, and number of individuals $|I|$. In addition to giving the ontology developer an indication of how small or large the module is, size is also important as it is used to calculate other structural criteria which is discussed in the remainder of the section.

$$Size(M) = |M| = |C| + |OP| + |DP| + |I|. \tag{4.1}$$

EM2: Relative size** We define relative size as the size of the module, i.e., the number of classes, properties and individuals compared to the original ontology. The relative size of a module strongly influences the result of the module on tasks such as reasoning and maintenance for if the module extracted is nearly the same as the original one, then not substantial optimisation will be obtained. To compute this, we have created an equation to calculate the relative size of an ontology module M as follows.

$$Relative\ size = \frac{|M|}{|O|} \tag{4.2}$$

where $|M|$ is the size of the module and $|O|$ is the size of the ontology as described in EM1.

Example 7 *The GFO-Basic ontology module contains 47 classes, 0 individuals, 41 object properties, and 0 data properties. The source ontology, GFO, contains 78 classes, 0 individuals, 67 object properties, and 0 data properties. Hence the relative size is* $\frac{47+0+41+0}{78+0+67+0} = 0.61\%$.

EM3: Appropriateness of module size The appropriateness can be specified by mapping the size of an ontology module to some appropriateness values. Schlicht and Stuckenschmidt [137] propose an appropriate function to measure this, which ranges between 0 and 1, where a module with an optimal size has a value of 1. The function they propose is based on software design principles: since the optimal size of software modules is between 200-300 logical lines of software code, an axiom value of 250 would be the optimal size for an ontology, restricting the module to be between 0 and 500 axioms. The appropriateness equation is defined as follows [137].

$$Appropriate(x) = \frac{1}{2} - \frac{1}{2}cos(x.\frac{\pi}{250}) \tag{4.3}$$

where x is the number of axioms in the module.

Example 8 *The Temporal Relations module of the ExtendedDnS descriptions and situations ontology has 435 axioms. Therefore, its appropriate size value is* $\frac{1}{2} -$

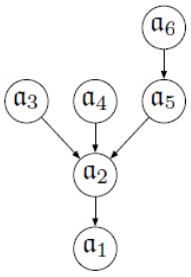
$\frac{1}{2}cos(435.\frac{\pi}{250}) = 0.16$. *The appropriate size value of the Temporal Relations module is rather low, due to the fact that the module has many more axioms than the optimal of 250 as defined by Schlicht and Stuckenschmidt.*

EM4: Atomic Size** The notion of atoms within ontology modules was first introduced by Del Vescovo et al. [156] in a study of BioPortal repository ontologies [163]. An atom is a group of axioms within an ontology that have dependencies between each other. Based on the findings from the study [163], that it is possible to modularise an ontology using atomic decomposition as a method, we propose to measure the size of atoms in ontologies. We can gain insight on whether an ontology may be easily decomposed using logical modularisation methods. We define the atomic size as the average size of a group of inter-dependent axioms in a module. We formulate an equation to measure the atomic size of a module by using the number of atoms and number of axioms present in the module.

$$Atomic\ Size(M) = \frac{|Axiom|}{|Atom|} \tag{4.4}$$

Example 9 *Consider the example in the screenshot (Figure 4.2) of an atomic decomposition [157]. The number of atoms in the example is 6 and there are 7 axioms in total. The atomic size is hence* $\frac{7}{6} = 1.17$. *This tells us that there is an average of 1.17 axioms per atom for the example.*

α_1 = 'Animal $\sqsubseteq$ (= 1hasGender.$\top$)',
α_2 = 'Animal $\sqsubseteq$ ($\geq$ 1hasHabitat.$\top$)',
α_3 = 'Person $\sqsubseteq$ Animal',
α_4 = 'Vegan $\equiv$ Person $\sqcap \forall$eats.(Vegetable $\sqcup$ Mushroom)',
α_5 = 'TeeTotaller $\equiv$ Person $\sqcap \forall$drinks.NonAlcoholicThing',
α_6 = 'Student $\sqsubseteq$ Person $\sqcap \exists$hasHabitat.University',
α_7 = 'GraduateStudent $\equiv$ Student $\sqcap \exists$hasDegree.{BA, BS}'



Here the $\perp$-atoms in the AD contain the following axioms respectively: $\mathfrak{a}_1 = \{\alpha_1, \alpha_2\}$, $\mathfrak{a}_2 = \{\alpha_3\}$, $\mathfrak{a}_3 = \{\alpha_4\}$, $\mathfrak{a}_4 = \{\alpha_5\}$, $\mathfrak{a}_5 = \{\alpha_6\}$, $\mathfrak{a}_6 = \{\alpha_7\}$.

Figure 4.2: An ontology's atomic decomposition. See example 9 for details. Source: [157].

EM5: Intra-module distance* d'Aquin et al. define the intra-module distance in a module as the distance between entities in a module [35]. It is calculated by counting the number of relations in the shortest path from one entity to the other, for every entity in the module.

Based on this definition, we formulate an equation to measure the intra-module distance of a module that considers the distance between an entity to another in terms of shortest-path relations. For measuring this distance, we use Freeman's Farness

value [43]. In the field of network centrality, Freeman's Farness value of a node is described as the sum of its distances to all other nodes in the network.

$$\textit{Intra-module distance}(M) = \sum_{i}^{n} \textit{Farness}(i) \tag{4.5}$$

where n is the number of nodes in the module, and Freeman's Farness value is defined as follows:

$$\textit{Farness}(i) = \sum_{j}^{n} \textit{distance}_{ij} \tag{4.6}$$

where i and j are two entities in the module.

The intra-module distance criterion is used in the calculation of the relative intra-module distance criterion which is described in the following section.

Example 10 *Consider the graphical notation of the source ontology in Figure 4.3. We only consider the distances between entities A, B, C, D, E, for the source ontology since those exist in the resulting module. We calculate the farness value for each node in the ontology as displayed in Table 4.2. As observed, the Farness for entity A in ontology O, Farness(A) is 1, 1, 4, and 4 between entities B, C, D, and E respectively. The values are all aggregated to calculate the intra-module distance of the ontology. Finally, the intra-module distance value for the ontology O = 32. Similarly, the intra-module distance for the module M is 16.*

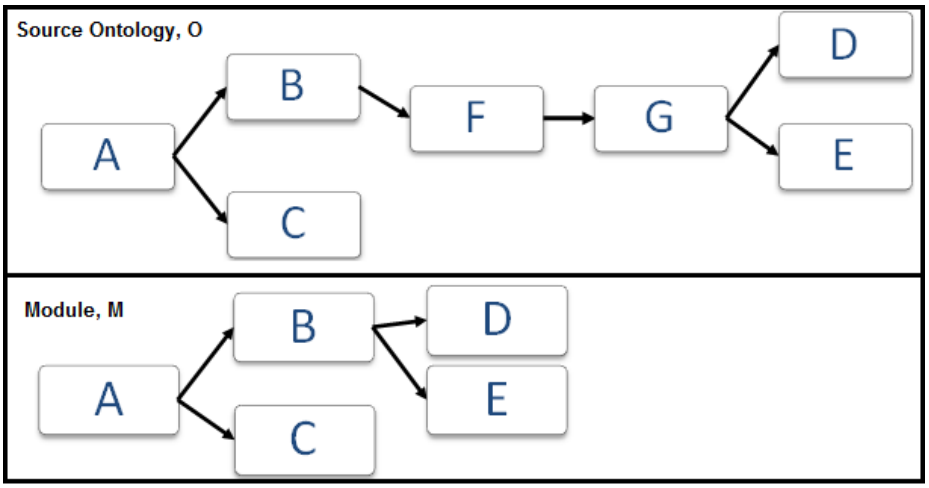


Figure 4.3: A source ontology and corresponding module for which we calculate intra-module distance. The arrows between the entities indicate the shortest-path relation between them.

EM6: Relative Intra-module distance** We define the relative intra-module distance of a module as the difference between distances of entities in a module M to a source ontology O. This difference would reveal if the overall distance between

Table 4.2: The farness values for each entity of the source ontology and corresponding module. See example 10 and Figure 4.3

	Farness, O							**Farness,** M					
	A	**B**	**C**	**D**	**E**	**Sum**		**A**	**B**	**C**	**D**	**E**	**Sum**
A	-	1	1	4	4	10	**A**	-	1	1	2	2	6
B	1	-	0	3	3	7	**B**	1	-	0	1	1	3
C	1	0	-	0	0	1	**C**	1	0	-	0	0	1
D	4	3	0	-	0	7	**D**	2	1	0	-	0	3
E	4	3	0	0	-	7	**E**	2	1	0	0	-	3
Intra-module distance(O) =						32	*Intra-module distance(M) =*						16

the entities in the module has been reduced, and by how many distance units. This is useful in comparing the difference in module size; whether the technique reduces the size considerably. To compare the distances of the original ontology, we compute the farness values for the subset of nodes that exist in a module, which is used to calculate the intra-module distance of the original ontology, and is defined as follows:

$$\textit{Relative intra-module distance(M)} = \frac{\textit{Intra-module distance(O)}}{\textit{Intra-module distance(M)}} \tag{4.7}$$

Example 11 *Recall in example 10, we calculated the intra-module distance of the source ontology O to be 32 and of the module M to be 16 (see Table 4.2). The relative intra-module distance is* $\frac{32}{16} = 2$*, hence the module entities are twice as close as the original ontology.*

EM7: Cohesion Cohesion refers to the extent to which entities in a module are related to each other. Several works describe ontology cohesion as a set of metrics to measure the modular relatedness of ontologies using different formula [48, 115, 118, 164]. In order to accurately measure the cohesiveness for ontology modules, we use a metric defined by Oh et al. [118]. Cohesion metrics provided by others were not suitable as they were generalised for ontologies and not towards modules [164], or too restrictive as they only considered relations containing strong and moderate dependencies between entities, and did not consider relations that are neither strong nor moderate [39]. This is measured by calculating the sum of the strength of relations divided by the number of all possible relations in a module M. Cohesion values are between the range of 0 and 1, where values close to 0 describe modules with entities that have few or no relations, and value close to 1 describe modules with entities that have many relations.

$$Cohesion(M) = \begin{cases} \sum_{C_i \in M} \sum_{C_j \in M} \frac{SR(c_i,c_j)}{|M|(|M|-1)} & if |M| > 1 \\ 1 & otherwise \end{cases} \tag{4.8}$$

where $|M|$ is the number of entities in the module as described in EM1. The product of $|M|(|M|-1)$ represents the number of possible relations between entities

in M. The strength of relation for each entity is calculated based on the farness centrality measure for graph theory proposed by Freeman [43] from equation 4.6.

$$SR(c_i, c_j) = \begin{cases} \frac{1}{farness(i)} & if\ relations\ exist\ between\ c_i\ and\ c_j \\ 0 & otherwise \end{cases} \tag{4.9}$$

Example 12 *Consider module M, from Figure 4.3. In Table 4.3, we have the values for farness and the inverse of farness which is used to calculate the cohesion. The sum of all the 1/farness values is 10. Hence the cohesion value is as follows: Cohesion* $= \frac{10}{5(4)} = 0.5$.

Table 4.3: The farness and strength of relation (1/farness) values for each entity of the ontology.

	Farness, *M*							**1/farness,** *M*					
	A	**B**	**C**	**D**	**E**	**Sum**		**A**	**B**	**C**	**D**	**E**	**Sum**
A	-	1	1	2	2	10	**A**	-	1/1	1/1	1/2	1/2	3
B	1	-	0	1	1	7	**B**	1/1	-	0	1/1	1/1	3
C	1	0	-	0	0	1	**C**	1/1	0	-	0	0	1
D	2	1	0	-	0	7	**D**	1/2	1/1	0	-	0	1.5
E	2	1	0	0	-	7	**E**	1/2	1/1	0	0	-	1.5
Cohesion (M) = 0.5													

4.2.2 Logical Criteria

The Web Ontology Language (OWL) is based on a decidable fragment of first order predicate logic. As such, it is possible to evaluate ontology modules by the logical criteria that they hold.

EM8: Correctness* Correctness states that every axiom that exists in the module also exists in the original ontology and that nothing new should be added to the module. Several works mention the logical correctness criterion [31, 35, 98, 124]. Correctness was proposed as a condition [31] or requirement [98] for logical modules, and later on as an evaluation criteria [35]. There is no mathematical equation to measure it hence we formulate it as follows:

$$Correctness(M) = \begin{cases} true & if Axioms(M) \subseteq Axioms(O) \\ false & otherwise \end{cases} \tag{4.10}$$

Example 13 *The GFO-Abstract-Top ontology is a subset of the GFO ontology. No new axioms have been added to the GFO-Abstract-Top ontology; it only contains those axioms which exist in the GFO ontology. Thus, the GFO-Abstract-Top ontology is logically correct. The GFO-Basic ontology, however, is a smaller module based on the GFO ontology but it also contains new axioms that do not exist in the GFO ontology.*

For instance, the entity Processual Structure *exists in the GFO-Basic module but not in the source ontology, GFO. Thus, the logical correctness property does not hold for the GFO-Basic module.*

EM9: Completeness* A module is logically complete if the meaning of every entity is preserved as in the source ontology. The completeness property evaluates that for a given set of entities or signature, every axiom that is relevant to the entity as in the source ontology is preserved in the module. Like correctness, several works mention the logical completeness criterion [31, 35, 98, 124]. Completeness was proposed as a condition [31] or requirement [98] for logical modules, and later on as an evaluation criteria [35]. There is no mathematical equation to measure it hence we formulate it as follows:

$$Completeness(M) = \begin{cases} true & if \sum_{i}^{n} Axioms(Entity_i(M)) \models Axioms(Entity_i(O)) \\ false & otherwise \end{cases} \tag{4.11}$$

Example 14 *In the source ontology, DOLCE the* endurant *entity is defined as follows:*

- *endurant* $\sqsubseteq \forall$ *part.endurant*
- *endurant* $\sqsubseteq$ *spatio-temporal-particular*
- *endurant* $\sqsubseteq \exists$ *participant-in.perdurant*
- *endurant* $\sqsubseteq \forall$ *specific-constant-constituent.endurant*
- *endurant* $\sqsubseteq \neg$ *quality*
- *endurant* $\sqsubseteq \neg$ *perdurant*
- *endurant* $\sqsubseteq \neg$ *abstract*

If DOLCE were to be modularised to create a branch module, containing only the branch of Endurant entities, DOLCE-endurants, the endurant *entity is defined as follows:*

- *endurant* $\sqsubseteq \forall$ *part.endurant*
- *endurant* $\sqsubseteq$ *spatio-temporal-particular*
- *endurant* $\sqsubseteq \exists$ *participant-in.perdurant*

The meaning of the endurant *entity was not preserved in the module since the axiom* endurant $\sqsubseteq \forall$ specific-constant-constituent.endurant *existed in the original ontology but not in the module. Therefore the DOLCE-endurants module has a false value for the completeness metric.*

4.2.3 Relational criteria

Relational criteria deal with the relations and behaviour that modules exhibit with other modules in a system of interrelated modules.

EM10: Inter-module distance* The inter-module distance in a set of modules has been defined as the number of modules that have to be considered to relate two entities [34, 35]. Based on this definition, we have created an equation to measure the inter-module distance of a network of modules.

$$\textit{Inter-module distance} = \begin{cases} \sum\limits_{C_i,C_j \in (M_i,,M_n)} \frac{NM(C_i,C_j)}{|(M_i,..,M_n)|(|(M_i,..,M_n)|-1)} & |(M_i,..,M_n)| > 1 \\ 1 & otherwise \end{cases} \tag{4.12}$$

where $NM(C_i, C_j)$ is the number of modules to consider to relate entities i and j. The product of $|(M_i,..,M_n)|(|(M_i,..,M_n)|-1)$ represents the number of possible relations between entities in a set of modules $M_i,.,M_n$.

Example 15 *Consider the two sets of inter-related modules S1 and S2 in Figure 4.4. For each entity pair in S1, we have the number of modules, NM, that have to be considered to relate them in Table 4.4. The sum of NM for S1 is 313. The number of entities in S1 is 12, hence the* $|(M_i,..,M_n)|(|(M_i,..,M_n)|-1)$ *value = 12(11). Thus the inter-module distance for S1 is* $\frac{313}{12(11)} = 2.37$*. For S1, it takes 2.37 modules to relate two entities in the set.*

In S2, some of the entities that exist in S1 have been removed, notably entities J, K, and L. For each entity pair in S2, we have the number of modules, NM, that have to be considered to relate them in Table 4.5 The sum of NM for S2 is 126. The number of entities in S2 is 9, hence the $|(M_i,..,M_n)|(|(M_i,..,M_n)|-1)$ *value = 9(8). Thus the inter-module distance for S2 is* $\frac{112}{9(8)} = 1.75$*. For S2, it takes 1.75 modules to relate two entities in the set.*

Thus, it is apparent that the inter-module distance for S1 is larger than that of S2; the entities in S1 are farther away than in S2.

EM11: Coupling* Coupling has been defined in several works as a measure of the degree of interdependence of a module [48, 118, 115, 119]. The coupling value is high if entities in a module have strong relations to entities in other modules; it is difficult to modify and update such modules independently because they affect other modules in the system. For instance, in the EDAM bioinformatics ontology [70], the object property is_format_of has as domain the class Format and range Data. When it was partitioned with SWOOP, the Data class was present in the first partition while Format and is_format_of were present in the third partition, and the different modules are linked together with ε-connections. Changes to the Data class in the first partition could affect the third partition since it is linked to entities in the third partition.

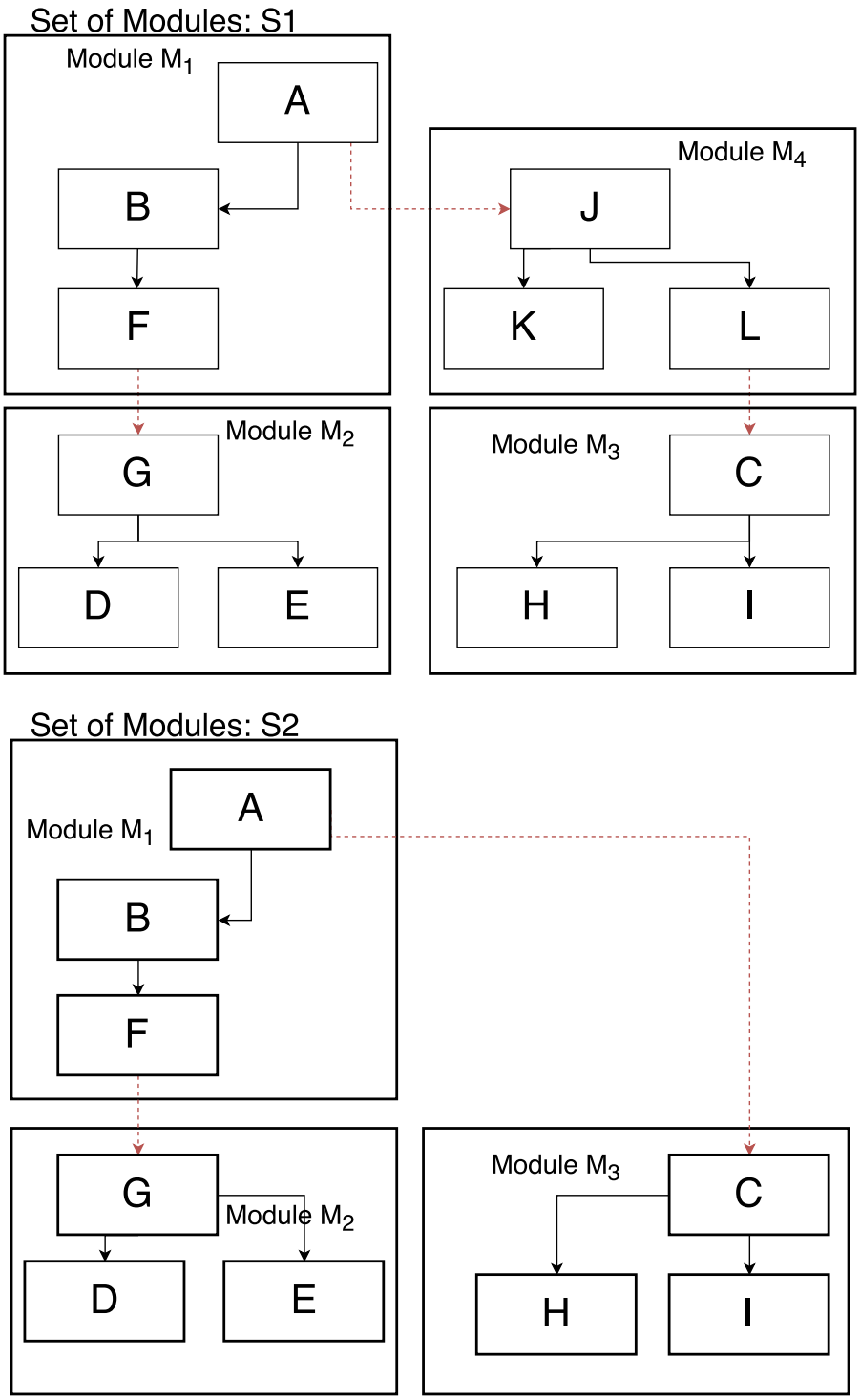


Figure 4.4: Two sets of modules S1, S2 with inter-related links. The plain arrow links between entities denote relations between entities in the same module while the dotted arrow links denote relations between entities in different modules.

Table 4.4: The number of modules, *NM*, that have to be considered to relate two entities in the set of modules (S1).

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**Sum**
A	-	1	3	2	2	1	2	3	3	2	2	2	23
B	1	-	3	2	2	1	2	3	3	2	2	2	23
C	3	3	-	4	4	3	4	1	1	2	2	2	29
D	2	2	4	-	1	2	1	4	4	3	3	3	29
E	2	2	4	1	-	2	1	4	4	3	3	3	29
F	1	1	3	2	2	-	2	3	3	2	2	2	23
G	2	2	4	1	1	2	-	4	4	3	3	3	29
H	3	3	1	4	4	3	4	-	1	2	2	2	29
I	3	3	1	4	4	3	4	1	-	2	2	2	30
J	2	2	2	3	3	2	3	2	2	-	1	1	23
K	2	2	2	3	3	2	3	2	2	1	-	1	23
L	2	2	2	3	3	2	3	2	2	1	1	-	23
$NM(C_i, C_j) =$													313

Table 4.5: The number of modules, *NM*, that have to be considered to relate two entities in the set of modules (S2).

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**Sum**
A	-	1	2	2	2	1	2	2	2	14
B	1	-	2	2	2	1	2	2	2	14
C	2	2	-	2	2	2	2	1	1	14
D	2	2	2	-	1	2	1	2	2	14
E	2	2	2	1	-	2	1	2	2	14
F	1	1	2	2	2	-	2	2	2	14
G	2	2	2	1	1	2	-	2	2	14
H	2	2	1	2	2	2	2	-	1	14
I	2	2	1	2	2	2	2	1	-	14
$NM(C_i, C_j)$										126

Table 4.6: The number of external links, *NEL*, that have to be considered to relate M1 to other modules in the system.

	M1	**M2**	**M3**
M1	-	1	1

To measure the coupling of a module, we define our own measure as a ratio of the number of external links between a module M_i and M_j, NEL_{M_i,M_j} for n modules in a system to every possible external link between a module M_i and M_j in a system.

$$Coupling(M_i) = \begin{cases} \sum_{i=0}^{n} \sum_{\substack{j=0 \\ i \neq j}}^{n} \frac{NEL_{M_i,M_j}}{|M_i||M_j|} & NEL_{M_i,M_j} > 0 \\ 0 & otherwise \end{cases} \tag{4.13}$$

where $|M_i|$ is the number of entities in the current module and $|M_j|$ is the number of entities in a related module in the set of n modules.

Example 16 *Consider module M_1 from the S2 set of inter-related modules in Figure 4.4. We have tabulated the number of external links that have to be considered to relate M_1 to other modules in the set, $NEL_{M1,Mi}$. The number of possible external link between a module M_1 and the other modules in the system is calculated as follows: $|M_1|(|M_2|) + |M_1|(|M_3|) = 18$. Hence the coupling($M_1$) $= \frac{2}{18} = 0.11$ which indicates a low interdependence toward other modules in the system.*

EM12: Redundancy Redundancy has been defined as the duplication of axioms within a set of ontology modules [137]. When a large ontology is partitioned into smaller modules, there are sometimes modules that overlap with regard to shared knowledge. Thus axioms exist in more than one modules. Sometimes this is required for robustness or efficiency. However, these redundant axioms cause difficulty in maintaining the consistency of the modules when modules are to be updated. For instance, consider that the same class Bread exists in two or more related modules in an ontology describing the domain of food science. In recent years, there has been a shift towards Gluten-free foods, therefore, the axiom $\mathsf{Bread} \sqsubseteq \exists \mathsf{hasProtein.Gluten}$ has now been removed from one of the modules. The class definitions for the Bread class may differ in the various modules causing inconsistency in the system, and each of the other modules containing the Bread class must be altered to remove the axiom.

To measure redundancy in a set of modules, we use Schlicht and Stuckenschmidt's equation [137]; the level of redundancy in a set of n modules is calculated as the fraction of duplicated axioms as follows:

$$Redundancy = \frac{(\sum_{i=1}^{k} n_i) - n}{\sum_{i=1}^{k} n_i} \tag{4.14}$$

where $\sum_{i=1}^{k} n_i$ is the total number of axioms and n is the number of distinct axioms in a module. The resulting fraction is a value of redundancy.

Example 17 *Consider the class declarations and axioms in the set of modules with no inter-related links that have been partitioned from a food ontology. There are 3 ontology modules: Fruit, Vegetable, and Meat, described below. Axioms that have been repeated more than once (redundant axioms) are shown in bold font.*

Fruit module

- *EdiblePlant*
- *Fruit*
- *Flavour*
- *SweetFlavour*
- *hasFlavour*
- *Fruit* $\sqsubseteq$ *EdiblePlant*
- *Fruit* $\sqsubseteq \exists$ *hasFlavour.SweetFlavour*

Vegetable module

- **EdiblePlant**
- *Vegetable*
- **Flavour**
- *SavouryFlavour*
- **hasFlavour**
- *Vegetable* $\sqsubseteq$ *EdiblePlant*
- *Vegetable* $\sqsubseteq \exists$ *hasFlavour.SavouryFlavour*

Meat module

- *Meat*
- *RedMeat*
- **Flavour**
- **SavouryFlavour**
- **hasFlavour**

- *RedMeat* $\sqsubseteq$ *Meat*
- *Meat* $\sqsubseteq \exists$ *hasFlavour.SavouryFlavour*

From the three modules, there is a total of 21 axioms, i.e., the Ax_t *value is 21. There are 15 distinct axioms that exist in the set of modules (these axioms exist at most once and are those that are not in bold font), hence* Ax_d *is 15. The redundancy of the set of partitioned modules is thus* $\frac{21-15}{21} = 0.29$*. Hence, 29% of the axioms in the set of modules are redundant.*

4.2.4 Information hiding

Ontology modules sometimes deal with hiding aspects of the source ontology from the module for privacy and simplification reasons. Information hiding within modules assesses whether the module encapsulates all the information in the module such that the privacy is preserved for each module. We formulate the following criteria to measure information hiding properties of an ontology module.

EM13: Encapsulation* d'Aquin et al. mention encapsulation with the notion that "a module can be easily exchanged for another, or internally modified, without side-effects on the application can be a good indication of the quality of the module" [35]. This general idea seems potentially useful for semantic interoperability. There are two components to d'Aquin et al.'s encapsulation:

- 'Swappability' of a module, which increases with fewer links to entities in another module in an ontology network; e.g., one can interchange their domain ontologies between foundational ontologies using the SUGOI tool [81].
- Casting it into a measure of knowledge preservation within the given module.

We have designed an equation to calculate the encapsulation of a module in a given a set of modules. For a module, with $n-1$ related , this is measured using the number of axioms in the given module $|Ax_i|$ and the number of axioms that occur in both the given module and related modules, $|Ax_{ij}|$.

$$Encapsulation(M\_i) = 1 - \frac{\sum_{j=1}^{n-1} \frac{|Ax_{ij}|}{|Ax_i|}}{n} \tag{4.15}$$

Encapsulation values in modules that are equal or close to 1 indicate a good value; all or most of the knowledge has been encapsulated. Conversely, values that are equal to or close to 0 indicate a poor encapsulation value; none or very little of the knowledge has been encapsulated, and privacy has not been preserved.

Example 18 *Consider the 3 ontology modules Fruit, Vegetable, and Meat, from Example 17. We calculate the encapsulation of the Fruit module as follows. There are 7 axioms in the Fruit module, i.e., the* $Ax_i = 7$*. In the Vegetable module, there are*

3 overlapping axioms, i.e., they also exist in the Fruit module. In the Meat module, there are 2 overlapping axioms, i.e., they also exist in the Fruit module. Hence, the Encapsulation(Fruit) is calculated as $1 - \frac{\frac{3}{7}+\frac{2}{7}}{3} = 0.76$*. Thus, 0.76 (76%), or a large amount of the domain knowledge is encapsulated in the Fruit module but the complete privacy of the Fruit module is not preserved.*

EM14: Independence* Independence evaluates whether a module is self-contained and can be updated and reused separately. In this way, ontology modules can evolve independently. Thus, the semantics of the entire ontology could change without the need for all the modules to be changed. For instance, for the set of Gist foundational ontology modules [103], if information about physical things need to be updated, the relevant module gistPhysicalThing could be updated without needing to alter the remaining modules. In order to determine whether a module is independent, we use the encapsulation and the coupling equations. A module is set to be independent if it has an encapsulation value of 1 and a coupling value of 0. This can be checked using the following code rule.

$$Ind(M\_i) = \begin{cases} true & Encapsulation(M_i) = 1 \text{ and } Coupling(M_i) = 0 \\ false & otherwise \end{cases} \tag{4.16}$$

where $|M_i|$ is the number of entities in the current module and $|M_j|$ is the number of entities in a related module in the set of n modules.

Example 19 *Consider the 3 ontology modules in Example 17. We have already worked out the encapsulation value for the Fruit module in Example 18 as 0.76. There are no inter-related links between the modules hence the coupling value is 0. Since the encapsulation value is not 1, the conditions for independence do not hold for the Fruit module hence it is not independent.*

4.2.5 Richness criteria

The richness or amount of information in an ontology is designed as one aspect to measure the quality of an ontology. For modules, this is important to measure in cases where abstraction is employed to compare the granularity of the source ontology to that of the module. Tartir et al. [151] propose measurable richness schema metrics for these which we describe in this section.

EM15: Attribute richness Attribute richness is defined as the average number of attributes per class [151]; i.e., each class is defined by a number of axioms with properties describing it, which is referred to as attributes.

$$AR(M) = \frac{|att|}{|C|} \tag{4.17}$$

where *att* is measured by the number of data properties in the module $|DP|$ and $|C|$ is the number of classes in the module. In an ontology, an attribute is used to describe an entity and each attribute, or data type, has a name and value.

Example 20 *The pizza ontology has no data properties (attributes) defined. The AR value is 0, therefore there is no attribute richness in the pizza ontology.*

EM16: Inheritance richness The formal definition of the inheritance richness of an ontology is the average number of subclasses per class, and its formula follows.

$$IR_S(M) = \frac{\sum_{C_i \in C} |H^C(C_1, C_i)|}{|C|} \tag{4.18}$$

where $|H^C(C_1, C_i)|$ is the number of subclasses per class and $|C|$ is the total number of classes in the ontology.

Example 21 *Refer back to ontology O and module M from Figure 4.3. For module M, the entities which have subclasses are entity A with 2 subclasses, and entity B with 2 subclasses. Hence the $|H^C(C_1, C_i)|$ value is $2 + 2 = 4$. There are 5 classes in total in M. The IR_S value for M is thus $\frac{4}{5} = 0.8$. Using the same method we work out the IR_S value for ontology O, which is $\frac{6}{7} = 0.85$.*

A summary of each evaluation metric, its value range, and good values where applicable is shown in Table 4.7. To assist with the lack of evaluation metrics and corresponding formulae in ontology modules, we presented a comprehensive list of metrics in the dimensions for modularity in Section 4.2. The problem at hand now is that it is unclear which metrics should be used for which module types. Metrics such as size do not fare well with modules created using locality-based techniques (recall the BFO-Occurrent module that was 92% of the size of the original BFO ontology because the existing tools which used locality-based techniques), while completeness and correctness do not measure well with partition-based modules [124]. This could mean that only specific metrics must be used to acquire meaningful results about the quality of an ontology module, based on the nature of the module. To solve this problem, we developed software support, Tool for Ontology Modularity Metrics (TOMM), to apply them to modules to measure these metrics and determine which metric values correspond to which module types, i.e., how to measure if a module is of good quality.

In the following Section, we introduce the TOMM tool. Thereafter we perform an experimental evaluation using TOMM and a set of modules to determine how to measure the quality of a module. TOMM is then evaluated with use-cases. This work has been published in [87]

4.2.6 Tool for ontology module metrics

In order to uncover information about how evaluation metrics relate to ontology modules, we have created a Tool for Ontology Modularity Metrics (TOMM). TOMM is

Table 4.7: A summary of the set of evaluation metrics with their expected value range and values that are considered good. For the good values, we use a 4-point scale of small (0-0.25), medium (0.25-0.5), moderate (0.51-0.75), and large (0.75-1), and true/false values.

Evaluation metric	**Value range**	**Value type**	**Good value**
EM1: Size	$i \geq 0$	integer	-
EM2: Relative size	$1 \geq i \geq 0$	decimal	small to medium
EM3: Appropriateness	$1 \geq i \geq 0$	decimal	large
EM4: Atomic size	$1 \geq i \geq 0$	decimal	-
EM5: Intra-module distance	$i \geq 0$	decimal	-
EM6: Relative intra-module distance	$i \geq 0$	integer	-
EM7: Cohesion	$1 \geq i \geq 0$	decimal	small
EM8: Correctness	true or false	boolean	true
EM9: Completness	true or false	boolean	true
EM10: Inter-module distance	$i \geq 0$	decimal	-
EM11: Coupling	$i \geq 0$	decimal	small
EM12: Redundancy	$1 \geq i \geq 0$	decimal	small to medium
EM13: Encapsulation	$1 \geq i \geq 0$	decimal	large
EM14: Independence	true or false	boolean	true
EM15: Attribute richness	$i \geq 0$	decimal	-
EM16: Inheritance richness	$i \geq 0$	decimal	-

programmed with all the defined equations describing the new and existing evaluation metrics discussed in Sections 4.2.1 to 4.2.5.

TOMM is a stand-alone Java application and can be downloaded from `http://www.thezfiles.co.za/Modularity/TOMM.zip`. TOMM allows one to upload a module or set of related ontology modules, together with an original ontology (if it exists), and then it computes metrics for the module/s. A screenshot of TOMM's interface is shown in Fig. 4.5. The metrics are saved as a log file on the user's computer as shown in Figure 4.6.

The metrics in the TOMM software tool are grouped into three different parts, depending on what the developer needs to measure. There are 1) metrics for a single module, for when the user wishes to evaluate a module on its own, 2) metrics for a set of related modules, for when the user wants to evaluate a set of modules that are related to each other, and 3) metrics for an original ontology, for when the user wants to evaluate a module or set of modules together wish a corresponding source ontology. Metrics for a single module contain the following evaluation metrics: EM2, EM6, EM8, and EM9. Metrics for a set of related modules contain the following evaluation metrics: EM11- EM14. Metrics for an original ontology contain the following evaluation metrics: EM1, EM3- EM5, EM7, EM15, and EM16.

By evaluating the modules with TOMM, we will be able to determine which evaluation metrics can be used to measure which module types, and which will lead

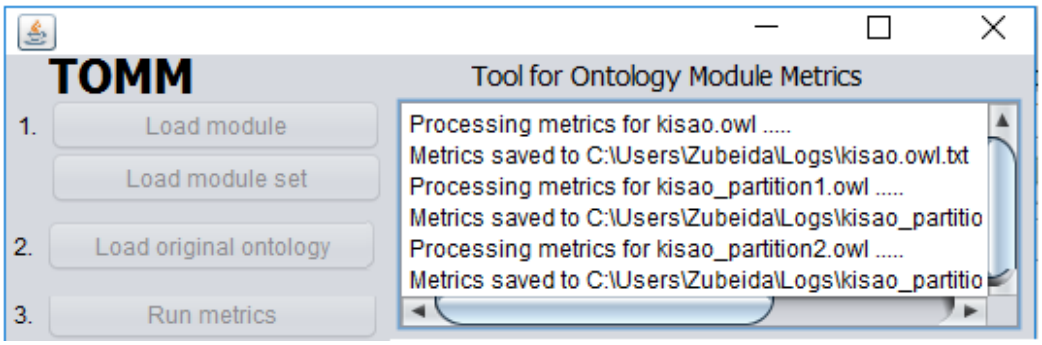


Figure 4.5: The interface of TOMM.

```
Log file for kisao_partition1.owl

Metrics for kisao_partition1.owl
No. of classes in ontology: 125
No. of OP in ontology: 6
No. of DP in ontology: 2
No. of Ind in ontology: 15
Size of ontology: 148
Atomic size of module: 4.263513513513513
No. of axioms in ontology: 327
Appropriateness of ontology: 0.7836344745633782
Intra module distance: 1493.5
Cohesion of ontology: 0.016505025433596783
Attribute richness of ontology: 0.016
Inheritance richness of ontology: 0.072
Encapsulation of ontology 0.9871559633027522
Coupling of ontology 1.7322016282695306E-4
Is the ontology independent? false
Log file for relative metrics for kisao_partition1.owl

Relative Metrics for kisao_partition1.owl compared to
kisao.owl
Relative size of module: 0.5873015873015873
Relative intra module distance of module: 22.45831938399732
The module is not logically correct. The following axiom
exists in the module but not in the original ontology:
ClassAssertion(owl:ForeignClass
<http://www.biomodels.net/kisao/KISAO#KISAO_0000418>)
The module is not logically complete. The meaning of the
entity: <http://www.biomodels.net/kisao/KISAO#KISAO_0000219>
is not preserved in the module as it is in the source
ontology.
Time taken for processing: 0.422 seconds, 0.007033333333333333
minutes, 1.1722222222222221E-4 hours.
```

Figure 4.6: A log file for the kisao-partition ontology module generated by TOMM.

to uncovering the problem of evaluating a module to determine whether it is of good quality.

4.2.7 Experimental evaluation

The purpose of the experiment is to evaluate modules with a set of metrics using TOMM to determine which metrics can be used to evaluate which module types and how to tell if a module is of good quality. We expect that the results will determine how the metrics of a module relate to other factors, such as technique to create them. Once the dependencies have been created between a module's type and evaluation metrics, we randomly selected two ontology modules to evaluate with TOMM and determine whether they are of good quality.

4.2.7.1 Materials and methods

The method for the experiment is straightforward:

1. Collect a set of ontology modules.
2. Run the TOMM tool for each module.
3. Conduct an analysis from the evaluation results for each module.

The materials used for the experiment were as follows: Protégé v4.3 [110], TOMM, and a set of 189 ontology modules previously used in the experiment on classifying modules in Section 3.6 that were collected from ontology repositories that serves as the training set. This set contains modules of 14 different types, which are summarised in Appendix A. All the test files used for this experimental evaluation can be downloaded from `www.thezfiles.co.za/Modules/testfiles.zip`.

4.2.7.2 Results

We ran TOMM for each of the 189 modules in order to discover potential relationships between the different modules and their metrics. Metrics were successfully generated for 188 modules. The 'FMA_subset' module (from T12: weighted abstraction modules) was too large for TOMM to process due to insufficient Java heap space size and increasing the parameters caused the machine to crash. We are looking at running TOMM on a High-Performance Computing Cluster in the future. The metrics values for each module type are displayed in Tables 4.8- 4.10, and we discuss them here.

For size, T7 (ontology matching) modules are very small, only 2% compared to the original ontology. T2 (subject domain) could not be evaluated with the relative size metric as there were no original ontologies. T13 (expressiveness sub-language) is as large as the original ontology. For appropriateness, T10 (entity type abstraction) is the most appropriate at 0.99, meaning that most of the modules have between 200-300 axioms.

Table 4.8: Averages for the structural metrics of the set of modules.

Type	No. of modules	Size	Relative Size	No. of axioms	Appropr-iateness	Atomic size	Intra module distance	Relative intra-module distance	Cohesion
T1	13	11.08	0.10	410.00	0.38	5.35	17.00	20.69	0.04
T2	42	125.62	-	409.19	0.64	5.18	16080.50	-	0.03
T3	7	85.43	0.79	367.86	0.24	6.31	8595.00	0.99	0.09
T4	3	29.00	0.34	261.67	0.47	10.10	853.33	63.65	0.09
T5	2	33.50	0.30	168.50	0.61	7.20	714.00	1.04	0.11
T6	10	417.20	0.21	922.5	0.49	3.17	504773.90	0.03	0.13
T7	90	2.26	0.02	14.02	0.009	1.33	0.97	2.48	0.15
T8	4	844.75	0.60	2166.75	-	3.77	163319.00	1.03	0.01
T9	1	94.00	1.00	884.00	-	2.89	12322.00	-	0.07
T10	1	103.00	0.56	257.00	0.99	4.21	23596.00	1.04	0.07
T11	3	279.67	0.51	715.67	0.89	3.72	1767.67	0.88	0.01
T12	3	158.00	0.41	582.00	0.02	5.84	23304.30	1.93	0.03
T13	6	305.50	1.00	1019.17	0.46	4.35	233449.00	1.00	0.02
T14	1	1360.00	0.97	4369.00	-	5.57	1396298.00	1.00	0.02

Table 4.9: Medians for the structural metrics of the set of modules.

Type	No. of modules	Size	Relative Size	No. of axioms	Appropriateness	Atomic size	Intra module distance	Relative intra-module distance	Cohesion
T1	13	10.00	0.02	115.00	0.34	5.50	5.00	20.69	0.005
T2	42	65.50	-	221.00	0.64	5.31	13.00	-	0.005
T3	7	97.00	0.90	444.00	0.11	6.31	11714.00	1.00	0.06
T4	3	35.00	0.02	211.00	0.47	5.00	1004.00	63.66	0.09
T5	2	33.50	0.30	168.50	0.61	7.20	714.00	1.04	0.11
T6	10	251.00	0.17	519.00	0.30	2.99	168705.00	0.00	0.11
T7	90	2.00	0.01	14.00	0.007	1.00	0.00	0.00	0.00
T8	4	781.00	0.56	1941.00	-	3.64	41582.5	1.02	0.01
T9	1	94.00	1.00	884.00	-	2.89	12322.00	0	0.07
T10	1	103.00	0.56	257.00	0.99	4.21	23596.00	1.03	0.07
T11	3	80.00	0.49	212.00	0.89	3.77	298.00	1.00	0.01
T12	3	99.00	0.42	580.00	0.02	5.87	3631.00	2.17	0.03
T13	6	90.00	1.00	393.00	0.38	4.33	473.50	1.00	0.01
T14	1	1360.00	0.97	4369.00	-	5.65	1396298.00	1.00	0.02

Table 4.10: Average, median, and boolean values for the logical, richness, information hiding, and relational criteria.

	Logical criteria				**Richness criteria**			
Type	**Correctness**		**Completeness**		**Attribute richness**		**Inheritance richness**	
	True	**False**	**True**	**False**	**Average**	**Median**	**Average**	**Median**
T1	0%	100%	100%	0%	0.83	0.67	1.48	1.00
T2	-	-	-	-	1.45	1.27	2.37	1.94
T3	29%	71%	0%	100%	0.84	0.92	2.30	2.42
T4	100%	0%	33%	67%	3.61	1.47	1.79	1.40
T5	0%	100%	0%	100%	0.87	0.87	2.45	2.45
T6	60%	40%	60%	40%	0.10	0.00	54.32	4.32
T7	52%	48%	1%	99%	0.05	0.00	1.19	1.00
T8	100%	0%	0%	100%	0.71	0.56	3.15	2.55
T9	100%	0%	0%	100%	0.00	0.00	2.38	2.38
T10	100%	0%	0%	100%	0.00	0.00	3.06	3.06
T11	33%	67%	0%	100%	0.58	0.67	2.44	2.57
T12	33%	67%	33%	67%	1.05	0.84	2.89	2.59
T13	83%	17%	0%	100%	0.73	0.76	2.72	2.49
T14	0%	100%	0%	100%	1.78	1.78	3.04	3.04

	Information hiding criteria				**Relational criteria**			
Type	**Encapsulation**		**Independence**		**Coupling**		**Redundancy**	
	Average	**Median**	**True**	**False**	**Average**	**Median**	**Average**	**Median**
T2	0.95	0.95	9%	91%	0.00	0.00	0.14	0.12
T6	0.99	0.99	70%	30%	0.0000256	0.00015	0.00065	0.00065
T7	1.00	1.00	100%	0%	0.00	0.00	0.00	0.00
T8	0.47	0.46	0%	100%	0.00	0.00	0.50	0.50

The relative intra-module distance values determine by how many units (paths between entities) the module has been reduced. T4 (locality) modules were reduced with a high value by 63.65 units followed by T1 (ontology design patterns) by 20.69 units. The T4 (locality), T8 (optimal reasoning), T9 (axiom abstraction), and T10 (entity type abstraction) modules all hold the correctness metrics; every axiom that exists in the module also exists in the source ontology and nothing new had been added. T1 (ontology design pattern) modules are the only set that all hold the completeness metric; the meaning of every entity in the module is preserved as in the source ontology. For attribute richness, T4 (locality) modules were the richest with a value of 3.61; these modules have on average 3.61 attributes per class. For inheritance richness, T6 (domain coverage) modules had a large value of 54.32 indicating many subclasses per class.

The information hiding and relational criteria only apply to module sets, T2 (subject domain), T6 (domain coverage), T7 (ontology matching), and T8 (optimal reasoning). For encapsulation, T7 (ontology matching) modules had a high value of 1; the knowledge is preserved in the individual modules, and they can be changed individually without affecting the other modules in the set. For coupling, most of the modules had 0 values (no links to other modules in the set). The T7 (ontology matching) modules are independent; they are self-contained and also do not contain links to other modules in the set.

4.2.7.3 Dependencies between type and evaluation metric

We examine the types of modules against the evaluation metrics. The dependency relationship between all these techniques and properties are displayed in Table 4.7. The metrics and values in bold font are those which evaluate well for a module type; hence such a module is considered of good quality. These values were set as 'good values' initially (recall Table 4.7), where the metrics were summarised. We discuss some notable dependencies here. T1: Ontology design pattern module is of good quality if its relative size is small, cohesion is small, and completeness is true. T2: Subject domain modules have a number of expected values for the metrics. For T3: Isolation branch modules, T13: Expressiveness sub-language modules, and T14: Expressiveness feature modules, the only dependencies that they have is that the cohesion is expected to be small. T6: Domain coverage modules, T7: Ontology matching modules, and T8: Optimal reasoning modules are highly dependent on relational criteria such as encapsulation, coupling, and redundancy.

4.2.7.4 Evaluating TOMM with ontology case-studies

We selected three existing cases of ontology modularisation to evaluate the metrics and the dependencies between the type and evaluation metrics. These three cases are modules that do not exist in the training set.

Example 22 (QUDT ontology modules) *This follows on from the earlier example 3 of the evaluation of the framework. Recall that the Quantities, Units, Dimensions and Data Types (QUDT) ontology modules are a set of modules about science*

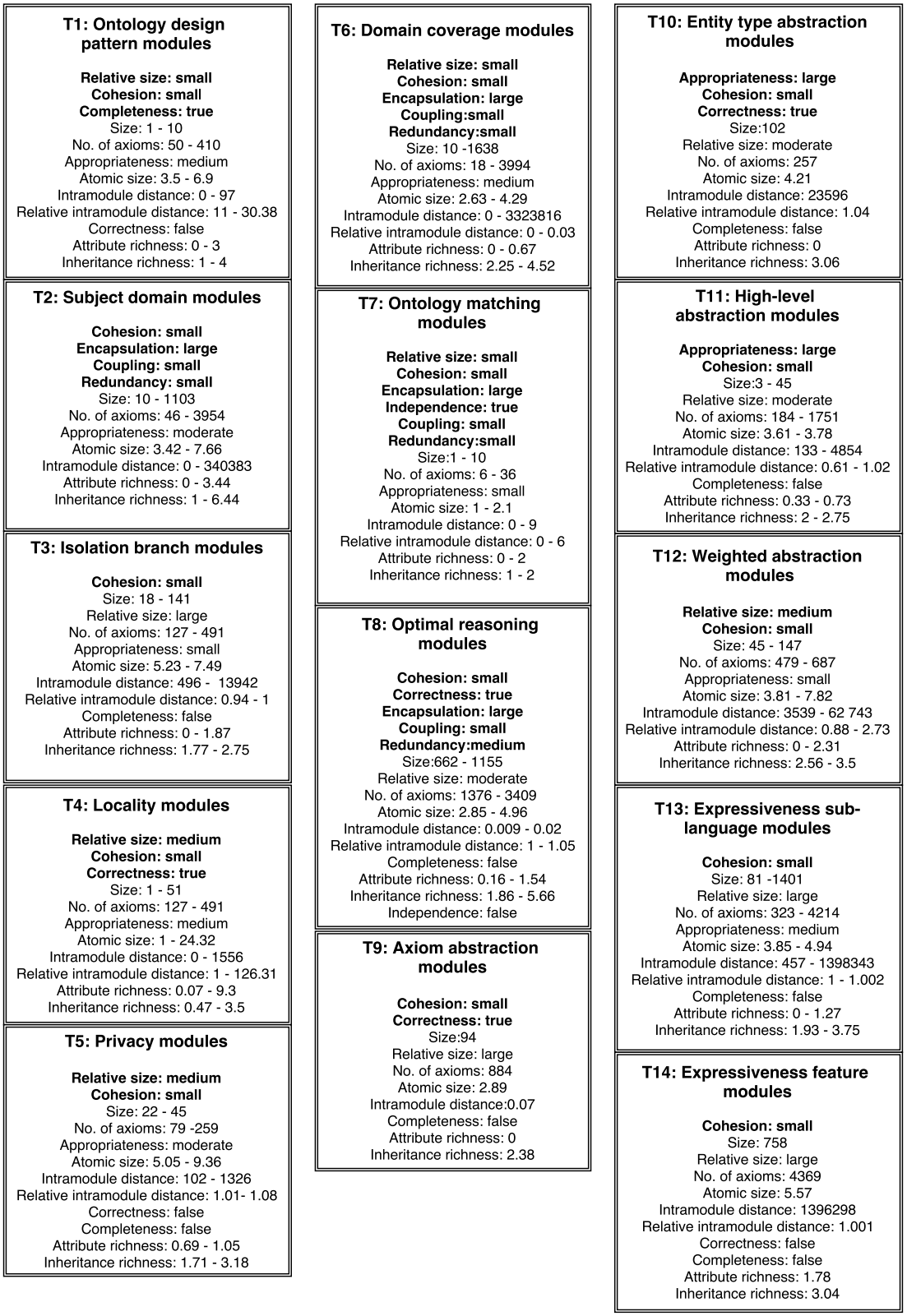


Figure 4.7: The set of metrics that can be measured for each module type. Metrics and values in bold font are those which evaluate well for a module type.

terminology for representing physical quantities, units of measure, and their dimensions [63]. According to the framework for ontology modularity, these modules are of type T2: Subject domain modules. The modules fare well for 3 out of the 4 metrics that are expected of T2: Subject domain modules; the cohesion is small, encapsulation is large, and coupling is small (see Table 4.11). The redundancy of the QUDT modules is 0.50 which is moderate, as opposed to an expected small value. For the metrics that are measured by their numerical values only, i.e., atomic size, attribute richness, etc., the metrics are within the expected ranges summarised in Fig. 4.7. Thus according to the metrics, the QUDT ontology modules are of good quality as subject domain modules.

Table 4.11: The metrics for the QUDT ontology modules generated by TOMM; approp = appropriateness, encap. = encapsulation, redund. = redundancy, avg. = average, med. = median.

	Structural criteria					
	Size	**Atomic size**	**No. of axioms**	**Approp.**	**Intra module distance**	**Cohesion**
Avg.	595.38	5.71	3112.00	0.91	8577.25	0.008
Med.	479.00	3.70	1443.50	0.91	86.50	0.003
	Richness criteria		**Information hiding criteria**		**Relational criteria**	
	AR	**IR**	**Encap.**	**Coupling**	**Independence**	**Redund.**
Avg.	1.69	1.89	0.99	0	25% true	0.50
Med.	1.40	1.84	0.99	0	75% false	0.50

Example 23 (The Pescado Ontology) *The Pescado ontology contains knowledge about the environment, such as meteorological conditions and phenomena, air quality, and disease information [132]. The PescadoDisease module is a subset of information only about diseases, so it is a locality module (type T4). The module fares well for the cohesion metric, which is small, the appropriateness value (being medium), the correctness metric (true), and for all those metrics measured by numerical ranges too, according to the expected values of Figure 4.7. The only metric that differs is relative size: the PescadoDisease module is small compared to the experimental data where locality modules were medium.*

Example 24 (The Symptom Ontology) *This follows on from the earlier example 6 of the evaluation of the framework. The Symptom ontology [6] contains knowledge about the symptoms and signs of diseases. We use the OWL Module extractor tool to extract a module containing knowledge about the skin symptoms, with a seed signature* skin and integumentary tissue symptom*, i.e., a locality module (type T4). We can now determine whether the module set is of good quality by checking the dependencies between type and evaluation metrics. The modules fare well for 2 out of the 3 metrics that are expected of locality modules; the cohesion is small, and*

Table 4.12: The metrics for the Pescado disease ontology generated by TOMM; app = appropriateness.

Structural criteria							
Size	**Atomic size**	**No. of axioms**	**App.**	**Intra module distance**	**Cohesion**	**Relative size**	**Relative intra module distance**
39.00	3.10	128.00	0.51	158.00	0.16	0.03	10.61
Richness criteria				**Logical criteria**			
Attribute richness		**Inheritance richness**		**Correctness**		**Completeness**	
0.00		1.67		True		True	

the correctness is true. The relative size of the module is small, 0.07, as opposed to a medium value.

Table 4.13: The metrics for the Symptom skin ontology module generated by TOMM; app = appropriateness, IMD = intra-module distance.

Structural criteria							
Size	**Atomic size**	**No. of axioms**	**App.**	**IMD**	**Cohesion**	**Relative size**	**Relative IMD**
68.00	2.97	468.00	0.04	5597.00	0.22	0.07	1.00
Richness criteria				**Logical criteria**			
Attribute richness		**Inheritance richness**		**Correctness**		**Completeness**	
0.04		8.38		True		True	

Using TOMM and the use-cases, we were able to evaluate the quality of ontology modules. For QUDT, Pescado, and Symptom we look up the expected values of metrics for their respective module types using the dependency diagram (Figure 4.7) that emerged from the experiment using TOMM metrics tool. With all three modules, for most of their metrics, the values, as determined by the existing and new metrics that were conveniently and automatically computed using TOMM are as expected for their types; this indicates that the modules are of 'good' quality.

From the list of new and existing metrics with corresponding equations that were compiled in Section 4.2, we were able to solve the problem that it is unclear how to determine whether a module is a good or bad module due to the lack of evaluation metrics. The list of module metrics was linked to various module types, by performing an experimental evaluation. The metrics are now comprehensive enough to apply to different types of modules whereas in previous work it was found that, for the criteria that do exist to evaluate modules, certain tools do not satisfy certain criteria [124].

4.3 Ontology interchangeability

In Section 4.2, we investigated measuring the quality of an ontology module by using new and existing evaluation metrics and the TOMM tool. In this Section, we investigate the idea of ontology interchangeability with the aim of improving the metrics of a module.

The growth in the amount of Semantic Web applications and ontology-mediated interoperability of complex software applications pushes demands for infrastructure to facilitate with semantic interoperability. Already from the early days of the Semantic Web, foundational ontologies have been proposed as a component to facilitate such interoperability for they provide high-level categories that are common across various domains and they have been proposed as a component to facilitate interoperability. A number of foundational ontologies have been developed e.g., DOLCE [102], BFO [102], GFO [62], SUMO [112], and YAMATO [106].

A top-domain ontology contains the fundamental concepts of a domain. They are commonly used in large, complex domains such as medical systems, and distributed media management [136]. Sometimes more than one top-domain ontology has been developed for a particular domain, e.g., BioTop and [13] and GFO-Bio [66], which provide concepts about life sciences. In some modular ontologies, foundational and top-domain ontologies are components that exist in the system. At other times, modules are used to extract a portion of the ontology for a particular use-case. For instance, the DMOP ontology is modularised by removing some expressive power to a DMOP-EL module to assist with faster reasoning [78].

Over the years, a considerable amount of research has been performed for modularity [20, 123, 131, 156] and modularity has been applied for various applications [13, 71, 78, 111, 121]. However, there is not much support for module management, i.e., software tools for modules. For instance, consider the modular ontology Subcellular Anatomy Ontology (SAO) [95] containing BFO ontology as a foundational ontology. Consider that upon validation of the ontology the developers notice that none of the SAO domain entities exists in the bfo:occurrent branch of the ontology, i.e., SAO does not need those entities. One solution is to modularise the SAO ontology by removing the bfo:occurrent branch and all sub-entities. Another option is to swap the BFO foundational ontology in the system for a subset module that already exists, the bfo-continuants.owl ontology module from the ROMULUS repository [88]. Currently, there is no tool support to do this.

Seeing that in many applications, modules are used together as components, and some of the components in a modular system are top-domain, foundational, and leaner modules, it is worthwhile to investigate the impact on interchanging ontology modules in a system, to first determine whether module interchangeability is possible for the purpose of ontology interoperability and integration and whether it could lead to some improvements concerning module metrics.

For this work on ontology interchangeability, we propose more mini questions, in addition to the main research questions of the thesis. The questions we focus on here are as follows, where O_A denotes a domain module, and O_X and O_Y are some modules (be they foundational, top-domain, ontology design pattern, or arbitrary):

1. If O_A is linked to a module O_X, is it feasible to interchange it to be linked to a different module O_Y?

2. Does interchanging ontology O_A between O_X and O_Y have an impact on the quality of the modules in the set?

3. Does interchanging ontology O_A between O_X and O_Y have an impact on the time taken for reasoning?

To solve this problem of insufficient module management tools and investigate ontology interchangeability and its impact on a modular ontology, we have developed Software Used to Gain Ontology Interchangeability (SUGOI). SUGOI was originally developed for assisting with semantic interoperability by allowing for the interchange among various foundational ontologies [81] but it was designed to easily handle extensibility making it easy to generalise it to be used among any kind of ontologies provided input mapping files are provided. We adapted SUGOI to make a generalised version, SUGOI-Gen. The aim of the investigation with SUGOI-Gen and ontology use-cases in the following sections is to determine whether the interchangeability of modules in modular systems is possible and whether they have an impact the quality of the modules. The remainder of this section has been submitted for publication in [89].

4.3.1 Interchangeability algorithm design

We first introduce the terms used for SUGOI's input files as they are referenced by the algorithm, and thereafter the algorithm.

4.3.1.1 The interchangeability files

Because several ontology files are used in the interchangeability, we describe here the terms used for each one. This follows with a diagram showing the various files in Figure 4.8 for the modular SAO ontology.

- The *Source Modular Ontology* (${}^s\mathcal{O}_m$) that the user wants to interchange, which comprises the *Source Interchange Module* (${}^s\mathcal{M}_i$) that is the modular component of the source ontology that is to be interchanged, the *Source Domain Modules* (${}^s\mathcal{M}_d$) that is the domain modules in the ontology, and any equivalence or subsumption mappings between entities in ${}^s\mathcal{M}_i$ and ${}^s\mathcal{M}_d$.

- The *Target Modular Ontology* (${}^t\mathcal{O}_m$) which has been interchanged, which comprises the *Target Interchange Module* (${}^t\mathcal{M}_i$), that is the modular component of the target ontology that the user has selected to interchange to, the *Target Domain Modules* (${}^t\mathcal{M}_d$) that is the domain modules in the ontology, and any equivalence or subsumption mappings between entities in ${}^t\mathcal{M}_i$ and ${}^t\mathcal{M}_d$.

- *Mapping ontology* (${}^a\mathcal{O}$): the mapping ontology between the ${}^s\mathcal{M}_i$ and the ${}^t\mathcal{M}_i$.

- *Domain entity*: an entity from ${}^s\mathcal{M}_d$ or ${}^t\mathcal{M}_d$.

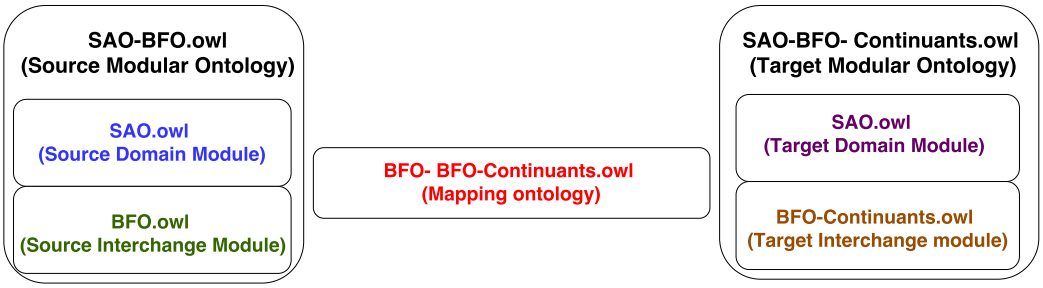


Figure 4.8: The terminologies used for the files involved in interchangeability using the SAO ontology.

4.3.1.2 Ontology interchangeability algorithm

The main steps for the interchangeability of the foundational ontology algorithm was first introduced in earlier work [81]. Now, we have generalised the algorithm to be used for the interchangeability of any module, and we provide the full algorithm here.

The general idea of the algorithm behind SUGOI is that it accepts a ${}^{s}\mathcal{O}_m$ consisting of a ${}^{s}\mathcal{M}_d$ linked to a ${}^{s}\mathcal{M}_i$ (for instance, a foundational ontology such as DOLCE, BFO or GFO) and converts it to a ${}^{t}\mathcal{O}_m$ with a different ${}^{t}\mathcal{M}_i$. The ${}^{s}\mathcal{O}_m$ is provided by the user. It does not matter whether the ${}^{s}\mathcal{M}_d$ is linked to a ${}^{s}\mathcal{M}_i$ by an import or a merge. SUGOI accesses the remainder of the ontologies either by loading the ontology from the online URI, by loading it from an offline file, or from the user's input files, depending on the version in use.

After the interchange process, all the domain entities from the ${}^{s}\mathcal{M}_d$ are present in the ${}^{t}\mathcal{M}_d$. SUGOI links domain entities from the ${}^{s}\mathcal{M}_d$ to the ${}^{t}\mathcal{M}_i$ as follows. SUGOI maps a domain entity's superentity in the ${}^{s}\mathcal{M}_i$ to its corresponding superentity in the ${}^{t}\mathcal{M}_i$ using the mapping ontology. If the domain entity's superentity does not have a corresponding mapping entity, SUGOI then treats that superentity as a domain entity and looks for a corresponding mapping entity at a higher level up in the taxonomy. Thus, eventually, the domain entity from the ${}^{s}\mathcal{M}_d$ is mapped with on-the-fly subsumption.

In this section, we provide the main steps of the algorithm and the full algorithm thereafter in two parts (Algorithms 1-2).

1. Create a ${}^{t}\mathcal{O}_m$ (line 2 of Algorithm 1).

2. Copy axioms from the ${}^{t}\mathcal{M}_i$ to the ${}^{t}\mathcal{O}_m$ (line 3 of Algorithm 1).

3. Copy the ${}^{s}\mathcal{M}_d$ (domain axioms) to the ${}^{t}\mathcal{O}_m$ (lines 4-9 of Algorithm 1).

4. Map the ${}^{s}\mathcal{M}_d$ domain entities to the ${}^{t}\mathcal{M}_i$ using the mapping ontology (lines 10-19 of Algorithm 1).

5. Perform on-the-fly subsumption if a domain entity from the previous step is not linked to a ${}^{t}\mathcal{M}_i$ (lines 20-37 of Algorithm 2).

6. Delete ${}^{s}\mathcal{M}_i$ entities that are not referenced by the domain entities in the ${}^{t}\mathcal{O}_m$ (lines 38-52 of Algorithm 2).

Algorithm 1 Ontology interchangeability algorithm: part 1

1: **Input** ${}^{s}\mathcal{O}_m$, ${}^{s}\mathcal{M}_i$, ${}^{t}\mathcal{M}_i$, ${}^{a}\mathcal{O}$
2: **Output** ${}^{t}\mathcal{O}_m$
3: ▷ Steps 1-2. Create an ontology ${}^{t}\mathcal{O}_m$. Copy axioms from the ${}^{t}\mathcal{M}_i$ to the ${}^{t}\mathcal{O}_m$
4: ${}^{t}\mathcal{O}_m \leftarrow {}^{t}\mathcal{M}_i$
5: ▷ Step 3. Copy the ${}^{s}\mathcal{M}_d$ domain axioms to the ${}^{t}\mathcal{O}_m$
6: **for all** *entity* $\in {}^{s}\mathcal{O}_m$ **do**
7: **if** *entity* not in ${}^{s}\mathcal{M}_i$ **then**
8: *currentAxiom* $\leftarrow$ get current axiom
9: add *currentAxiom* to ${}^{t}\mathcal{O}_m$
10: **end if**
11: **end for**
12: ▷ Step 4. Map the ${}^{s}\mathcal{M}_d$ domain entities to the ${}^{t}\mathcal{M}_i$ using the ${}^{a}\mathcal{O}$
13: **for all** *entity* $\in {}^{t}\mathcal{O}_m$ **do** ▷ if *entity* is a domain entity
14: **if** *entity* not in ${}^{s}\mathcal{M}_i$**and** *entity* not in ${}^{t}\mathcal{M}_i$ **then**
15: *currentAxiom* $\leftarrow$ get current axiom
16: *entitySet* $\leftarrow$ get entities in signature of *currentAxiom*
17: **for all** *signatureEntity* $\in$ *entitySet* **do**
18: **if** *signatureEntityin* ${}^{a}\mathcal{O}$ **then** *eSignatureEntity* $\leftarrow$ get equivalent entity of *signatureEntity currentAxiom* $\leftarrow$ replace *signatureEntity* with *eSignatureEntity* in *currentAxiom*
19: **end if**
20: **end for**
21: **end if**
22: **end for**

4.3.2 SUGOI ontology interchangeability tool

To test the effect of swapping a module for another aligned module in a system of modular ontologies, we have implemented the algorithm in Section 4.3.1.2, and extended the SUGOI tool, Software Used to Gain Ontology Interchangeability, to a generalised version, SUGOI-Gen.

There are a few different platform-independent versions of SUGOI as follows:

1. **SUGOI-Gen Desktop online version:** a platform independent jar file to be executed on a local machine but requires internet connectivity. This version can interchange any type of module but requires all input and mapping files from the user.

2. **SUGOI-FO Applet:** an online web version integrated into the ROMULUS repository [80]. This version is especially-designed to interchange between

Algorithm 2 Ontology interchangeability algorithm: part 2

▷ Step 5. Perform on-the-fly subsumption, if a domain entity from previous step is not linked to a ${}^{t}\mathcal{M}_i$
for all *entity* $\in {}^{t}\mathcal{O}_m$ **do**
    if *entity* not in ${}^{t}\mathcal{M}_i$ **then**
        if *entity* has no superclasses in ${}^{t}\mathcal{O}_m$ **then**
            ancestorSet ← get ancestor entities of *entity* from ${}^{s}\mathcal{M}_i$
            mappableSet ← empty set
            for all *ancestorEntity* $\in$ *ancestorSet* **do**
                if *ancestorEntity* exists in ${}^{a}\mathcal{O}$ **then**
                    add *ancestorEntity* to *mappableSet*
                end if
            end for ▷ get lowest level entity
            selectedEntity ← get lowest level entity from *mappableSet*
            mappedSelectedEntity ← get entity $\equiv$ *selectedEntity* from ${}^{a}\mathcal{O}$
            newAxiom ← create axiom stating that *entity* is a subclass of *mappedSelectedEntity*
            add *newAxiom* to ${}^{t}\mathcal{O}_m$
        end if
    end if
end for
▷ Step 6. Delete source foundational ontology entities that are not referenced by the domain entities
for all *entity* $\in {}^{t}\mathcal{O}_m$ **do**
    if *entity* $\in {}^{s}\mathcal{M}_i$ **then** *entitySet* ← get referencing entities of *entity*
        for all $referencedEntity$ in *entitySet* **do** ▷ if $referencedEntity$ is a domain entity
            if $referencedEntity$ not in ${}^{s}\mathcal{M}_i$**and** $referencedEntity$ not in ${}^{t}\mathcal{M}_i$ **then**
                checker ← true
            end if
        end for
        ▷ if *entity* is not referenced by any domain entities
        if *checker* == false **then**
            if *entity* in ${}^{t}\mathcal{M}_i$ **then**
                remove *entity* from ${}^{t}\mathcal{O}_m$
            end if
        end if
    end if
end for
save ${}^{t}\mathcal{O}_m$
generate log file with metrics

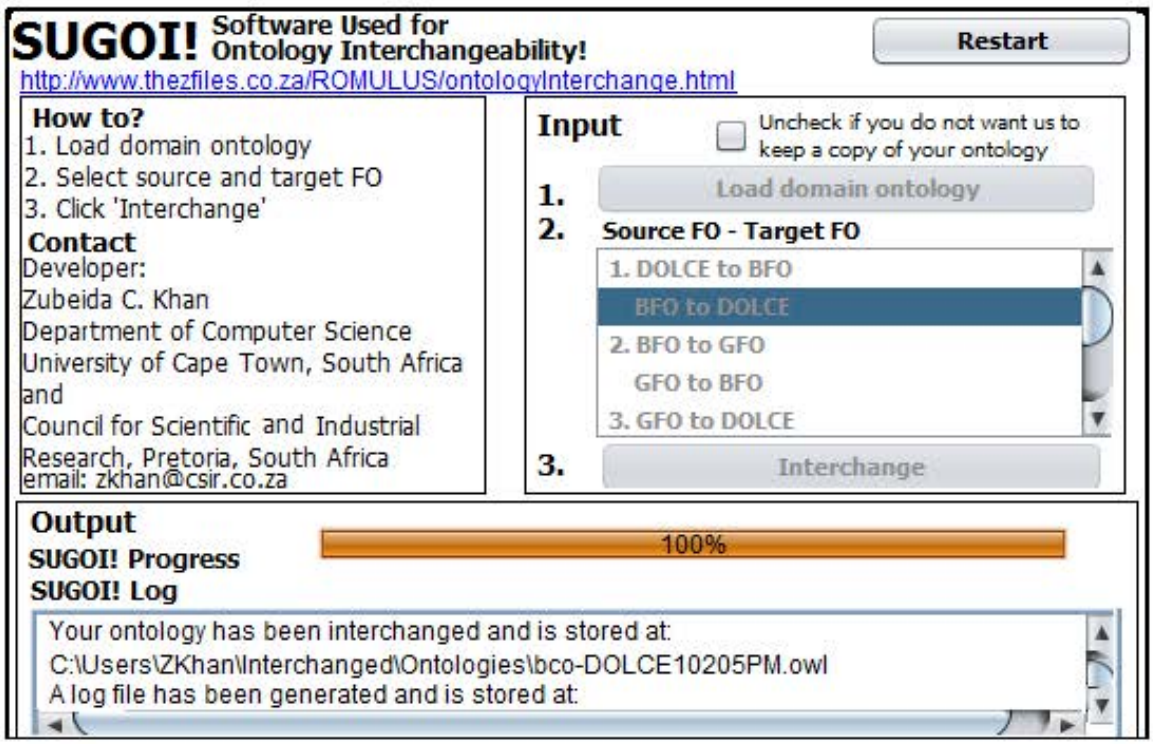


Figure 4.9: The interface of the online desktop version of SUGOI.

DOLCE, BFO, and GFO foundational ontologies and has the mapping files built into the tool.

3. **SUGOI-FO Desktop online version:** a platform independent jar file to be executed on a local machine but requires internet connectivity. This version is especially-designed to interchange between DOLCE, BFO, and GFO foundational ontologies and has the mapping files built into the tool.

4. **SUGOI-FO Desktop offline version:** a platform independent jar file to be executed on a local machine and is bundled with foundational and mapping ontology files. This version is especially-designed to interchange between DOLCE, BFO, and GFO foundational ontologies and has the mapping files built into the tool.

SUGOI was developed in Java using the OWLAPI v3.5.0 in Netbeans IDE 8.0. The Applet of SUGOI is deployed online within any browser that has the Java TM Platform plugin installed and activated. The desktop versions of SUGOI are platform independent jar files together with their dependencies (all bundled together) that require minimal disk space, and Java runtime components installed. The different version of SUGOI can be downloaded from the foundational ontology library ROMULUS at `http://www.thezfiles.co.za/ROMULUS/ontologyInterchange.html`.

Fig. 4.9 is a screenshot of the online desktop version of SUGOI when interchanging an ontology from BFO to DOLCE, and Fig. 4.10 is a screenshot of the online desktop version of SUGOI-Gen.

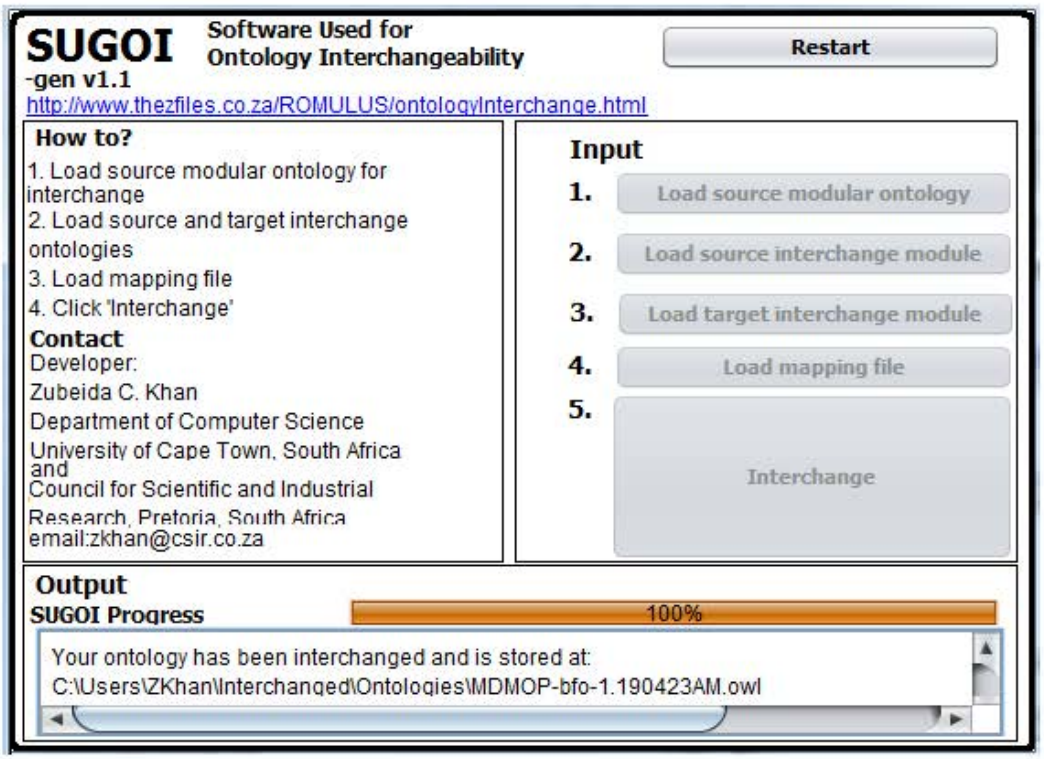


Figure 4.10: The interface of the online desktop version of SUGOI-Gen.

4.3.3 Interchangeability with modular ontologies

To investigate the interchangeability of various ontology modules, we use the generalised version of SUGOI, SUGOI-Gen, since it can be used to swap any type of OWL ontology modules. We have identified three scenarios to investigate ontology module interchangeability which we discuss here.

4.3.3.1 Swapping foundational ontologies

For this scenario, we consider ontology modules that contain a domain component and foundational ontology component. Since there are several foundational ontologies that have been developed, this poses problems if a heterogeneous system needs to access ontologies that are linked to conflicting foundational ontologies. Swapping a foundational ontology for another could be performed to solve this problem of Semantic interoperability. It is also worthwhile to investigate whether swapping its foundational ontology to an aligned foundational ontology has an impact on the modularisation metrics, and the time taken for reasoning.

A characteristic of this scenario is that the ${}^{s}\mathcal{M}_i$, ${}^{a}\mathcal{O}$, and ${}^{t}\mathcal{M}_i$ cover the philosophical, high-level categories that are common across various domains

We provide three illustrative examples for interchanging between foundational ontologies in Examples 25, 26, and 27. Figure 4.11 illustrates Example 25. The source ${}^{s}\mathcal{O}_m$ and two target ${}^{t}\mathcal{O}_m$ ontologies is shown for Example 26 for interchanging the sao:Membrane Surface entity from the SAO ontology in Fig 4.12.

Example 25 *The basic steps of the algorithm for interchanging between foundational ontologies DOLCE to GFO are as follows, using the data mining DMOP ontology [78] as an example:*

1. *Create a new ontology file, a* ${}^{t}\mathcal{O}_m$*:* `dmop-gfo.owl`*.*
2. *Copy the entire* ${}^{t}\mathcal{M}_i$ *to the* ${}^{t}\mathcal{O}_m$*: copy the GFO ontology into* `dmop-gfo.owl`*.*
3. *Copy the axioms from the* ${}^{s}\mathcal{M}_d$ *to the* ${}^{t}\mathcal{O}_m$*: e.g., consider the axioms, axiom1:* dmop:DecisionBoundary $\sqsubseteq$ dolce:abstract *and axiom2:* dmop:Strategy $\sqsubseteq$ dolce:Non PhysicalEndurant *which exist in the* ${}^{s}\mathcal{O}_m$ *DMOP. We add these axioms to the* `dmop-gfo.owl` ${}^{t}\mathcal{O}_m$ *and they are referred to as 'new' axioms.*
4. *Change the 'new' axioms to reference* ${}^{t}\mathcal{M}_i$ *entities, if mappings exist: for axiom1, there is an equivalence mapping between* gfo:Abstract *and* dolce:abstract*, hence we change axiom1* dmop:DecisionBoundary $\sqsubseteq$ dolce:abstract *to* dmop: DecisionBoundary $\sqsubseteq$ gfo:Abstract*. For axiom2, there is no equivalence mapping between* dolce:NonPhysicalEndurant *and GFO entities; we skip this step.*
5. *If a mapping does not exist, perform on-the-fly subsumption: For axiom2,* dolce:NonPhysicalEndurant *has a superclass* dolce:Endurant *and the mapping ontology has* dolce:endurant $\equiv$ gfo:Presential*, so* dolce:NonPhysicalEndurant $\sqsubseteq$ gfo:Presential *is added to* `dmop-gfo.owl`*.*
6. *Delete entities that exist in the* ${}^{t}\mathcal{O}_m$ *that are from the* ${}^{s}\mathcal{M}_i$ *but do not appear in an axiom with entities from the* ${}^{t}\mathcal{M}_d$*, resulting in the final* ${}^{t}\mathcal{O}_m$*,* `dmop-gfo.owl`*. Delete the* dolce:abstract *entity from* `dmop-gfo.owl`*.*

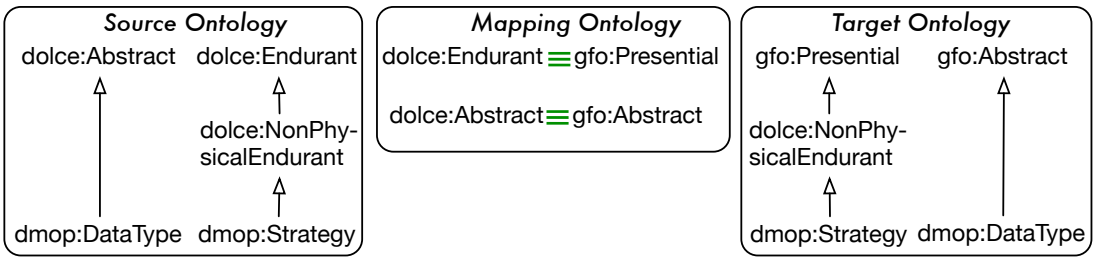


Figure 4.11: Examples of interchanging the dmop:DataType and dmop:Strategy domain entities from ${}^{s}\mathcal{M}_i$ DOLCE to ${}^{t}\mathcal{M}_i$ GFO with SUGOI, using equivalence and subsumption mappings. Source: [81]

Example 26 *The basic steps of the algorithm for interchanging between foundational ontologies BFO to DOLCE are as follows, using the Subcellular Anatomy Ontology (SAO) [96] as an example:*

1. *Create a new ontology file, a* ${}^{t}\mathcal{O}_m$*:* `sao-dolce.owl`*.*
2. *Copy the entire* ${}^{t}\mathcal{M}_i$ *to the* ${}^{t}\mathcal{O}_m$*: copy the OWLized DOLCE ontology into* `sao-dolce.owl`*.*

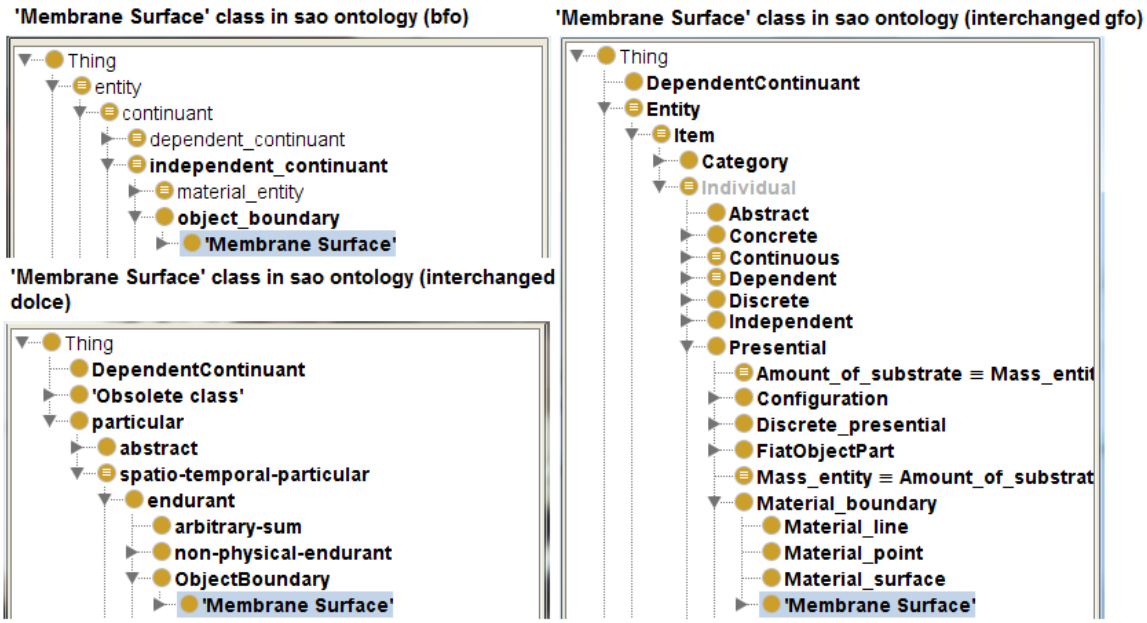


Figure 4.12: The position of the sao:Membrane Surface class in source and target ontologies. Source: [84].

3. *Copy the axioms from the* ${}^{s}\mathcal{M}_d$ *to the* ${}^{t}\mathcal{O}_m$*: e.g., the axiom* sao:Membrane Surface $\sqsubseteq$ bfo:Object_boundary *exists in the* ${}^{s}\mathcal{O}_m$ *SAO, which is added to the* `sao-dolce.owl` ${}^{t}\mathcal{O}_m$ *and is referred to as a 'new' axiom.*

4. *Change the 'new' axioms to reference* ${}^{t}\mathcal{M}_i$ *entities, if mappings exist: for the example in the previous step, no mapping exists for* bfo:Object_boundary *between BFO and DOLCE, so it proceeds to the next step.*

5. *If a mapping does not exist, perform on-the-fly subsumption: continuing with the example,* bfo:Object_boundary *has a superclass* bfo:Independent_Continuant *and the mapping ontology has* bfo:Independent_Continuant $\equiv$ dolce:endurant*, so* bfo:Object_ boundary $\sqsubseteq$ dolce:endurant *is added to* `sao-dolce.owl`*.*

6. *Delete entities that exist in the* ${}^{t}\mathcal{O}_m$ *that are from the* ${}^{s}\mathcal{M}_i$ *but do not appear in an axiom with entities from the* ${}^{t}\mathcal{M}_d$*, resulting in the final* ${}^{t}\mathcal{O}_m$*,* `sao-dolce.owl`*.*

Example 27 *The SEGO ontology [36] is about sensing geographical occurrences, and it is linked to the DOLCE foundational ontology. Let us assume that we wish to integrate the SEGO ontology to the Infectious Disease Ontology (IDO) [28] to gain information about the geographical occurrences of diseases. The IDO ontology, however, is linked to a different foundational ontology than the SEGO ontology. IDO is linked to the BFO foundational ontology. These conflicting foundational ontologies prevent such interoperability. In order to solve this problem, we could consider using SUGOI-Gen to interchange the SEGO ontology from DOLCE to BFO, or to interchange the IDO ontology from BFO to DOLCE, provided that there are mappings*

that have been created between DOLCE and BFO foundational ontologies. The basic steps of the algorithm for interchanging between DOLCE to GFO as a foundational ontology are as follows, using the SEGO ontology as an example:

1. *Create a new ontology file, a* ${}^{t}\mathcal{O}_m$*:* `sego-gfo.owl`*.*
2. *Copy the entire* ${}^{t}\mathcal{M}_i$ *to the* ${}^{t}\mathcal{O}_m$*: copy the GFO ontology into* `sego-gfo-bio.owl`*.*
3. *Copy the axioms from the* ${}^{s}\mathcal{M}_d$ *to the* ${}^{t}\mathcal{O}_m$*: e.g., consider the axioms, axiom1:* sego:sensor $\sqsubseteq$ dolce:physical-object *and axiom2:* sego:geo-process $\sqsubseteq$ dolce:stative *which exist in the* ${}^{s}\mathcal{O}_m$ *sego. We add these axioms to the* `sego-gfo.owl` ${}^{t}\mathcal{O}_m$ *and they are referred to as 'new' axioms.*
4. *Change the 'new' axioms to reference* ${}^{t}\mathcal{M}_i$ *entities, if mappings exist: for axiom1, there is an equivalence mapping between* `dolce:physical-object` *and* `gfo:Material_object`*, hence we change axiom1* sego:sensor $\sqsubseteq$ dolce:physical-object *to* sego:sensor $\sqsubseteq$ gfo:Material_object*. For axiom2, there is no equivalence mapping between* dolce:stative *and GFO entities; we skip this step.*
5. *If a mapping does not exist, perform on-the-fly subsumption: For axiom2,* dolce:stative *has a superclass* dolce:perdurant *and the mapping ontology has* dolce:perdurant $\equiv$ gfo:Occurrent*, so* dolce:stative $\sqsubseteq$ gfo:Occurrent *is added to* `sego-gfo.owl`*.*
6. *Delete entities that exist in the* ${}^{t}\mathcal{O}_m$ *that are from the* ${}^{s}\mathcal{M}_i$ *but do not appear in an axiom with entities from the* ${}^{t}\mathcal{M}_d$*, resulting in the final* ${}^{t}\mathcal{O}_m$*,* `sego-gfo.owl`*. Delete the* dolce:physical-object *entity from* `seo-gfo.owl`*.*

4.3.3.2 Swapping top-domain ontologies

For this scenario, we consider ontology modules that contain a domain component and top-domain component. A user may want to interchange the top-domain component for another to assist with ontology integration. It is also worthwhile to investigate whether swapping its top domain ontology to a top domain ontology has an impact on the modularisation metrics, and the time taken for reasoning.

A characteristic of this scenario is that the ${}^{s}\mathcal{M}_i$, ${}^{a}\mathcal{O}$, and ${}^{t}\mathcal{M}_i$ cover the fundamental concepts of a particular domain.

Example 28 *BioTop [13] and GFO-Bio [66] are both top-domain biological ontologies and CELDA [142] is an ontology for complex cells. CELDA imports the BioTop ontology as a top-domain ontology. Suppose there is a set of bio-medical ontologies that use GFO-Bio as a top-domain ontology and we wish to use CELDA in an application together with the set of bio-medical entities. It is difficult to achieve seamless integration because CELDA has BioTop as a top-domain component. Hence, we consider interchanging CELDA's top-domain ontology from BioTop to GFO-Bio. The basic steps of the algorithm for interchanging between BioTop to GFO-Bio as a top-domain ontology as an example are as follows.*

1. *Create a new ontology file, a* ${}^t\mathcal{O}_m$*:* `CELDA-gfo-bio.owl`*.*

2. *Copy the entire* ${}^t\mathcal{M}_i$ *to the* ${}^t\mathcal{O}_m$*: copy the GFO-Bio ontology into* `CELDA-gfo-bio.owl`*.*

3. *Copy the axioms from the* ${}^s\mathcal{M}_d$ *to the* ${}^t\mathcal{O}_m$*: e.g., consider the axioms, axiom1:* celda:compound $\sqsubseteq$ biotop:MaterialObject *and axiom2:* celda:GO_008150 $\sqsubseteq$ biotop:bio_molecular_process *which exist in the* ${}^s\mathcal{O}_m$ *CELDA. We add these axioms to the* `CELDA-gfo-bio.owl` ${}^t\mathcal{O}_m$ *and they are referred to as 'new' axioms.*

4. *Change the 'new' axioms to reference* ${}^t\mathcal{M}_i$ *entities, if mappings exist: for axiom1, there is an equivalence mapping between* biotop:MaterialObject *and* gfo-bio:Material_object*, hence we change axiom1* celda:compound $\sqsubseteq$ biotop:MaterialObject *to* celda:compound $\sqsubseteq$ gfo-bio:Material_Object*. For axiom2, there is no equivalence mapping between* biotop:bio_molecular_process *and GFO entities; we skip this step.*

5. *If a mapping does not exist, perform on-the-fly subsumption: For axiom2,* biotop:bio_molecular_process *has a superclass* biotop:biological_processual_entity *and the mapping ontology has* biotop:biological_processual_entity $\equiv$ gfo-bio:Biological_process*, so* biotop:bio_molecular_process $\sqsubseteq$ gfo-bio:Biological_process *is added to* `CELDA-gfo-bio.owl`*.*

6. *Delete entities that exist in the* ${}^t\mathcal{O}_m$ *that are from the* ${}^s\mathcal{M}_i$ *but do not appear in an axiom with entities from the* ${}^t\mathcal{M}_d$*, resulting in the final* ${}^t\mathcal{O}_m$*,* `dmop-gfo.owl`*. Delete the* biotop:MaterialObject *entity from* `CELDA-gfo-bio.owl`*.*

4.3.3.3 Swapping an ontology for a leaner fragment the ontology

For this scenario, we consider ontology modules that contain a domain component and an arbitrary module that has a potential alignment to a 'leaner' module. That is, a particular type of fragment of an ontology, such as one represented in a language of lower expressiveness, instead of the 'comprehensive' version of the ontology. A user may want to interchange one of the modules in the set of ontologies for another to perhaps enhance user comprehension or increase performance of the automated reasoner, or of the ontology-driven information system. It is also worthwhile to investigate whether swapping one of its modules for a smaller version of the module has an impact on the modularisation metrics, and the time taken for reasoning.

A characteristic of this scenario is that the interchanged modules do have an exact overlap in content, just fewer axioms and possibly a subset of the vocabulary as well in the 'leaner' version.

Example 29 *In the DMOP data mining ontology [78], there is the domain ontology, DMOP and the foundational ontology, DOLCE-Lite. Aside from this, there already exists an EL version of the DOLCE-Lite ontology in the ROMULUS repository [88]. We could consider interchanging the DOLCE-Lite foundational ontology module for*

the DOLCE-EL foundational ontology module to assist with faster reasoning of the DMOP module. The basic steps of the algorithm for interchanging between DOLCE to DOLCE-Lite in the DMOP ontology are as follows:

1. *Create a new ontology file, a* ${}^t\mathcal{O}_m$*:* `dmop-dolce-el.owl`*.*

2. *Copy the entire* ${}^t\mathcal{M}_i$ *to the* ${}^t\mathcal{O}_m$*: copy the dolce-el ontology into* `dmop-dolce-el.owl`*.*

3. *Copy the axioms from the* ${}^s\mathcal{M}_d$ *to the* ${}^t\mathcal{O}_m$*: e.g., consider the axioms, axiom1:* dmop:Characteristic $\sqsubseteq$ dolce:abstract-quality *which exists in the* ${}^s\mathcal{O}_m$ *dmop-dolce. We add this axiom to the* `dmop-dolce-el.owl` ${}^t\mathcal{O}_m$ *and it is referred to as a 'new' axiom.*

4. *Change the 'new' axioms to reference* ${}^t\mathcal{M}_i$ *entities, if mappings exist: for axiom1, there is an equivalence mapping between* dolce:abstract-quality *and* dolce-el:abstract- quality*, hence we change axiom1* dmop:Characteristic $\sqsubseteq$ dolce:abstract-quality *to* dmop:Characteristic $\sqsubseteq$ dolce-el:abstract-quality*.*

5. *If a mapping does not exist, perform on-the-fly subsumption. There are mappings for every DOLCE to DOLCE-EL entity hence we skip this step.*

6. *Delete entities that exist in the* ${}^t\mathcal{O}_m$ *that are from the* ${}^s\mathcal{M}_i$ *but do not appear in an axiom with entities from the* ${}^t\mathcal{M}_d$*, resulting in the final* ${}^t\mathcal{O}_m$*,* `dmop-dolce-el.owl`*. Delete the* dolce:abstract-quality *entity from* `dmop-dolce-el.owl`*.*

4.3.3.4 Swapping aligned ontology design patterns

For this scenario, we consider ontology modules that contain a domain component and an ontology pattern module that has a potential alignment. It is worthwhile to investigate whether swapping the pattern for its aligned pattern has an impact on the modularisation metrics. For instance, consider the class vs. object property, which can also be found as issue in conceptual modelling: e.g., should 'marriage' be a class Marriage with a number of persons participating in it, or a relationship/object property married-to, i.e., to reify a relationship or not. Or the modelling approach of subsumption vs. inherence: e.g., for a library ontology, take the knowledge 'Librarian inheres in some Person', or: social objects are related to physical objects through an inheresIn object property (Librarian $\sqsubseteq \exists$inheresIn.Person), yet an ontology developer wishes to use this knowledge in a database for a library system, for which an assertion of 'Librarian isA Person' (Librarian $\sqsubseteq$ Person) is much more efficient. Hence, the instantiation of an ontology *pattern*—i.e., involving more than one element—in the library ontology has to be swapped for the simple named class subsumption. Five such types of patterns are aligned in [41], and there are surely more variants.

A characteristic of this scenario is that the ontology patterns are not necessarily of equal size and may use different language features, such as simple class subsumption vs. existential quantification. The overall effect on ontology metrics for one pattern

interchange is not expected to have an impact, but it will if this were to be done throughout an ontology for each such instance.

In the next section, we will investigate ontology interchangeability with modules for these scenarios.

4.3.4 Experimental Evaluation

We now perform an experimental evaluation with a set of modules using SUGOI-Gen.

The first purpose of the experimental evaluation is to investigate the interchangeability of modules for the identified scenarios to determine how well the algorithm for interchangeability performs. For this, we assess whether SUGOI-Gen can successfully interchange a ${}^{s}\mathcal{O}_m$ to a ${}^{t}\mathcal{O}_m$ and determine the amount of the ontology that will be effectively interchanged, which refers to those entities within the ${}^{t}\mathcal{M}_i$ that have been mapped with equivalence relations, thereby not required to use parts of the ${}^{s}\mathcal{M}_i$ in the ${}^{t}\mathcal{O}_m$. Second, we assess the source and target modules to determine the effects that interchangeability may have on a modular ontology in terms of the module's metrics and reasoning processing.

4.3.4.1 Materials and methods

The assessment has been designed using the three of the four scenarios for module interchangeability, i.e., swapping foundational ontologies, swapping top-domain ontologies, and swapping an ontology for a leaner version of it. The method for the experiment is as follows:

1. Collect module sets from existing works for each of the three scenarios.
2. Create a mapping ontology file for each module set.
3. Interchange the *Source Modular Ontology* (${}^{s}\mathcal{O}_m$) to a *Target Modular Ontology* (${}^{t}\mathcal{O}_m$).
4. Analyse the *raw interchangeability* of each ${}^{t}\mathcal{O}_m$, i.e., a measure to determine the amount of the ${}^{t}\mathcal{O}_m$ that has been correctly interchanged using equivalence mappings thereby not referring to the ${}^{s}\mathcal{M}_i$ entities. This is calculated from the ${}^{t}\mathcal{O}_m$ as follows: Let *GT, good target linking axioms*, represent the sum of axioms that link domain ontology entities and ${}^{t}\mathcal{M}_i$ entities in the ${}^{t}\mathcal{O}_m$. Let *BT, bad target linking axioms*, represent the sum of axioms that link domain ontology entities and ${}^{s}\mathcal{M}_i$ entities in the ${}^{t}\mathcal{O}_m$; the raw interchangeability is calculated as follows:
$$\textit{Raw interchangeability} = \frac{|GT|}{|GT\ +\ BT|} \times 100 \tag{4.19}$$
5. Compare the time taken for reasoning for the ${}^{s}\mathcal{O}_m$ against the ${}^{t}\mathcal{O}_m$.
6. Run the TOMM metrics tool for the ${}^{s}\mathcal{O}_m$ and the ${}^{t}\mathcal{O}_m$ module sets.
7. Analyse and compare the metrics for the ${}^{s}\mathcal{O}_m$ and ${}^{t}\mathcal{O}_m$ module sets.

Figure 4.13: The mapping file showing alignments for classes between DOLCE and BFO foundational ontologies.

The materials consist of six ${}^{s}\mathcal{O}_m$ that were randomly selected from existing ontology modules. For the six ${}^{s}\mathcal{O}_m$, two were from each scenario, covering domains about animals, space and time, anatomy, cells, etc. We used the SUGOI-Gen interchangeability tool and TOMM metrics tool.

The materials consist of six ${}^{s}\mathcal{O}_m$, two from each scenarios, covering domains about animals, space and time, anatomy, cells, etc., mapping files created for each ${}^{s}\mathcal{M}_i$ to ${}^{t}\mathcal{M}_i$ alignment created with Logmap [72] and manually, the SUGOI-Gen interchangeability tool, and TOMM metrics tool [87] as it generates evaluation metrics to measure the quality of a module. The mapping files that were used for the foundational ontology modules were manually created for previous experiments [81], and for the rest of the modules, we used Logmap to create them. A mapping file for the alignment between BioTop and GFO-Bio is shown in Figure 4.13. All the test files used for this experimental evaluation can be downloaded from `www.thezfiles.co.za/ROMULUS/testfiles.zip`.

4.3.4.2 Results

All of the modules were successfully interchanged with SUGOI-Gen. To determine how the effective swapping the various modules were, we first look at the interchangeability values that were calculated with the SUGOI-Gen tool, displayed in Table 4.14. For the first set of modules, for interchanging foundational ontologies, the raw interchangeability values were between 25-28%; i.e., about a quarter of the ${}^{s}\mathcal{O}_m$ was successfully interchanged completely. This is because there are only 15 mappings that exist between DOLCE and GFO, and 7 between DOLCE and BFO, in the respective mapping files. The remaining 75% of the foundational ontology modules that were not successfully interchanged are represented by the BT values in the raw interchangeability formula. These are the axioms that link domain ontology entities to ${}^{s}\mathcal{M}_i$ entities in the ${}^{t}\mathcal{O}_m$. For instance, the axiom ontoderm:DermDiseaseType $\sqsubseteq$ dolce:abstract-quality

Table 4.14: A comparison of the ${}^{s}\mathcal{O}_m$ and ${}^{t}\mathcal{O}_m$ for the raw interchangeability and reasoning.

Source and target ontologies	${}^{s}\mathcal{M}_d$ **to** ${}^{s}\mathcal{M}_i$ **links**	**Domain entities**	**Raw interchangeability**	**Reasoning time (s)**
Interchange a foundational ontology for another				
naive_animal-dolce	43	438		75
naive_animal-gfo		452	25.58%	146
ontoderm-dolce	14	301		19
ontoderm-bfo		308	28.57%	0.5
Interchange a module for a leaner module				
sceneOntology-dolce	18	246		0.9
sceneOntology-dolce-el		246	100%	0.5
sao-bfo	66	809		0.7
sao-bfo-continuant		809	100%	0.4
Interchange a top-domain ontology for another				
umlssn-biotop	311	714		300
umlssn-gfo-bio		863	22.18%	0.3
dco-biotop	399	1446		1832
dco-gfo-bio		1650	24.06%	0.2

containing an entity from the DOLCE ${}^{s}\mathcal{O}_m$ exists in the `ontoderm-bfo` module.

For the next set of modules, for interchanging a module for a leaner module, both ontologies had a raw interchangeability of 100%; the ontologies were able to be completely interchanged because there were mappings between the ${}^{s}\mathcal{M}_i$ and ${}^{t}\mathcal{M}_i$ for all entities. DOLCE and DOLCE-EL have the same entities, and BFO and BFO-Continuant also have the same entities, so they were all mapped.

Lastly, for the set of modules for interchanging top-domain modules, they had a raw interchangeability of between 22-24%. They were 28 mappings available between the top-domain ontologies, BioTop and GFO-Bio but the ${}^{s}\mathcal{M}_d$ to ${}^{s}\mathcal{M}_i$ links were high, 311 and 399 for the `umlssn` and `dco` top-domain ontologies, causing the lower raw interchangeability values. Thus, for all ontologies in the set, some interchangeability can be achieved using SUGOI-Gen.

Comparing the number of domain entities in the ${}^{s}\mathcal{O}_m$ and ${}^{t}\mathcal{O}_m$, we note that there are extra domain entities in the ${}^{t}\mathcal{O}_m$. This is because a number of ${}^{s}\mathcal{M}_i$ entities have been added to the ${}^{t}\mathcal{O}_m$ when on-the-fly subsumption occurs (recall the biotop:bio_molecular_process entity that is added to the CELDA ${}^{t}\mathcal{O}_m$ in Figure 4.11). Next, we analyse the reasoning for the modules. All of the interchanged modules except the `naive_animal` have an improved time taken for reasoning after interchangeability (see Table 4.15). In some cases, the improvement is considerable such as the case of `umlssn-biotop` where the interchanged ontology has a reasoning time of just 0.3 seconds compared to the source ontology that has a reasoning time of 5 minutes. Another case is the `dco` interchanged ontology with a reasoning time of 0.2 seconds compared to the source ontology that has a reasoning time of 30.5 minutes.

Table 4.15: The metrics for the ${}^{s}\mathcal{O}_m$ and ${}^{t}\mathcal{O}_m$ ontologies; Coh. = cohesion, AR = attribute richness, IR = inheritance richness, IMD = Intra-module distance.

Source and target ontologies	**Size**	**Atomic size**	**No. of axioms**	**IMD**	**Coh.**	**AR**	**IR**
Interchange a foundational ontology for another foundational ontology							
naive_animal-dolce	545	6.82	3085	186156	0.017	2.62	3.1
naive_animal-gfo	598	6.26	3104	292913	0.02	2.37	3.19
ontoderm-dolce	408	5.14	1470	313977	0.04	0.76	3.16
ontoderm-bfo	347	4.42	1217	345117	0.06	0.51	3.31
Interchange a module for a leaner module							
sceneOntology-dolce	353	5.57	1298	150432	0.04	1.09	4.53
sceneOntology-dolce -el	353	5.13	1221	151118	0.04	0.89	4.34
sao-bfo	848	7.82	8037	2108127	0.04	0.45	2.9
sao-bfo-continuant	830	7.86	7931	1990462	0.04	0.46	2.9
Interchange a top-domain ontology for another top-domain ontology							
umlssn-biotop	1119	7.56	5795	4664881	0.04	2.97	4.00
umlssn-gfo-bio	1105	7.10	5360	3262657	0.04	2.57	4.39
dco-biotop	1851	4.52	8925	297579	0.005	1.42	3.21
dco-gfo-bio	1802	4.11	8470	383592	0.006	1.09	3.13

We now inspect the ontologies to assess whether interchangeability has an effect on the modularity metrics by comparing the ${}^{s}\mathcal{O}_m$ metrics to the ${}^{t}\mathcal{O}_m$ metrics. The metrics are presented in Table 4.15. For modularity, since Definition 6 from Section 3.1 states that "A *Module* M is a subset of a source ontology O, $M \subset O$, either by abstraction, removal or decomposition...", smaller metrics indicate a favourable module. When interchanging a foundational ontology for another foundational ontology, as in the `naive_animal` and `ontoderm` ontologies, this could cause an increase or decrease in the metrics for the module. For the `naive_animal` ontology, all the numerical metrics had increased except the atomic size and the attribute richness because the GFO module is larger than the DOLCE module. For the `ontoderm` ontology, when interchanging from DOLCE to BFO, conversely most of the numerical metrics had decreased, except the intra-module distance and cohesion; thus the metrics had improved for modularity. Interchanging a module for a leaner module should results in improved metrics for modularity.

For the `SceneOntology`, after the interchange, the atomic size, number of axioms, attribute richness, and inheritance richness has decreased. However, the intra-module distance has increased, meaning that the entities moved further apart due to the removal of some of the relations between classes. A larger intra-module distance might promote tool processing since the ontology module would have fewer relations, be less-expressive and have a smaller number of axioms while a smaller intra-module distance means that the classes are closer together and easier to traverse through and this might promote human understanding. For the `SAO` ontology, after the interchange,

all the metrics have decreased except the atomic size and the attribute richness. This is because the number of classes has decreased in the module, but the number of data properties are the same. For the next use-case, interchanging a top-domain ontology for another top-domain ontology, there is a decrease in all the metrics except the inheritance richness value meaning that the target ontology has a higher number of subclasses per class than the source ontology. Hence interchanging top-domain modules for the set in question is indeed favourable.

We now return to the questions regarding interchangeability posed at the onset of the section.

1. If O_A is linked to a module O_X, is it feasible to interchange it to be linked to a different module O_Y?
   It is feasible to interchange a module within an ontology for another module. However, the success of the interchangeability depends on the number of mappings that are available between the source and target modules

2. Does interchanging ontology O_A between O_X and O_Y have an impact on the quality of the modules in the set?
   Interchangeability does have an impact on the metrics, depending on whether the module that has been interchanged is smaller or larger (DOLCE vs. GFO), more-expressive or less- expressive (DOLCE vs. DOLCE-EL), etc., this could impact certain module metrics, and the ontology developer may consider interchangeability to assess which module is the best fit for an application.

3. Does interchanging ontology O_A between O_X and O_Y have an impact on the time taken for reasoning?
   The interchangeability has a significant impact on the time taken for reasoning. A comparison of the reasoning times for the source and target modules reveals that, for the set of modules used in this experiment, the reasoning times were greatly improved.

The investigation of ontology interchangeability for ontology modules that we conducted revealed that we can successfully and automatically interchange modules within a system, thanks to the SUGOI-Gen tool. The experiment with SUGOI-Gen reveals that interchanging modules with SUGOI-Gen could have a positive impact on the modularisation metrics and reasoning processing times of a module and that it could aid the user in selecting the best candidate module for a use-case.

4.4 Ontology modularisation techniques

In Section 2.6, the techniques for modularisation that were discussed include traversal, graph partitioning, locality-based, abstraction, and expressiveness. From the summary of existing techniques for modularity we note that: 1) the abstraction techniques have no tool support at present, and some of the abstraction techniques are tailored towards conceptual data modules and not ontologies, and 2) there is limited tool support for language simplification techniques whereby OWL EL profile

modules can be generated using Protégé v4.3 [110]. In Section 3.6.2.3, from the classification of ontology modules, it was found that tool-based support is lacking for generating various module types. To solve this problem of insufficient tools to automatically generate modules, we have formalised various definitions for abstraction and expressiveness with regard to modularity, designed new algorithms, and developed the Novel Ontology Modularisation SoftwAre (NOMSA). NOMSA implements five new algorithms to automatically generate modules. In the remainder of this Section, we first introduce the five algorithms of NOMSA. This is followed by illustrative examples demonstrating how the algorithms work, and the implementation of the NOMSA and comparison with other modularisation tools. NOMSA is then experimentally evaluated with a set of ontologies to create modules, and the resultant modules are evaluated. The work in this section has been submitted for publication already [83].

4.4.1 New modularisation algorithms

In this section, we present five new algorithms which cover the semantic-based abstraction and language simplification techniques to generate modules of the following types: axiom abstraction, entity type abstraction, high-level abstraction, and weighted abstraction (T9-T12), and feature expressiveness (T14).

Axiom abstraction Axiom abstraction generates a module without complex relations between entities; therefore, the technique decreases the horizontal structure of the ontology and make it a bare taxonomy. Axiom abstraction is formally defined as follows:

Definition 7 (Axiom Abstraction) *Let $\mathcal{O}$, $\mathcal{O}'$ be two ontologies, $\mathcal{S} = \{\alpha_1, ..., \alpha_k\}$ a set of axioms involving at least two classes or a class and a data type, and either at least one object property or data property from $\mathcal{O}$ (i.e., GCIs). We say that $\mathcal{O}'$ is an* axiom abstraction *module of $\mathcal{O}$, if $\mathcal{O}' \cup \mathcal{S} = \mathcal{O}$ such that there exists no element of $\mathcal{S}$ in $\mathcal{O}'$ (i.e., $\mathcal{S} \cap \mathcal{O}' = \emptyset$), hence, $\mathcal{O}' \subset \mathcal{O}$.*

Algorithm 3 (AxAbs) generates axiom abstraction modules. For instance, if $\mathsf{Professor} \sqsubseteq \exists \mathsf{teaches.Course}$ is an axiom in $\mathcal{O}$ and this axiom is in set $\mathcal{S}$ (as it is not a simple subsumption between named classes nor is it a declaration of the Professor class; line 4 of AxAbs), then the axiom will be removed (lines 8-10) resulting in module $\mathcal{O}'$ that will contain only the classes Professor and Course.

Vocabulary abstraction Applying this abstraction to an ontology generates a module where a certain vocabulary element is removed from the ontology. Vocabulary abstraction is formally defined as follows:

Definition 8 (Vocabulary abstraction) *Let $\mathcal{O}$, $\mathcal{O}'$ be two ontologies, $\mathcal{C} = \{C_1, ..., C_k\}$ the set of classes in $\mathcal{O}$, $\mathcal{OP} = \{OP_1, ..., OP_k\}$ the set of object properties in $\mathcal{O}$, and $\mathcal{DP} = \{DP_1, ..., DP_k\}$ the set of data properties in $\mathcal{O}$. We say that $\mathcal{O}'$ is a*

Algorithm 3 Axiom abstraction to compute module M from an ontology O (AxAbs).

Input Ontology O
Output Module M
for all $axiom \in O$ **do**
    if $axiom.type == subclass\_axiom$ **or** $axiom.type == declaration\_axiom$ **then**
        $cExpression \leftarrow axiom.getNestedClassExpressions()$
        $cExpressionSet \leftarrow cExpressionSet + cExpression$ ▷ $cExpressionSet$ is a data structure where we store all the class expressions
        for all $cExpression \in cExpressionSet$ **do**
            if $cExpression.type \neq class$ **then**
                remove $axiom$
            end if
        end for
    else
        remove $axiom$
    end if
end for
$M \leftarrow O$

vocabulary abstraction *module of* $\mathcal{O}$*, if* $\mathcal{O}' \cup \mathcal{C} = \mathcal{O}$ *or* $\mathcal{O}' \cup \mathcal{OP} = \mathcal{O}$ *or* $\mathcal{O}' \cup \mathcal{DP} = \mathcal{O}$ *such that there exists no element of* $\mathcal{C}$*,* $\mathcal{OP}$*, or* $\mathcal{DP}$ *in* $\mathcal{O}'$ *(i.e.,* $\mathcal{C} \cap \mathcal{O}' = \emptyset$*,* $\mathcal{OP} \cap \mathcal{O}' = \emptyset$*,* $\mathcal{DP} \cap \mathcal{O}' = \emptyset$*), hence,* $\mathcal{O}' \subset \mathcal{O}$*.*

Algorithm 4 Vocabulary abstraction to compute module M from an ontology O (VocAbs).

Input Ontology O, Entity e, EntityType t ▷ e is a named class, object property, or data property and t is the type of e in O
Output Module M
if $t ==$ class **then**
    remove e
else if $t ==$ object property **then**
    remove e
else if $t ==$ data property **then**
    remove e
else if $t ==$ individual **then**
    remove e
end if
$M \leftarrow O$

Algorithm 4 (VocAbs) generates vocabulary abstraction modules. For instance, consider the Subcellular Anatomy Ontology (SAO) domain ontology that is linked to the BFO foundational ontology, but the developers want to change that to the

DOLCE foundational ontology. Since the domain ontology does not contain any object properties, one could be interested in removing the object properties from the DOLCE-aligned version of the SAO ontology using the vocabulary abstraction algorithm.

High-level abstraction High-level abstraction generates a module where entities at a higher level are regarded more important than others. High-level abstraction is formally defined as follows, specifying the notion of *depth* in a taxonomy first:

Definition 9 *(Depth) Let $\mathcal{O}$ be an ontology. A* depth *in the hierarchy of $\mathcal{O}$ represents the subclass distance between the hierarchy's top-level entity and a given entity, e.g.,* depth *1 refers only to the top-level classes,* depth *2 refers to the top 2 layers of classes (the parent classes and its direct subclasses) and so on.*

Definition 10 *(High-level abstraction) Let $\mathcal{O}$, $\mathcal{O}'$ be two ontologies, n be a* depth *where n is an integer ≥ 1. We say that $\mathcal{O}'$ is a* High-level abstraction *of $\mathcal{O}$, if the entities with a* depth $> n$ *are removed, hence, $\mathcal{O}' \subset \mathcal{O}$.*

Algorithm 5 (HLAbs) generates high-level abstraction modules. For instance, in the ROMULUS repository [88], the GFO-abstract-top module of the GFO ontology is based on the Abstract Top Level layer of GFO which contains mainly two meta-categories: set and item. To generate this module automatically, one could use HLAbs with the GFO ontology and set the depth as 2 (see Figure 4.14).

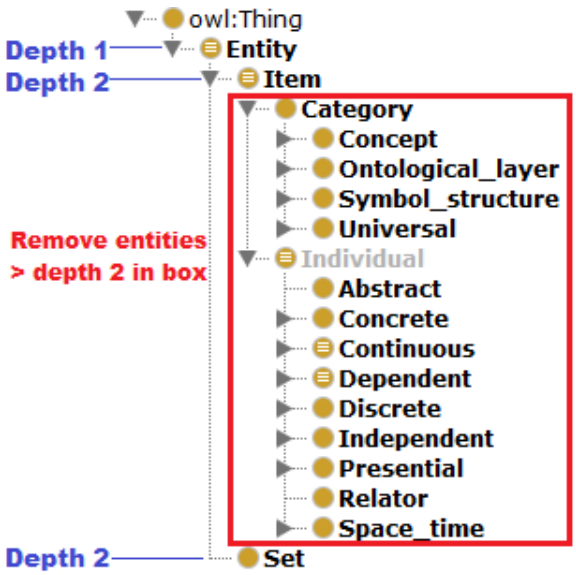


Figure 4.14: Generating a high-level abstraction module with depth = 2 from the GFO ontology.

Weighted abstraction Weighted abstraction deals with removing entities from an ontology that are deemed less important than others by assigning weight to the classes, properties, and individuals in an ontology. Our approach for determining entities that are more important than others is based on looking at entities that other entities are highly dependent on. For instance, in the pizza ontology, the class TomatoTopping is the most widely used, being referenced 61 times by other entities. Weighted abstraction is formally defined in Definition 13, availing of the notions of relative and absolute thresholds:

Algorithm 5 High-level abstraction to compute module M from an ontology O (HLAbs).

```
1: Input Ontology O, LevelNumber
2: Output Module M
3: oldSet ← ontology.getAxioms()
4: for all class ∈ O do
5:     if class.superclasses() is empty then
6:         topLevelClassSet ← topLevelClassSet + class
7:     end if
8: end for
9: counter ← 0
10: while counter ≠ levelNumber do
11:     for all topclass ∈ topLevelClassSet do
12:         newset ← newset + topclass.getAxioms()
13:         if topclass.subclasses() ≠ empty then
14:             temp ← topclass.subclasses()
15:         end if
          ▷ Repeat lines 3 - 13 for object properties, data properties, and instances.
16:         topLevelClassSet.clear()
17:         topLevelClassSet ← temp
18:     end for
19: end while
20: for all axiom ∈ oldAxioms do
21:     if newAxioms does not contain axiom then
22:         ontology.remove(axiom)
23:     end if
24: end for
25: M ← O
```

Definition 11 *(Relative threshold) Let $\mathcal{O}$ be an ontology, $\mathcal{E} = \{\mathcal{E}_1, ..., \mathcal{E}_k\}$ be the set classes, object properties, data properties, and individuals in $\mathcal{O}$. A* Relative threshold *θ is a percentage value to decide which elements of $\mathcal{E}$ are to be removed from $\mathcal{O}$. Each element of $\mathcal{E}$ is weighted according to the number of axioms it participates in and ordered according to a position p. If $p(\mathcal{E}_i) < \theta.|\mathcal{O}|$, $\mathcal{E}_i$ is removed from $\mathcal{O}$.*

Definition 12 *(Absolute threshold) Let $\mathcal{O}$ be an ontology, $\mathcal{E} = \{\mathcal{E}_1, ..., \mathcal{E}_k\}$ be the set classes, object properties, data properties, and individuals in $\mathcal{O}$. An* Absolute threshold *θ is a numerical value to decide which elements of $\mathcal{E}$ are to be removed from $\mathcal{O}$. Each element of $\mathcal{E}$ is weighted according to the number of axioms it participates in and ordered according to a position p, If $p(\mathcal{E}_i) < \theta$, $\mathcal{E}_i$ is removed from $\mathcal{O}$.*

Definition 13 *(Weighted abstraction) Let $\mathcal{O}$, $\mathcal{O}'$ be two ontologies, $\mathcal{E} = \{\mathcal{E}_1, ..., \mathcal{E}_k\}$ be the set classes, object properties, data properties, and individuals in $\mathcal{O}$. We say that $\mathcal{O}'$ is a* Weighted abstraction *of $\mathcal{O}$, if elements of $\mathcal{E}$ are removed from $\mathcal{O}$ according to some* Absolute threshold *or* Relative threshold*, hence, $\mathcal{O}' \subset \mathcal{O}$.*

Algorithm 6 (WeiAbs) generates weighted abstraction modules. For instance, consider modularising the BioTop ontology [14] using the weighted abstraction algorithm with $\theta = 4$. For the class Phosphate, it has two referencing axioms: a declaration axiom and the axiom Phosphate $\sqsubseteq$ InorganicMolecularEntity. Since the number of referencing axioms ($p(\mathcal{E}) = 2$) is less than the absolute threshold value (4), the Phosphate class is removed from the ontology.

Algorithm 6 Weighted abstraction to compute module M from an ontology O (WeiAbs).

1: **Input** Ontology O, thresholdPercentage, Weight_array, Class_array
2: **Output** Module M
3: $i \leftarrow 0$
4: **for all** $class \in O$ **do**
5: $\quad Weight\_array \leftarrow Num\_of\_referencing\_axioms$
6: $\quad Class\_array(i) \leftarrow class$
7: $\quad i \leftarrow i + 1$
8: **end for**
9: Sort($Weight\_array$,$Class\_array$) ▷ Sort $Weight\_array$ from low to high, with $Class\_array$ corresponding to $Weight\_array$
10: $limit \leftarrow thresholdPercentage * |Class\_array|$
11: **for** $i \leftarrow 0, limit$ **do**
12: $\quad$ remove $Class\_array(i)$ from O
13: **end for** ▷ repeat lines 4-12 for ObjectProperty_array, DataProperty_array, Individual_array
14: $M \leftarrow O$

Feature expressiveness Feature expressiveness modules deal with removing some axioms of the ontology based on the language features, e.g., cardinality constraints, disjointness, object property features etc. By manipulating complex constructs of the ontology language features, the feature expressiveness algorithm results in a simplified model of the ontology. We have designed 7 rules for this. The algorithm takes these 7 rules, and removes them, from the least important to the most important. At each rule removal, a 'layer' of the ontology is produced where that ontology is

represented in a language of lower expressivity than the previous layer. Once the algorithm is complete, seven modules (layers) are produced, each having a lower level of expressivity than the previous. Feature expressiveness is formally defined as follows:

Definition 14 *(Feature expressiveness) Let $\mathcal{O}$, $\mathcal{O}'$ be two ontologies, $\mathcal{R} = \{\mathcal{R}_1, ..., \mathcal{R}_k\}$ a set of rules describing various OWL language features. We say that $\mathcal{O}'$ is a* Feature expressiveness *of $\mathcal{O}$, if we remove axioms that follow $\mathcal{R}$ from $\mathcal{O}$, hence $\mathcal{O}' \subset \mathcal{O}$*

Rules with higher values are deemed more important than others, so the rules with the lower levels are applied first to remove those less-important entities. We decided to assign lower points to those OWL ontology features that serve to restrict and refine entities such as cardinality, domain and range, and property characteristics. We assign higher points to disjointness, equality and inequality, and complex classes since they can be used in conjunction to define new classes. This is, in a way, subjective and motivated by the modelling perspective on language features. It is conceivable, and possible if desired, to assign different weights to them; e.g., such that they are motivated by, and aligned with, the various OWL profiles.

In the notation of the rules that follow, we use the usual Description Logics notation, and we assume a Semantic Web context with ontologies represented in at most OWL 2 DL. The classes C, D, E are concept descriptions, which could be simple (named classes) or complex, R, S object properties, U, V data properties, and a, b, c individuals in the vocabulary of the ontology, with n a non-negative integer, and all declared knowledge adheres to the OWL 2 DL syntax.

R1: Qualified cardinality Cardinality allows one to specify the number of individuals that will be involved in an interaction between a class and an object or data property. We weigh cardinality in an ontology with 1 point. Rules: Remove axioms of the following axiom patterns, if present:

$C \sqsubseteq \leq n\ R.D$ $\quad$ $C \sqsubseteq \leq n\ U.D$
$C \sqsubseteq \geq n\ R.D$ $\quad$ $C \sqsubseteq \geq n\ U.D$
$C \sqsubseteq = n\ R.D$ $\quad$ $C \sqsubseteq = n\ U.D$

R2: Domain and range Domain and range relate an object property to a class by restricting that the object property must belong to a certain class. We weigh domain and range with 2 points. Rule: Remove axioms of the following axiom patterns, if present:

$\exists R.\top \sqsubseteq C$
$\top \sqsubseteq \forall R.\ C$

R3: Property characteristics Object property characteristics are used to further refine object properties with logical features. We weigh property characteristics with 3 points. Rule: Remove axioms of the following axiom patterns, if present (in $\mathcal{SROIQ}$'s shorthand notation):

$\mathsf{Func}(R)$	$\mathsf{Trans}(R)$
$\mathsf{Func}(R^-)$	$\mathsf{Ref}(R)$
$\mathsf{Sym}(R)$	
$\mathsf{Asym}(R)$	$\mathsf{Irr}(R)$

R4: Disjointness Class disjointness in an ontology means that the classes cannot have any instances that are the same. We weigh it with 4 points.

Rule: Remove axioms of the following axiom patterns, if present

$C \sqcap D \sqsubseteq \bot$

$C \sqsubseteq \neg\, D$

R5: Assertions Assertions are used to describe individuals in an ontology by asserting membership to a class, data property, or object property. We weigh it with 5 points. Rule: Remove axioms of the following axiom patterns, if present:

a: C

R(a,b)

U(a,c)

R6: Equivalence and equality Equivalence can be asserted for classes and properties, and (in)equality can be asserted for individuals. We weigh equality and inequality in ontologies with 6 points. Rule: Remove axioms of the following axiom patterns, if present:

$C \equiv D$	$a = b$
$R \equiv S$	$a \neq b$
$U \equiv V$	

R7: Complex classes Complex classes are used when a class is defined or described using a combination of named classes, object properties, and data properties. We weigh complex classes as the most important of the seven rules, 7 points. Rule: Remove axioms of the following axiom patterns, if present:

$C \sqsubseteq D \sqcap E$	$C \sqsubseteq D \sqcup E$
$C \equiv D \sqcap E$	$C \equiv D \sqcup E$

Algorithm 7 (FeatExp) uses these seven rules to generate feature expressiveness modules. For instance, when modularising the Koala ontology, for rule 1 concerning cardinality, the axiom $\mathsf{Animal} \sqsubseteq \leq 1\ \mathsf{hasGender}.\top$ is removed, whilst keeping Animal and hasGender in the ontology.

Finally, note that the algorithms are linear for AxAbs, VocAbs, and WeiAbs, and quadratic for HLAbs and FeatExp.

4.4.2 Illustration of the algorithms

We now illustrate the algorithms introduced in the previous section with a sample ontology, where we focus on the WeiAbs and FeatExp algorithms. Consider the

Algorithm 7 Feature Expressiveness to compute module M from an ontology O (FeatExp).

```
1: Input Ontology O, ruleSet {r_1, ..r_7}
2: Output Module moduleSet {M_1, ..M_7}
3: i ← 1
4: for all axiom ∈ O do
5:     for all r_i ∈ ruleSet do
6:         if axiom.type is r_i then
7:             remove axiom
8:         end if
9:     end for
10:    M_i ← O
11:    i ← i + 1
12: end for
```

following axioms in a toy Burger ontology in Figure 4.15 (entity declaration axioms omitted). The complete Burger ontology with declarations in OWL functional syntax is shown in Appendix B.

4.4.2.1 Weighted abstraction module

To generate a weighted abstraction module, we apply WeiAbs. Let us assume we wish to create a module whereby we remove 25% of the entities. To achieve this, we set the threshold value to 25%. First, we apply lines 4-8 of WeiAbs, where we weigh each class in the ontology with its number of referencing axioms, and we store both the number of referencing axioms and each class in two arrays with corresponding indices. For line 9 of the algorithm, we sort the weight array values from low to high and the class array such that it matches the weight array. In line 10, a limit variable is calculated as the product of the threshold percentage (.25) and the number of classes in the ontology (21) which is rounded off to a value of 5. In lines 11-13, the classes with the 5 lowest values are removed, as displayed in Table 4.16. The classes in bold font are the 25% of the classes that are deemed less-important than the rest and are to be removed due to having the least number of referencing axioms in the ontology.

Table 4.16: The classes of the burger ontology with the number of referencing axioms. Those in bold font are the classes to be removed for the resulting module.

WhiteBun	**2**	Medium	3	Patty	4
Customer	**2**	Lettuce	3	BeefBurger	4
Cheese	**2**	HealthyBurger	3	BurgerBun	4
Sauce	**2**	BeefPatty	3	Hamburger	4
Chef	**2**	Tomato	3	Filling	5
WholeWheatBun	**2**	WellDone	3	PattyCook	6
Person	3	Rare	3	Burger	7

$$\text{BeefPatty} \sqsubseteq \text{Patty} \quad (1)$$
$$\text{Beefburger} \equiv \text{HamBurger} \quad (2)$$
$$\text{Beefburger} \sqsubseteq \text{Burger} \quad (3)$$
$$\text{Cheapburger} \sqsubseteq \leq 1\ \text{hasFilling.Filling} \quad (4)$$
$$\text{Cheapburger} \sqsubseteq \text{Burger} \quad (5)$$
$$\text{Cheese} \sqsubseteq \text{Filling} \quad (6)$$
$$\text{Chef} \sqsubseteq \text{Person} \quad (7)$$
$$\text{Customer} \sqsubseteq \text{Person} \quad (8)$$
$$\text{HamBurger} \equiv \text{Beefburger} \quad (9)$$
$$\text{HamBurger} \sqsubseteq \text{Burger} \quad (10)$$
$$\text{HealthyBurger} \sqsubseteq \forall \text{hasFilling.}(\text{Lettuce} \sqcup \text{Tomato}) \quad (11)$$
$$\text{HealthyBurger} \sqsubseteq \text{Burger} \quad (12)$$
$$\text{Lettuce} \sqsubseteq \text{Filling} \quad (13)$$
$$\text{Medium} \sqsubseteq \text{PattyCook} \quad (14)$$
$$\text{PattyCook} \equiv \text{Medium} \sqcup \text{Rare} \sqcup \text{WellDone} \quad (15)$$
$$\text{Rare} \sqsubseteq \text{PattyCook} \quad (16)$$
$$\text{Sauce} \sqsubseteq \text{Filling} \quad (17)$$
$$\text{Tomato} \sqsubseteq \text{Filling} \quad (18)$$
$$\text{WellDone} \sqsubseteq \text{PattyCook} \quad (19)$$
$$\text{WhiteBun} \sqsubseteq \text{BurgerBun} \quad (20)$$
$$\text{WholeWheatBun} \sqsubseteq \text{BurgerBun} \quad (21)$$
$$\text{WholeWheatBun} \sqsubseteq \neg \text{WhiteBun} \quad (22)$$
$$\text{Func(hasBun)} \quad (23)$$
$$\exists\ \text{hasBun.}\top \sqsubseteq \text{Burger} \quad (24)$$
$$\top \sqsubseteq \forall \text{hasBun.BurgerBun} \quad (25)$$
$$\exists\ \text{hasPatty.}\top \sqsubseteq \text{Burger} \quad (26)$$
$$\top \sqsubseteq \forall \text{hasPatty.Patty} \quad (27)$$
$$\exists\ \text{hasPattyCook.}\top \sqsubseteq \text{Patty} \quad (28)$$
$$\top \sqsubseteq \forall \text{hasPattyCook.PattyCook} \quad (29)$$
$$\text{MarthasBurger} \neq \text{MyBurger} \quad (30)$$
$$\text{MarthasBurger} : \text{Burger} \quad (31)$$
$$\text{MyBurger} : \text{Beefburger} \quad (32)$$
$$\text{MyBurger} : \text{Burger} \quad (33)$$
$$\text{MyBurger} : \text{Beefburger} \quad (34)$$
$$\text{ChefRose} : \text{Chef} \quad (35)$$
$$\text{cookedBy(MyBurger, ChefRose)} \quad (36)$$

Figure 4.15: The burger ontology to which the algorithms are applied; see text for details.

4.4.2.2 Expressiveness feature module scenario

For the expressiveness feature module, each rule is applied according to the order in FeatExp. For each axiom in the ontology, lines 4-9 of the algorithm are executed, therefore each rule is applied as follows.

- R1: Cardinality: Applying this rule results in the removal of Axiom 4 from the burger ontology.
- R2: Domain and range Applying this rule results in the removal of Axioms 24-29.
- R3: Property characteristics: Applying this rule results in the removal of Axiom 23.
- R4: Disjointness: Applying this rule results in the removal of Axiom 22.
- R5: Assertions: Applying this rule results in the removal of Axioms 31-36.
- R6: Equivalence and equality: Applying this rule results in the removal of Axioms 2, 9, 15, and 30.
- R7: Complex classes: Applying this rule results in the removal of Axiom 11.

The resulting ontology module is simpler, having 16 logical axioms (compared to the original 32 logical axioms). All of the classes, properties, and individuals of the original ontology are preserved in the module; however, the hierarchy of entities differ from that of the original ontology since some language features are no longer present.

4.4.3 NOMSA Modularisation tool

From the existing literature on modularisation described in Section 2.6, most of the tools that exist are partitioning tools and modularisation tools that create locality-based and query-based modules. Techniques have been described for abstraction but there are no software tools to apply them to ontologies, and there are limited techniques for language simplification. Furthermore, in the classification experiment of modules in Section 3.6, it was found that there was a heavy reliance on using manual methods for module creation and that it may be possible to automate some of these techniques.

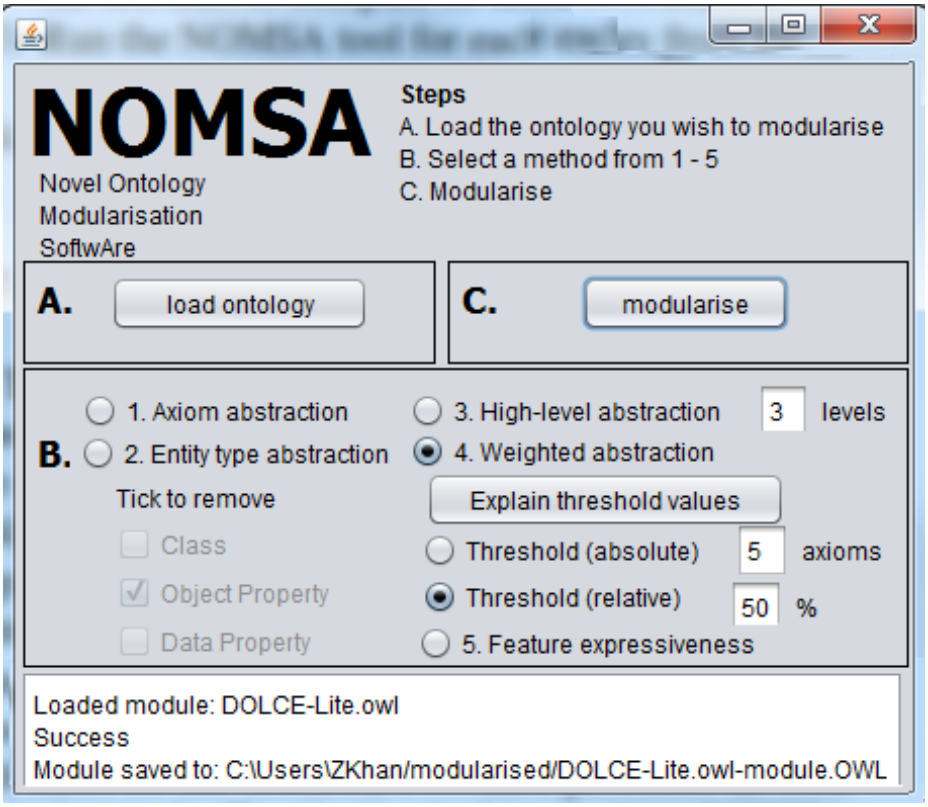


Figure 4.16: The interface of NOMSA.

We have designed new algorithms for modularisation techniques that were lacking and implemented them in NOMSA. NOMSA can be used to modularise ontologies using abstraction and expressiveness algorithms. NOMSA allows the user to upload an ontology (including its imports), and select one of five approaches (AxAbs, VocAbs, HLAbs, WeiAbs, or FeatExp) to modularise it. A module is then generated. NOMSA is a stand-alone Java application and can be downloaded from `http://www.thezfiles.co.za/modularisation`. A screenshot of the interface of NOMSA is displayed in Figure 4.16.

4.4.4 Experimental evaluation

The purpose of the experiment is to evaluate all the algorithms implemented in NOMSA to check if it performs modularisation on a set of ontologies, and how well it modularises, i.e., the quality of the generated modules. We use the default parameters for all the algorithms as shown in Figure 4.16. We evaluate NOMSA in two ways. First, we compare its features to existing modularisation tools. Then we evaluate NOMSA's modules by analysing their evaluation metrics, e.g., relative size, intra-module distance, etc.

4.4.4.1 Materials and methods

The method for the experiment is as follows:

1. Take a set of ontologies.
2. Run each modularisation tool's algorithm with a subset of 10 randomly selected ontologies from the test files to compare its features to NOMSA: level of interaction, algorithm complexity, technique, and time taken for processing.
3. Run the NOMSA tool for each ontology from the test files for all five algorithms.
4. Run the Tool for Ontology Module Metrics (TOMM) for NOMSA's modules to acquire evaluation metrics for the modules.
5. Conduct an analysis from the evaluation metrics for each module.

The materials used for the experiment were as follows: Protégé v4.3 [110], SWOOP [73], OWL module extractor [30], PROMPT [113], PATO [149], TaxoPart [58], NOMSA tool using the default parameters for all algorithms, TOMM metrics tool, and a set of 114 ontologies. The set of 114 ontologies were from various domains, and extracted from a set of 338 ontologies described elsewhere [49]. The first 114 ontologies from the initial set were selected to be used for our tests. Our tests were conducted on a 3.00 GHz Intel Core 2 Duo PC with 4 GB of memory running Windows 7 Enterprise. All the test files are available at `http://www.thezfiles.co.za/modularisation`.

4.4.4.2 Results

The results of comparing NOMSA's algorithm features to the existing modularisation tools are shown in Table 4.17. For most of the features, NOMSA performs as well as or better than the other tools, with the benefit of full automation of the process. For the level of interaction, NOMSA is automatic. NOMSA includes the most number of algorithms in a tool (five) compared to the other tools. For techniques, NOMSA used semantic-based abstraction and language simplification techniques; semantic-based abstraction has not been applied in other tools to-date. NOMSA's algorithms have a processing time of between 2-4 seconds for modularisation for the set of 114

Table 4.17: A comparison of three features of several modularisation tools against NOMSA and the average running times of the respective algorithms for the test set of ontologies (excluding the time of manual modularisation tasks such as loading the ontology and setting the parameters).

	Level of interaction	**Algorithm complexity**	**Technique**	**Time (seconds)**
SWOOP Algorithm 1	semi-automatic	quadratic	locality-based	1
SWOOP Algorithm 2	automatic	quadratic	graph-partition	6
OWL module extractor	semi-automatic	quadratic	locality-based	1
PROMPT	user-driven	unknown	query-based	-
PATO	automatic	unknown	graph-partition	16
Protégé Algorithm 1	semi-automatic	unknown	locality-based	1
Protégé Algorithm 2	automatic	unknown	language-based	1
Protégé Algorithm 3	semi-automatic	unknown	locality-based	1
Protégé Algorithm 4	semi-automatic	unknown	language-based	1
TaxoPart	automatic	linear	graph-partition	15
NOMSA AxAbs	automatic	linear	semantic-based abstraction	3
NOMSA VocAbs	automatic	linear	semantic-based abstraction	2
NOMSA HLAbs	automatic	quadratic	semantic-based abstraction	2
NOMSA WeiAbs	automatic	linear	semantic-based abstraction	4
NOMSA FeatExp	automatic	quadratic	language-based	3

ontologies. The locality-based algorithms have the quickest time (1 second) while partitioning algorithms take longer (6-16 seconds).

The other tools also require the user to manually save the modules after module generation, while this is done automatically with NOMSA and is included in the time taken. It was not possible to test PROMPT for time taken because it was completely user-driven. We do not compare the resultant modules of the other modularisation tools because they all generate different types of modules and their underlying techniques differ.Some of the require an input seed entity for generation (SWOOP [73] and OWL module extractor [30]), others are heavily user-driven (PROMPT [113] and Protégé [110]), and others generate a set of partition modules (PATO [149] and TaxoPart [58]).

Concerning the quality of the NOMSA-generated modules, all 114 ontologies were successfully modularised using all five algorithms in NOMSA, and their metrics were generated using the TOMM module metrics tool. The metrics for the modules are displayed in Table 4.18 and we discuss the notable metrics here.

This means that the entities in the module are to that degree closer than in the original ontology. The rest of the algorithms have values less than 1, meaning that the entities are to that degree further away than in the original ontology. For instance, removing axioms that describe equality between classes simplifies the expressivity of the module for easier tool processing, but increases the distance between classes.

The average time taken in modularising the set of modules for all algorithms is less than 5 seconds; all five algorithms perform quickly. This includes the time taken for saving the generated module. All five algorithms result in a reduction of the size of the original ontology, ranging from modules that have a relative size of 0.25 (WeiAbs) to modules that have a size of 0.86 (VocAbs). For the relative intra module distance, the modules of HLAbs and WeiAbs have values greater than 1 (20.31 and 3.66 respectively).

According to Table 4.18, all the generated modules are notably different from the source ontologies according to their metrics. In order to determine whether the modules are of good quality, we compare the results obtained from the generated modules to the benchmark dependencies between modularity metrics of the framework for ontology modularity presented in Section 3.7. When comparing WeiAbs and FeatExp modules to the dependencies from the framework, all the metric values for the generated modules correspond with what is expected; these modules are of 'good' quality. We note the following from the comparison of the modules for AxAbs, VocAbs, and HLAbs to the dependencies. For the AxAbs modules, the metrics match the expected metrics for 73 out of the 114 modules; the remaining 41 modules fail to meet 1 out of the 2 expected values. For the VocAbs modules, the metrics match the expected metrics for only 4 out of the 114 modules; the remaining 110 modules fail to meet all out of the 3 expected values. For the HLAbs modules, the metrics match the expected metrics for 16 out of the 114 modules; the remaining 98 modules fail to meet 1 out of the 2 expected values.

Table 4.18: The average values for the metadata for all the generated modules; app. = appropriateness, AR = attribute richness, IR = inheritance richness.

	Size	**Atomic size**	**App.**	**Intra module distance**	**Cohesion**	**AR**	**IR**	**Relative size**	**Relative intra module distance**	**Time**
AxAbs modules	233.04	2.28	0.22	577658.80	0.06	0.54	5.13	0.67	0.66	3.01
VocAbs modules	413.24	3.21	0.23	566255.90	0.06	0.50	5.08	0.86	0.76	2.34
HLAbs modules	174.14	3.36	0.24	96148.77	0.03	0.40	5.17	0.68	20.31	2.41
WeiAbs modules	138.58	3.40	0.30	123491.50	0.08	0.41	2.73	0.25	3.66	4.10
FeatExp modules	237.74	2.30	0.20	462826.50	0.06	0.20	5.13	0.68	0.66	2.67
Original ontologies	431.72	3.56	0.24	577658.80	0.03	0.91	5.11	-	-	-

One of the reasons why the VocAbs and HLAbs modules do not meet all the expected metrics is because one of the metrics is not applicable for this set of modules; the appropriateness value which is supposed to be large (0.75-1) cannot always be met because some of the original ontologies have a size that is less than 167 which means that the module will always be less than 167 and therefore the metric cannot be achieved. From our data, 71 out of the 114 original ontology files had this inapplicable metric for appropriateness. This was the only condition, for this set of test files where a metric was not applicable and this was due to the size of the original ontologies in question. Further investigation using other ontologies may reveal other conditions where a metric may not be applicable.

Therefore, collectively, the algorithms result in some good quality modules according to the dependencies. Upon closer analysis of the modules that don't match the dependencies for good quality, it was found that some of the expected values may depend on the structure of the source ontology. For instance, VocAbs modules are expected to have a large appropriateness value (> 0.75). However, this is only possible where a module has between 167-333 axioms, and in some cases, a source ontology may have fewer axioms than this range, therefore, the metric is inapplicable for certain modules.

4.5 Discussion

The problem that it is unclear how to determine whether a module is a good or bad module due to the lack of evaluation metrics is addressed in this Chapter using the Tool for Ontology Module Metrics (TOMM). The experimental evaluation performed with TOMM in Section 4.2.7, revealed a dependency diagram that shows which metrics values are expected for which module types. A user can then automatically generate metrics for their module, using the TOMM tool, and refer to the dependency diagram to check whether their module is of good quality or not. The TOMM metrics tool can also be used to automatically calculate some of the metadata that is required in the case-study ROMULUS ontologies. These ontologies are annotated with metadata such as ModuleCoverage, which corresponds to the relative size metrics, ModuleCompleteness, which corresponds to the logical completeness metric, etc. It is possible to automatically generate these metrics using the TOMM tool. Thus, an ontology developer need not perform ontology metric calculations manually for annotating a module with relevant metadata. We also note that other metadata values, such as the ModuleType found in the ROMULUS ontologies in the exploratory study in Section 4.1.1 could be easily extracted from our framework in Section 3.7.

Thanks to SUGOI-Gen and the experimental evaluation using a set of modules, we now know that interchangeability with modules is indeed possible. The success of the interchangeability depends on the number of mappings that are available between the source and target modules. For the set of modules used in our experiment, the success of interchangeability ranged from 22% to a 100%. The foundational ontology and top-domain ontology modules had a lower raw interchangeability because only some of the entities could be mapped. For the leaner modules, all their entities were

mappable resulting in the 100% raw interchangeability.

An ontology developer can also gain insight on how a certain foundational, top-domain, or leaner modules could impact the metrics of a module, and which module might be better to use, depending on the developer's desired metrics. For instance, a developer may prefer to have a module with a larger intra-module distance such as the case with the `sceneOntology-dolce-el` module to promote tool processing at the expense of human comprehension, or vice versa. SUGOI-Gen could also be used to swap modules within a system for other ones to improve processing times for reasoning tools. For the set of modules in our experiment, all the modules that were interchanged except one had an improved time for reasoning.

The problem that there are insufficient tools for modularity was demonstrated in the classification of modules in the previous Chapter (Section 3.6.2.3), where we learned that for 9 out of the 14 module types, manual methods were used. Following this, to further explore this problem, we looked at two case-studies on modularisation: the ROMULUS foundational ontologies and the DMOP ontology in Section 4.1. Once again, it was discovered that existing tools were not sufficient for the type of modules that we needed to generate and that manual intervention was necessary. This problem has been solved in this Chapter as follows. Five new algorithms have been designed and implemented into a novel tool for modularisation, NOMSA. The algorithms and new tool, NOMSA, designed for generating abstraction and expressiveness type modules solve the problems of: 1) users' reliance on manual methods for modularity, 2) the lack of abstraction and expressiveness techniques in existing automated tools. Our experiments show that our algorithms can be used to automatically modularise ontologies, thus, it both broadens the scope of the extant set of algorithms for automated modularisation [30, 58, 73, 110, 113, 149] to enable generation of more types of modules, and it refines and realises theory-based approaches, such as presented in [51, 76, 77, 101, 120], so that it is usable by ontology engineers.

The performance of the algorithms is good; the time taken to modularise the ontologies is fast for all five algorithms (under 5 seconds on average). Assessing the quality of the metrics of the modules reveal that for this test set of ontologies, WeiAbs and FeatExp algorithms generate modules of 'good' quality for all its resulting modules according to the expected dependencies. For the remaining three algorithms, they generate some 'good' quality modules but it is not possible to meet the expected metric values for all the resulting modules, for some of the metrics depend on the source ontology. The resulting modules are, however, still an improvement compared to the original ontologies; the sizes of the modules have been reduced considerably, and other metrics such as attribute richness, inheritance richness, etc., are notably different when compared to the original ontologies as displayed in Table 4.18.

4.6 Conclusion

The exploratory studies conducted in this study using existing resources for modularisation revealed what is lacking in current modularisation tools. The existing tools do not satisfy some of the modularisation techniques that were required for the ex-

ploratory study, module management in the form of annotating modules with useful data is lacking, and acquiring some of the metadata for modules in existing repositories, e.g., logical completeness is a manual process which is time-consuming. To assist with automatically generating some of this metadata, we compiled a list of new and existing evaluation metrics. They have been implemented in the TOMM tool to enable scaling up of module evaluation. Our evaluation carried out with 189 modules revealed which metrics work well with which types of modules. It is now possible for an ontology developer to evaluate the quality of a module/set of modules by first classifying its type using the framework for ontology modularity, and then generating its metrics using the TOMM metrics tool. Ontology developers are then able to determine whether their ontology module is of good quality based on comparing the modules metrics to what is expected in the dependency diagram.

To facilitate module management of the swappability of modules, we investigated ontology interchangeability on modules and the impact on their metrics. We presented the design and implementation of the SUGOI-Gen software tool which can be used to interchange a module for another module within a system, covering the four principal scenarios of module interchange: leaner versions of an ontology, foundational and top-domain ontologies, and the outline of ontology pattern swapping, provided that the user uploads the source and mapping files. An experimental evaluation of three different use-cases using six ontologies reveals that some interchangeability is possible and that the success of interchangeability depends on the mappings that are available. The investigation revealed that an ontology developer can gain insight on how a certain foundational, top-domain, or subset modules could have an effect on the metrics, and the processing times for reasoning for modules.

To solve the problem that some techniques for modularisation are lacking, five new algorithms were designed to generate abstraction and expressiveness modules. They have been implemented in the NOMSA tool to modularise ontologies accordingly. The tool is fully-automated for all five algorithms. Our evaluation was carried out on 114 diverse modules. It was revealed that NOMSA does modularise the ontologies and that the tool performance is good (under 5 seconds on average). By inspecting the metrics of NOMSA's generated modules, we note that all five algorithms result in a reduction of size of the original ontology. For some of NOMSA's modules, the intra-module distance is smaller than the original ontology meaning that the entities appear to be closer linked than previously while for others, the intra-module distance is larger. We also checked the quality of the modules using the metrics generated with TOMM metrics tool against the benchmark dependencies between modularity metrics from an existing framework. It was found that the average values for the generated modules of two of the algorithms from our experiment correspond with what is expected hence these are modules of 'good' quality. All the generated modules, however, are notably different from the source ontologies according to their metrics.

Chapter 5

Conclusion and future research

This chapter presents the conclusions about the research and suggestions for future avenues of research. In Section 5.1 we discuss our conclusions and answer our research questions. We complete the chapter in Section 5.2 where we discuss future work.

5.1 Conclusion

In this thesis, we have studied ontology modularisation to solve the problems that: 1) existing techniques are not sufficient for modularisation, 2) a user has no guidance on how to initiate modularisation for an ontology, which type of module to extract, which technique or tool to use, and 3) how to determine if the module is of good quality.

The first problem was solved by performing a classification on a set of ontology modules to determine which techniques are lacking in tools, performing an exploratory study on modularisation with existing resources to determine the issues, and by designing and implementing novel algorithms to perform modularisation. The second problem was solved by identifying dimensions of modularity, classifying a set of modules with dimensions, and linking various dimensions together to create dependencies. This resulted in a framework for modularity which a user can use to guide the modularisation process. The last problem was solved by identifying new and existing evaluation metrics and providing equations for those that did not have any, the development of a tool to compute the metrics for an ontology module, and performing an investigation to determine which metrics can be used to measure which module types.

To solve the problems for modularity, we formulated and answered research questions pertaining to modularity. In the following section, we re-visit these questions and provide answers and our contributions for them.

5.1.1 Research questions

Our main research question proposed in Section 1.4 was as follows: **1. How can one devise a formal foundation for modularity to improve existing modularity**

techniques and results? This was broken down into six sub-questions which we discuss here.

1(a) What are the different types of modules that exist?

We propose that modules are of different types. We have identified and grouped together 14 types of ontology modules (T1-T14). These modules are sub-divided into functional (T1-T5), structural (T6-T8), abstraction (T9-T12), and expressiveness (T13-T14) modules. The types are as follows: T1: Ontology design patterns, T2: Subject domain modules, T3: Isolation branch modules, T4: Locality modules, T5: Privacy modules, T6: Domain coverage modules, T7: Ontology matching modules, T8: Optimal reasoning modules, T9: Axiom abstraction modules, T10: Entity-type modules, T11: High-level abstraction, T12: Weighted modules, T13: Expressiveness sub-language modules, and T14: Expressiveness feature modules. For a full definition of each type of module, with corresponding examples, refer to Section 3.3.

1(b) What are the properties with which we can characterise each module type?

There are 16 properties with which we can characterise ontology modules (P1-P14). These properties exist in isolation in a single module (P1-P8) and in a set of modules (P9-P14). The properties are as follows: P1: Seed signature, P2: Information removal, P3: Abstraction, P3.1: Breadth abstraction, P3.2 Depth abstraction, P4: Refinement, P5: Stand-alone, P6: Source ontology, P7: Proper subset, P8: Imports, P9: Overlapping, P10: Mutual exclusion, P11: Union Equivalence, P12: Partitioning, P13: Inter-module interaction, and P14: Pre-assigned number of modules. For a full definition of each property, with corresponding examples, refer to Section 3.4.1.

1(c) What are the different purposes behind module creation? There are seven different use-cases for which modules exist (U1-U7). The use-cases are as follows: U1: Maintenance, U2: Automated reasoning, U3: Validation, U4: Processing, U5: Comprehension, U6: Collaborative efforts, and U7: Reuse. For a full definition of each use-case, with corresponding examples, refer to Section 3.2.

1(d)Which techniques have been proposed to perform different types of modularisation?

There are nine techniques that have been proposed to perform modularisation (MT1-MT9). This includes graph theory approaches (MT1-MT2), statistical approaches (MT3), and semantic approaches (MT4-MT9). The techniques are as follows: MT1: Graph partitioning, MT2: Modularity maximisation, MT3: Hierarchical clustering, MT4: Locality modularity, MT5: Query-based modularity, MT6: Semantic-based abstraction, MT7: *A priori* modularity, MT8: Manual modularity, and MT9: Language simplification. For a full definition of each technique, with corresponding examples, refer to Section 3.5.

1(e) How can existing techniques for modularity be improved?

Existing techniques are improved as follows. We performed a literature review of existing techniques for modularity in Section 2.6 where we concluded that the abstraction techniques have no tool support at present, and some of the abstraction techniques are tailored towards conceptual data modules and not ontologies, and there is limited tool support for language simplification techniques. Following this, we performed a classification of ontology modules, and from this, in Section 3.6.2.3 it was

found that tool-based support is lacking for generating various module types. To solve these problems and improve existing techniques we formalised various definitions for abstraction and expressiveness with regard to modularity, designed new algorithms, and implemented the algorithms in the Novel Ontology Modularisation SoftwAre (NOMSA). The five new algorithms we designed are used to generate abstraction and expressiveness type modules: 1) T9: axiom abstraction, 2) T10: entity type abstraction, 3) T11: high-level abstraction, 4) T12: weighted abstraction, and 5) T14: feature expressiveness algorithms. All five of these algorithms are presented in Section 4.4.1.

1(f) What is the criteria for 'good' or 'usable' ontology modules to meet?

The criteria for good modules are dependent on the type of module that it is. This is presented in Section 4.2.7.3. We compiled a list of evaluation metrics by studying existing literature on modularity and creating new metrics. This resulted in 13 metrics from the literature, of which seven were short of a metric for quantitative evaluation that have now been devised (indicated with an asterisk below), and three new ones have been added (indicated with a double asterisk below). The evaluation metrics are grouped into: structural (EM1-EM7), logical (EM8-EM9), relational (EM10-EM12), information hiding (EM13-EM14), and richness (EM15-EM16) criteria. The evaluation metrics are as follows: EM1: Size, EM2: Relative size**, EM3: Appropriateness, EM4: Atomic Size**, EM5: Intra-module distance*, EM6: Relative Intra-module distance**, EM7: Cohesion, EM8: Correctness*, EM9: Completeness*, EM10: Inter-module distance*, EM11: Coupling*, EM12: Redundancy, EM13: Encapsulation*, EM14: Independence*, EM15: Attribute richness, EM16: Inheritance richness. For a full definition of each evaluation metric, with corresponding examples, refer to Section 4.2. The list of evaluation metrics was implemented in a software tool, TOMM. We performed an experiment to evaluate modules with a set of metrics using TOMM to determine which metrics can be used to evaluate which module types and how to tell if a module is of good quality. The experiment revealed what makes a good quality module, which differs for the various module types. This is shown in the dependency diagram in Figure 5.1.

1(g) Is there a way to link the above answers in order to guide the modularity process?

The answers to questions 1(a)-1(e) are linked as follows, based on a classification of ontology modules. Each dimension from the questions (use-case, technique, type, property, and evaluation metric) is linked systematically to create dependencies between them. When a user wishes to create a module, having the use-case on hand, the user is directed towards which type of module is to be created thanks to the dependencies between the use-case and type. From this, the user can now check which is the appropriate technique to use to generate such a module, thanks to the dependencies between type and technique. Next, the technique tells the user which properties the module ought to exhibit, thanks to the dependencies between technique and property. These properties can be used to annotate the module with metadata which promotes the long-term goal of ontology discovery and reuse. A high-level view of this process is shown in the framework for modularity in Figure 5.2.

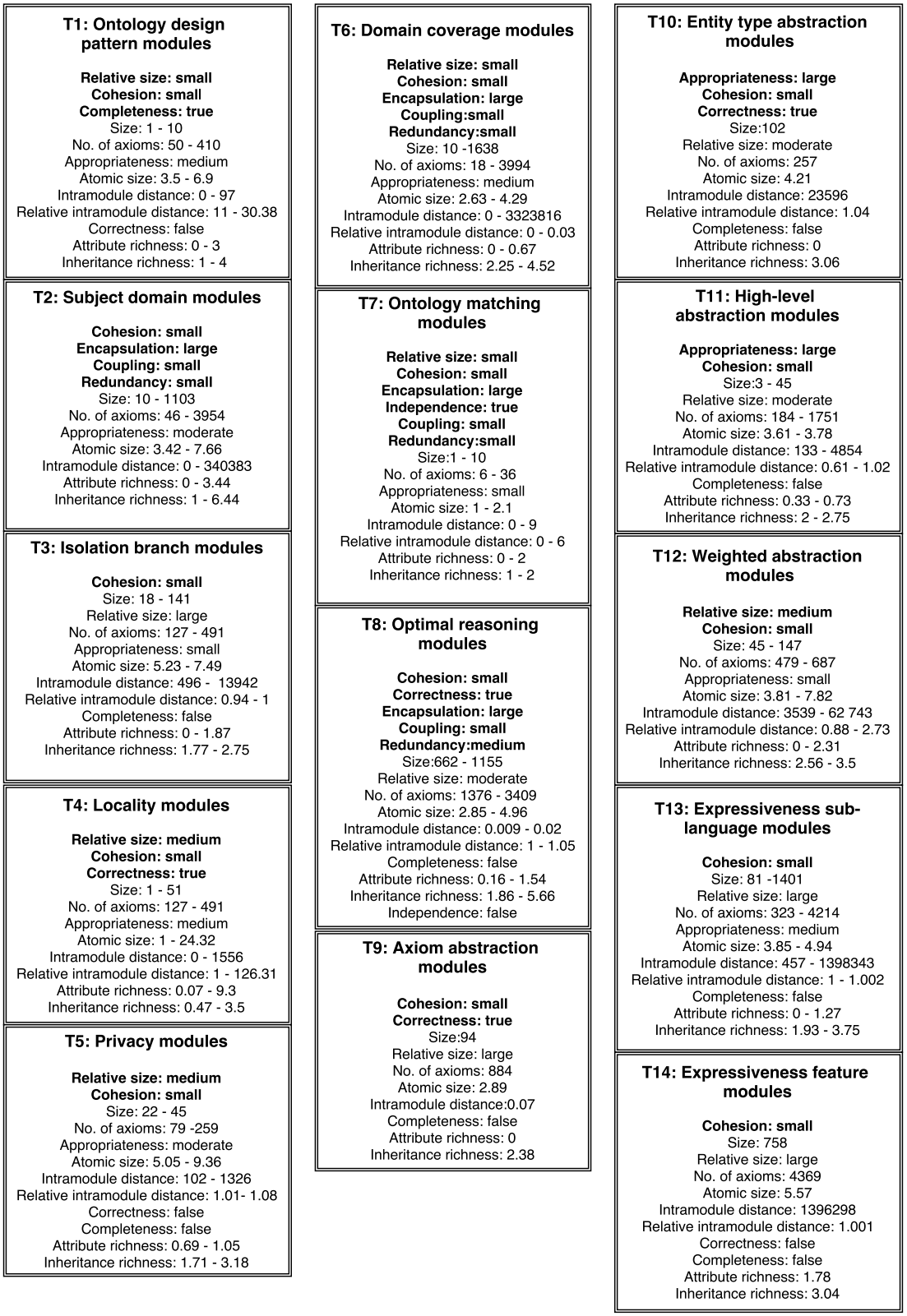


Figure 5.1: The set of metrics that can be measured for each module type. Metrics and values in bold font are those which evaluate well for a module type.

Figure 5.2: A high-level view of the framework for modularity.

Now we can re-visit our main research question.

1. How can one devise a formal foundation for modularity to improve existing modularity techniques and results? The work done in the thesis, on solving the problems concerning modularity has led to a formal foundation for modularity. We created the foundation as follows:

Identify dimensions for modularity: Questions 1 (a)-(d),(f) deal with identifying dimensions for modularity and populating each dimension with a list of sub-dimensions.

Create dependencies between dimensions: An experimental evaluation where a set of modules was classified revealed dependencies between the dimensions. The dependencies make up a formal framework for modularity, and they can be used systematically to guide the modularisation process (question 1(g)), and to annotate modules with information about its properties. The framework was evaluated using both ontology and conceptual data model use-cases in Section 3.8.1.

Determine how to evaluate a module: The dependencies between a module's type and evaluation metrics provide insight on whether a module is good quality (question 1(f)). The dependencies were created thanks to the Tool for Ontology Metrics (TOMM) software we created to generate metrics for ontology modules. TOMM was evaluated using ontology module use-cases to check if the modules were of good quality.

Improve modularisation techniques: Question 1(e) deals with improving modularisation techniques. From the review of existing literature and the experimental evaluation where a set of modules were classified, we gained insight on algorithms that were lacking for modularisation and techniques that were lacking in existing modularisation tools. This drove us to design new algorithms to generate modules and to automate them in the NOvel Modularisation SoftwAre (NOMSA) tool. NOMSA was evaluated by comparing it to other tools, and with an experimental evaluation involving a set of ontologies which were to be modularised and analysed.

The formal foundation for modularisation comprises: an exhaustive set of modularity dimensions that are linked to provide dependencies between them. These dependencies make up the formal framework for modularity. The framework may be used to systematically guide the modularisation process, in cases where users are

unsure about which techniques should be used to generate a module, and how to evaluate a module thanks to the TOMM metrics tool and dependencies. The formal framework also provides ontology developers with properties that a module ought to exhibit to annotate it for promoting ontology reuse. An investigation into module evaluation revealed how to measure the quality of an ontology module using the new and existing metrics that were implemented in the novel tool, TOMM. An investigation, into module interchangeability with the novel tool, SUGOI-Gen revealed that swapping modules in a modular ontology is possible, and has an impact on the module's metrics, and reasoning processing time. We formalised definitions for various abstraction and expressiveness terms and created five new algorithms to fill in the gaps of insufficient modularity tools and techniques. These algorithms were implemented in a software tool, NOMSA, to automate the process. All the algorithms were able to modularise the set of ontologies successfully, and assessing the quality of the modules reveal that two of the algorithms generate modules that are of good quality and for the remaining three algorithms they generate some 'good' quality modules, but it is not possible to meet the expected metric values for all the resulting modules, for some of the metrics depend on the source ontology.

5.2 Future research

Our significant achievement of providing a foundation for modularity encompasses various contributions: a framework for modularity, new algorithms for modularisation, a method and tool for evaluating the quality of a module, and a tool for module management in the form of automatic module swapping, and the foundation successfully solves several problems concerning modularity. A number of topics merit further investigation to expand this work.

Automating manual methods: The classification of existing ontology modules revealed that 9 out of 14 module types were created using manual methods. While we created algorithms and a tool to automate the manual methods for 5 more module types, there are still 4 module types that need to be manually created.

For creating ontology design patterns, isolation branch, privacy, and sub-language expressiveness modules, manual methods need to be used. Since ontology design patterns deal with creating a module by isolating a part of the ontology that can be reused as a best practice for recurring ontology issues, it is unlikely that this can be automated. The remaining three module types, however, could be achieved with by designing new rules and algorithms and implementing them in software.

Linking modules: Modules are not always stand-alone ontologies. In some cases, users wish to link concepts in a module to those of a related module. To date, there are several linking languages such as ε-connections [94], Distributed Description Logics (DDL) [19], Package-based Description Logics (P-DL) [8], C-OWL [21], and the Distributed Ontology Language (DOL) [108]. Each of these

languages have drawbacks and are insufficient for providing complete module links [7, 9]. In DDL, relations are restricted to one-to-one domain relations. In ε-connections, general subsumption is affected whereby a class cannot be declared to be a subclass of a class of a related module; properties are affected in the same way. P-DL uses the OWL import statement which has a major drawback of importing all the classes, axioms and properties of the imported ontology and does not allow partial reuse as needed for ontology modularity linking. C-OWL does not support linking modules with relational properties, and that there are some reasoning difficulties. For the DOL linking language, there has been no formal investigation on using DOL to exhaustively link ontology modules, with different types of axioms and relations yet. It is worthwhile to investigate using DOL as a module linking language for a set of inter-related modules.

Module management: Ontology modules are constantly evolving. Managing these modules is a cumbersome task due to the lack of modular ontology evolution systems and tools. While we did provide some core solution for this with the properties that are used to annotate modules to promote ontology reuse, it is difficult to manage and maintain changes among interrelated modules. Module management requires further investigation to assist with the changes and the interactions among related modules.

Module classification: For this work to have an impact on the current field of ontology modularisation, more classification and testing needs to be performed using real-world modules. To do this, we will look at active ontology repositories such as OntoHub[107] and perform classification and tests for verification and to uncover more information.

Module annotation The properties dimension of the framework reveals information about an ontology module. This could be added as a parameter to existing ontology repositories to annotate modules and enable ontology discovery and reuse.

Ontology methodology integration To promote widespread use of the work, the framework for modularisation could be integrated into the NeOn ontology methodology [150], particularly at the onset of the methodology at the specification step (1). This step could be expanded to mention that use-cases need to be identified and to then traverse through the steps of the modularisation framework.

Appendix A

Classification of the set of modules

For the experimental evaluation in Section 3.6 where we classified a set of 189 ontology modules according to the dimensions for modularity, we present the full classification for the modules according to use-case, type, property, and technique in Table A.1.

Table A.1: The classification of the set of modules for the use-case, type, property, and technique dimensions; graph p = graph partitioning. Modules without sources are ones that were generated for the classification due to the lack of certain module types.

	Module	**Original ontology**	**Use-case**	**Type**	**Properties**	**Technique**	**Source**
1.	Set	Collections	U7	M1	P2, P6, P7, P8	MT8: Manual	[18]
2.	Typesofentities	DUL	U7	M1	P2, P6, P7, P8	MT8: Manual	[127]
3.	ActingFor	DUL	U7	M1	P2, P6, P7, P8	MT8: Manual	[45]
4.	Situation	DUL	U7	M1	P2, P6, P7, P8	MT8: Manual	[46]
5.	BathroomLocation	-	U7	M1	P8	MT8: Manual	[1]
6.	Calendar	-	U7	M1	P8	MT8: Manual	

7.	CalendarFreeTime	-	U7	M1	P8	MT8: Manual	
8.	Content	-	U7	M1	P8	MT8: Manual	
9.	ContentCollection	-	U7	M1	P8	MT8: Manual	
10.	ContentList	-	U7	M1	P8	MT8: Manual	
11.	ContentRealizations	-	U7	M1	P8	MT8: Manual	
12.	Context	-	U7	M1	P8	MT8: Manual	
13.	ContextSequence	-	U7	M1	P8	MT8: Manual	
Set of OntoSpace modules					P9, P14		
14.	Amirequirements	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	[11]
15.	BuildingArchitecture	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
16.	BuildingConstruction	-	U1, U3, U6, U7	M2	P5	MT7: *A priori*	
17.	DOLCE-Lite	-	U1, U3, U6, U7	M2	P5	MT7: *A priori*	
18.	DomOnto	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
19.	RCC-Ontology	-	U1, U3, U6, U7	M2	P5	MT7: *A priori*	
20.	SpatialOntology	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
Set of Onto-DM modules					P9, P14		
21.	OntoDM-core	-	U1,U3, U6, U7	M2	P8	MT7: *A priori*	[121]
22.	OntoDM-KDD	-	U1,U3, U6, U7	M2	P5	MT7: *A priori*	
23.	OntoDT	-	U1,U3, U6, U7	M2	P8	MT7: *A priori*	
Set of myExperiment modules					P9, P14		
24.	SNARM	-	U1,U3, U6, U7	M2	P5	MT7: *A priori*	[111]
25.	Base	-	U1,U3, U6, U7	M2	P5	MT7: *A priori*	
26.	Attribution&Credit	-	U1,U3, U6, U7	M2	P5	MT7: *A priori*	
27.	Annotations	-	U1,U3, U6, U7	M2	P5	MT7: *A priori*	
28.	Packs	-	U1,U3, U6, U7	M2	P5	MT7: *A priori*	
29.	Experiments	-	U1,U3, U6, U7	M2	P5	MT7: *A priori*	
30.	Viewings&Downloads	-	U1,U3, U6, U7	M2	P5	MT7: *A priori*	
31.	Contributions	-	U1,U3, U6, U7	M2	P5	MT7: *A priori*	
32.	Components	-	U1,U3, U6, U7	M2	P5	MT7: *A priori*	

33.	Specific	-	U1,U3, U6, U7	M2	P5	MT7: *A priori*	
Set of GIST modules					P9, P14		
34.	gistAddress	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	[103]
35.	gistAgreement	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
36.	gistCategory	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
37.	gistContent	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
38.	gistCore	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
39.	gistEvent	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
40.	gistID	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
41.	gistIntention	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
42.	gistMagnitude	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
43.	gistMeasure	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
44.	gistOrganization	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
45.	gistPerson	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
46.	gistPhysicalThing	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
47.	gistPlace	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
48.	gistTemporalRelation	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
49.	gistTime	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
50.	gistTop	-	U1, U3, U6, U7	M2	P5	MT7: *A priori*	
51.	gistUnit	-	U1, U3, U6, U7	M2	P8	MT7: *A priori*	
Set of NCS modules					P9, P14		
52.	ncsLN	-	U1, U3, U6, U7	M2	P5	MT7: *A priori*	[25]
53.	ncsNY	-	U1, U3, U6, U7	M2	P5	MT7: *A priori*	
54.	ncsSW	-	U1, U3, U6, U7	M2	P5	MT7: *A priori*	
55.	ncsXH	-	U1, U3, U6, U7	M2	P5	MT7: *A priori*	
56.	GFO No Occurrents	GFO	U7	M3	P2, P5, P6, P7	MT8: Manual	[88]
57.	GFO No Persistants and Presentials	GFO	U7	M3	P2, P5, P6, P7	MT8: Manual	
58.	BFO Continuants	BFO	U7	M3	P2, P5, P6, P7	MT8: Manual	

59.	BFO Occurrents	BFO	U7	M3	P2, P5, P6, P7	MT8: Manual	
60.	DOLCE-Endurants	DOLCE-Lite	U7	M3	P2, P5, P6, P7	MT8: Manual	
61.	DOLCE-Perdurants	DOLCE-Lite	U7	M3	P2, P5, P6, P7	MT8: Manual	
62.	DOLCE-No-Quality-Qualia	DOLCE-Lite	U7	M3	P2, P5, P6, P7	MT8: Manual	
63.	LargeAliphatic AminoAcid	Amino acid-inferred	U7	M4	P1, P2, P5, P6, P7	MT4: Locality	
64.	chemical entity	SIO	U7	M4	P1, P2, P5, P6, P7	MT4: Locality	
65.	Seizure_Types	Epilepsy Ontology	U7	M4	P1, P2, P5, P6, P7	MT4: Locality	
66.	MeatTopping	pizza	U6, U7	M5	P2, P5, P6, P7	MT8: Manual	
67.	VegetableTopping	pizza	U6, U7	M5	P2, P5, P6, P7	MT8: Manual	
Set of Amino acid modules					P9, P12, P13		
68.	Aminoacid_ partition1	amino acid - inferred	U1, U3, U6	M6	P2, P4, P6	MT1: Graph p	
69.	Aminoacid_ partition2	amino acid-inferred	U1, U3, U6	M6	P2, P4, P6	MT1: Graph p	
Set of EDAM modules					P9, P12, P13		
70.	edam_partition1	EDAM_1.9	U1, U3, U6	M6	P2, P4, P6	MT1: Graph p	
71.	edam_partition2	EDAM_1.9	U1, U3, U6	M6	P2, P4, P5, P6	MT1: Graph p	
72.	edam_partition3	EDAM_1.9	U1, U3, U6	M6	P2, P4, P6	MT1: Graph p	
73.	edam_partition4	EDAM_1.9	U1, U3, U6	M6	P2, P4, P6	MT1: Graph p	
74.	edam_partition5	EDAM_1.9	U1, U3, U6	M6	P2, P4, P6	MT1: Graph p	
Set of MEO modules					P12, P14		
75.	meo_partition1	meo_v07	U1, U3, U6	M6	P2, P5, P6, P7	MT1: Graph p	
76.	meo_partition2	meo_v07	U1, U3, U6	M6	P2, P5, P6, P7	MT1: Graph p	
77.	meo_partition3	meo_v07	U1, U3, U6	M6	P2, P5, P6, P7	MT1: Graph p	
78.	meo_partition4	meo_v07	U1, U3, U6	M6	P2, P5, P6, P7	MT1: Graph p	
79.	meo_partition5	meo_v07	U1, U3, U6	M6	P2, P5, P6, P7	MT1: Graph p	

Set of CARO modules					P10, P12		
80.	single_134	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
81	single_138	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
82.	single_141	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
83.	single_146	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
84.	single_151	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
85.	single_158	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
86.	single_161	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
87	single_169	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
88.	single_170	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
89.	Source_Block_145	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
90.	Source_Block_147	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
91.	Source_Block_149	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
92.	Source_Block_153	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
93.	Source_Block_154	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
94.	Source_Block_156	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
95.	Source_Block_157	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
96.	Source_Block_159	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
97.	Source_Block_160	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
98.	Source_Block_162	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
99.	Source_Block_165	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
100.	Source_Block_166	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
101.	Source_Block_168	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
102.	Source_Block_172	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
103.	Source_Block_173	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
104.	Source_Block_176	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
105.	Source_Block_177	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
106.	Source_Block_179	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	

107.	Source_Block_180	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
108.	Source_Block_181	CARO	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
Set of Spatial modules					P12, P14		
109.	Target_Block_4	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
110.	Target_Block_9	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
111.	Target_Block_13	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
112.	Target_Block_24	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
113.	Target_Block_27	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
114.	Target_Block_46	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
115.	Target_Block_48	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
116.	Target_Block_49	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
117.	Target_Block_50	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
118.	Target_Block_51	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
119.	Target_Block_52	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
120.	Target_Block_55	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
121.	Target_Block_56	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
122.	Target_Block_57	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
123.	Target_Block_58	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
124.	Target_Block_59	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
125.	Target_Block_60	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
126.	Target_Block_62	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
127.	Target_Block_64	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
128.	Target_Block_66	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
129.	Target_Block_70	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
130.	Target_Block_73	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
131.	Target_Block_75	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
132.	Target_Block_77	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
133.	Target_Block_78	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	

134.	Target_Block_79	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
135.	Target_Block_80	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
136.	Target_Block_81	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
137.	Target_Block_83	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
138.	Target_Block_84	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
139.	Target_Block_85	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
140.	Target_Block_90	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
141.	Target_Block_92	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
142.	Target_Block_95	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
143.	Target_Block_96	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
144.	Target_Block_97	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
145.	Target_Block_99	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
146.	Target_Block_101	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
147.	Target_Block_102	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
148.	Target_Block_104	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
149.	Target_Block_106	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
150.	Target_Block_107	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
151.	Target_Block_109	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
152.	Target_Block_110	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
153.	Target_Block_112	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
154.	Target_Block_114	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
155.	Target_Block_115	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
156.	Target_Block_116	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
157.	Target_Block_117	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
158.	Target_Block_119	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
159.	Target_Block_120	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
160.	Target_Block_122	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
161.	Target_Block_123	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	

162.	Target_Block_124	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
163.	Target_Block_125	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
164.	Target_Block_126	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
165.	Target_Block_127	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
166.	Target_Block_129	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
167.	Target_Block_130	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
168.	Target_Block_131	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
169.	Target_Block_132	Spatial	U4, U6	M7	P2, P5, P6, P7	MT1: Graph p	
170.	DMOP-branch-Endurant	DMOP	U2, U4	M8	P1, P2, P6, P7	MT4: Locality	[78]
171.	DMOP-branch-Perdurant	DMOP	U2, U4	M8	P1, P2, P6, P7	MT4: Locality	
172.	DMOP-branch-Abstract-Quality	DMOP	U2, U4	M8	P1, P2, P6, P7	MT4: Locality	
173.	DMOP-branch-Toplevel	DMOP	U2, U4	M8	P1, P2, P6, P7	MT4: Locality	
174.	FGA_taxonomy	FGA	U5	M9	P3, P3.1, P5, P6, P7	MT8: Manual	
175.	bco_classes_only	bco	U5	M10	P3, P3.2, P4, P5	MT8: Manual	
176.	GFO ATO	GFO	U5	M11	P3, P3.2, P5, P6, P7	MT8: Manual	[88]
177.	GFO ACO	GFO	U5	M11	P3, P3.2, P5, P6, P7	MT8: Manual	
178.	DMOP-branch-Toplevel	DMOP	U5	M11	P3, P3.2, P6, P7	MT8: Manual	[78]
179.	Biotoplite	Biotop	U5	M11	P3, P3.2, P4, P5	MT8: Manual	[13]
180.	GFO-Basic	GFO	U5	M12	P3, P7, P8	MT8: Manual	[88]
181.	FMA_subset	FMA	U5	M12	P3, P3.2, P4, P5	MT8: Manual	[133]
182.	FamilyHealthHistory_familyrelations	FamilyHealth History	U5	M12	P1, P3, P3.2, P4, P5	MT8: Manual	
183.	DMOP-profile-EL	DMOP	U2	M13	P2, P6, P7	MT8: Manual	[78]

184.	Fire-EL	Fire0.9.1	U2	M13	P2, P5, P6, P7	MT8: Manual	
185.	Fire-RL	Fire0.9.1	U2	M13	P2, P5, P6, P7	MT8: Manual	
186.	Typon-EL	Typon	U2	M13	P2, P5, P6, P7	MT8: Manual	
187.	Typon-QL	Typon	U2	M13	P2, P6, P7	MT8: Manual	
188.	Typo-RL	Typon	U2	M13	P2, P6, P7	MT8: Manual	
189.	DMOP-WithoutInverseRoles	DMOP	U2	M14	P2, P4, P6	MT8: Manual	[78]

Appendix B

The Burger Ontology

We provide the complete Burger OWL ontology written in OWL Functional Syntax here.

```
Prefix(:=<http://www.thezfiles.co.za/burger#>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Prefix(xml:=<http://www.w3.org/XML/1998/namespace>)
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(rdfs:=<http://www.w3.org/2000/01/rdf-schema#>)
Prefix(untitled-ontology-415:=<http://www.semanticweb.org/zkhan/
ontologies/2016/8/untitled-ontology-415#>)

Ontology(<http://www.thezfiles.co.za/burger>

Declaration(Class(:BeefPatty))
Declaration(Class(:Beefburger))
Declaration(Class(:Burger))
Declaration(Class(:BurgerBun))
Declaration(Class(:Cheapburger))
Declaration(Class(:Cheese))
Declaration(Class(:Chef))
Declaration(Class(:Customer))
Declaration(Class(:Filling))
Declaration(Class(:HamBurger))
Declaration(Class(:HealthyBurger))
Declaration(Class(:Lettuce))
Declaration(Class(:Medium))
Declaration(Class(:Patty))
Declaration(Class(:PattyCook))
Declaration(Class(:Person))
Declaration(Class(:Rare))
Declaration(Class(:Sauce))
Declaration(Class(:Tomato))
Declaration(Class(:WellDone))
Declaration(Class(:WhiteBun))
Declaration(Class(:WholeWheatBun))
Declaration(ObjectProperty(:cookedBy))
Declaration(ObjectProperty(:hasBun))
```

```
Declaration(ObjectProperty(:hasFilling))
Declaration(ObjectProperty(:hasPatty))
Declaration(ObjectProperty(:hasPattyCook))
Declaration(NamedIndividual(:ChefRose))
Declaration(NamedIndividual(:MarthasBurger))
Declaration(NamedIndividual(:MyBurger))

#############################
#   Object Properties
#############################

# Object Property: :hasBun (:hasBun)

FunctionalObjectProperty(:hasBun)
ObjectPropertyDomain(:hasBun :Burger)
ObjectPropertyRange(:hasBun :BurgerBun)

# Object Property: :hasPatty (:hasPatty)

ObjectPropertyDomain(:hasPatty :Burger)
ObjectPropertyRange(:hasPatty :Patty)

# Object Property: :hasPattyCook (:hasPattyCook)

ObjectPropertyDomain(:hasPattyCook :Patty)
ObjectPropertyRange(:hasPattyCook :PattyCook)

#############################
#   Classes
#############################

# Class: :BeefPatty (:BeefPatty)

SubClassOf(:BeefPatty :Patty)

# Class: :Beefburger (:Beefburger)

EquivalentClasses(:Beefburger :HamBurger)
SubClassOf(:Beefburger :Burger)

# Class: :Cheapburger (:Cheapburger)

SubClassOf(:Cheapburger :Burger)
SubClassOf(:Cheapburger ObjectMaxCardinality(1 :hasFilling :Filling))

# Class: :Cheese (:Cheese)

SubClassOf(:Cheese :Filling)

# Class: :Chef (:Chef)

SubClassOf(:Chef :Person)

# Class: :Customer (:Customer)
```

```
SubClassOf(:Customer :Person)

# Class: :HamBurger (:HamBurger)

EquivalentClasses(:HamBurger ObjectIntersectionOf(:Burger
ObjectSomeValuesFrom(:hasPatty :BeefPatty)))
SubClassOf(:HamBurger :Burger)

# Class: :HealthyBurger (:HealthyBurger)

SubClassOf(:HealthyBurger :Burger)
SubClassOf(:HealthyBurger ObjectAllValuesFrom(:hasFilling ObjectUnionOf
(:Lettuce :Tomato)))

# Class: :Lettuce (:Lettuce)

SubClassOf(:Lettuce :Filling)

# Class: :Medium (:Medium)

SubClassOf(:Medium :PattyCook)

# Class: :PattyCook (:PattyCook)

EquivalentClasses(:PattyCook ObjectUnionOf(:Medium :Rare :WellDone))

# Class: :Rare (:Rare)

SubClassOf(:Rare :PattyCook)

# Class: :Sauce (:Sauce)

SubClassOf(:Sauce :Filling)

# Class: :Tomato (:Tomato)

SubClassOf(:Tomato :Filling)

# Class: :WellDone (:WellDone)

SubClassOf(:WellDone :PattyCook)

# Class: :WhiteBun (:WhiteBun)

SubClassOf(:WhiteBun :BurgerBun)
DisjointClasses(:WhiteBun :WholeWheatBun)

# Class: :WholeWheatBun (:WholeWheatBun)

SubClassOf(:WholeWheatBun :BurgerBun)

#############################
#   Named Individuals
```

```
############################

# Individual: :MarthasBurger (:MarthasBurger)

# Individual: :MyBurger (:MyBurger)

ClassAssertion(:Beefburger :MyBurger)
ObjectPropertyAssertion(:cookedBy :MyBurger :ChefRose)
DifferentIndividuals(:MarthasBurger :MyBurger))
```

Bibliography

[1] Set of IKS content ontology design patterns. `http://www.ontologydesignpatterns.org/iks/ami/2011/02/` last accessed: 20 June 2017, 2011. Online (January 2011).

[2] Sarra Ben Abbès, Andreas Scheuermann, Thomas Meilender, and Mathieu d'Aquin. Characterizing Modular Ontologies. In *The 6th International Workshop on Modular Ontologies (WoMO '12)*, volume 875 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

[3] Soraya Setti Ahmed, Mimoun Malki, and Sidi Mohamed Benslimane. Ontology partitioning: Clustering based approach. *International Journal of Information Technology and Computer Science*, 7(6):1–11, 2015.

[4] F. Amato, A. De Santo, V. Moscato, F. Persia, A. Picariello, and S.R. Poccia. Partitioning of ontologies driven by a structure-based approach. In *Ninth International Conference on Semantic Computing (ICSC'15)*, pages 320–323. IEEE, 2015. Anaheim, California, USA, February 7-9 2015.

[5] Erick Antezana, Mikel Egaña, Ward Blondè, Aitzol Illarramendi, Iñaki Bilbao, Bernard De Baets, Robert Stevens, Vladimir Mironov, and Martin Kuiper. The cell cycle ontology: an application ontology for the representation and integrated analysis of the cell cycle process. *Genome Biology*, 10(5):R58, 2009.

[6] Kenneth Baclawski, Christopher J. Matheus, Mieczyslaw M. Kokar, Jerzy Letkowski, and Paul A. Kogut. Towards a symptom ontology for semantic web applications. In *The Third International Semantic Web Conference, (ISWC'14)*, volume 3298 of *LNCS*, pages 650–667. Springer, 2004. Hiroshima, Japan, November 7-11.

[7] Jie Bao, Doina Caragea, and Vasant Honavar. On the Semantics of Linking and Importing in Modular Ontologies. In *5th International Semantic Web Conference (ISWC'06)*, volume 4273 of *Lecture Notes in Computer Science*, pages 72–86. Springer, 2006. Nov 5-9, Athens. GA, USA.

[8] Jie Bao, Doina Caragea, and Vasant Honavar. Towards Collaborative Environments for Ontology Construction and Sharing. In *International Symposium on Collaborative Technologies and Systems (CTS '06)*, pages 99–108. IEEE Computer Society, 2006. May 14-17, Las Vegas, NV, USA.

[9] Jie Bao and Vasant Honavar. Adapt OWL as a Modular Ontology Language. In Bernardo Cuenca Grau, Pascal Hitzler, Conor Shankey, and Evan Wallace, editors, *The OWLED Workshop on OWL: Experiences and Directions (OWLED '06)*, volume 216 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006. November 10-11, Athens, Georgia, USA.

[10] Vladimir Batagelj. Analysis of large networks - islands. Dagstuhl seminar 03361: Algorithmic Aspects of Large and Complex Networks, 2003. Dagstuhl, August 31 - September 5.

[11] John A. Bateman, Kerstin Fischer, Reinhard Moratz, Scott Farrar, and Thora Tenbrink. Project I1-OntoSpace: Ontologies for Spatial Communication. In *DiaBruck, 7th Workshop on the Semantics and Pragmatics of Dialogue, Proceedings*, pages 163–164, 2003. 4th-6th September,Wallerfangen, Germany.

[12] Johannes Bauer, Ulrike Sattler, and Bijan Parsia. Explaining by example: Model exploration for ontology comprehension. In *The 22nd International Workshop on Description Logics (DL'09)*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009. Oxford, UK, July 27-30, 2009.

[13] Elena Beisswanger, Stefan Schulz, Holger Stenzhorn, and Udo Hahn. BioTop: An upper domain ontology for the life sciences – a description of its current structure, contents and interfaces to OBO ontologies. *Applied Ontology*, 3(4):205–212, 2008.

[14] Elena Beisswanger, Stefan Schulz, Holger Stenzhorn, and Udo Hahn. Biotop: An upper domain ontology for the life sciencesa description of its current structure, contents and interfaces to OBO ontologies. *Applied Ontology*, 3(4):205–212, 2008.

[15] Kele T. Belloze, Daniel Igor S. B. Monteiro, Tulio F. Lima, Floriano P. Silva Jr., and Maria Cláudia Reis Cavalcanti. An evaluation of annotation tools for biomedical texts. In *Proceedings of Joint V Seminar on Ontology Research in Brazil and VII International Workshop on Metamodels, Ontologies and Semantic Technologies*, volume 938 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012. Recife, Brazil, September 19-21.

[16] Johann Bergh, Aurona Gerber, Tommie Meyer, and Lynette van Zijl. Path analysis for ontology comprehension. In *The Seventh Australasian Ontology Workshop (AOW'11)*, CRPIT. ACS, 2011. 5 December, Perth Australia.

[17] Norman Biggs, E Keith Lloyd, and Robin J Wilson. *Graph Theory, 1736-1936*. Clarendon Press Oxford, 1976.

[18] Eva Blomqvist. Set ontology design pattern. `http://ontologydesignpatterns.org/wiki/Submissions:Set` last accessed: 20 June 2017, 2010. Online (December 2010).

[19] Alex Borgida and Luciano Serafini. Distributed Description Logics: Directed Domain Correspondences in Federated Information Sources. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, volume 2519 of *Lecture Notes in Computer Science*, pages 36–53. Springer Berlin Heidelberg, 2002.

[20] Stefano Borgo. Goals of modularity: A voice from the foundational viewpoint. In Oliver Kutz and Thomas Schneider, editors, *The Fifth International Workshop on Modular Ontologies (WOMO'2011)*, volume 230 of *Frontiers in Artificial Intelligence and Applications*, pages 1–6. IOS Press, 2011. Ljubljana, Slovenia, August.

[21] Paolo Bouquet, Fausto Giunchiglia, Frank Harmelen, Luciano Serafini, and Heiner Stuckenschmidt. C-OWL: Contextualizing Ontologies. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *The Second International Semantic Web Conference (ISWC '03)*, volume 2870 of *Lecture Notes in Computer Science*, pages 164–179. Springer, 2003. October 20-23, Sanibel Island, FL, USA.

[22] L. J. Campbell, Terry A. Halpin, and Henderik Alex Proper. Conceptual schemas with abstractions: Making flat conceptual schemas more comprehensible. *Data Knowledge Engineering*, 20(1):39–85, 1996.

[23] Werner Ceusters, Barry Smith, and Christoffel DHAEN' Anand KUMAR. Mistakes in medical ontologies: Where do they come from and how can they be. In *Ontologies in Medicine. Proceedings of the Workshop on Medical Ontologies*, volume 102, pages 145–164. IOS Press, 2003. Ronem October 2003.

[24] C. Chavula and C. M. Keet. An orchestration framework for linguistic task ontologies. In *Proceedings of the 9th Metadata and Semantics Research Conference (MTSR'15)*, CCIS, page in print. Springer, 2015. 9-11 Sept., 2015, Manchester, UK.

[25] Catherine Chavula and C. Maria Keet. An orchestration framework for linguistic task ontologies. In Emmanouel Garoufallou, Richard J. Hartley, and Panorea Gaitanou, editors, *9th Metadata and Semantics Research Conference (MTSR'15)*, volume 544 of *CCIS*, pages 3–14. Springer, 2015. 9-11 September, 2015, Manchester, UK.

[26] Jiyang Chen, Osmar R. Zaïane, and Randy Goebel. Detecting communities in social networks using max-min modularity. In *Proceedings of the SIAM International Conference on Data Mining (SDM'09)*, pages 978–989. SIAM, 2009. April 30 - May 2, Sparks, Nevada, USA.

[27] Paolo Ciccarese and Silvio Peroni. The collections ontology: Creating and handling collections in OWL 2 DL frameworks. *Semantic Web*, 5(6):515–529, 2014.

[28] Lindsay Grey Cowell and Barry Smith. *Infectious Disease Ontology*, pages 373–395. Springer New York, 2010.

[29] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. A logical framework for modularity of ontologies. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 298–303, 2007. Hyderabad, India, January 6-12, 2007.

[30] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Modular Reuse of Ontologies: Theory and Practice. *Journal of Artificial Intelligence Research (JAIR)*, 31:273–318, 2008.

[31] Bernardo Cuenca Grau, Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Modularity and Web Ontologies. In *10th International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 198–209. AAAI Press, 2006. June 2-5, Lake District, United Kingdom.

[32] Wasila M. Dahdul, Hong Cui, Paula M. Mabee, Christopher J. Mungall, David Osumi-Sutherland, Ramona Walls, and Melissa Haendel. Nose to tail, roots to shoots: spatial descriptors for phenotypic diversity in the biological spatial ontology. *J. Biomedical Semantics*, 5:34, 2014.

[33] Mathieu d'Aquin, Marta Sabou, and Enrico Motta. Modularization: a Key for the Dynamic Selection of Relevant Knowledge Components. In *1st International Workshop on Modular Ontologies (WoMO'06)*, volume 232 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006. Nov 5, Athens, Georgia, USA.

[34] Mathieu d'Aquin, Anne Schlicht, Heiner Stuckenschmidt, and Marta Sabou. Ontology modularization for knowledge selection: Experiments and evaluations. In Roland Wagner, Norman Revell, and Günther Pernul, editors, *The 18th International Conference on Database and Expert Systems Applications (DEXA'07)*, volume 4653 of *Lecture Notes in Computer Science*, pages 874–883. Springer, 2007. Regensburg, Germany, September 3-7, 2007.

[35] Mathieu d'Aquin, Anne Schlicht, Heiner Stuckenschmidt, and Marta Sabou. Criteria and evaluation for ontology modularization techniques. In *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, volume 5445 of *Lecture Notes in Computer Science*, pages 67–89. Springer, 2009.

[36] Anusuriya Devaraju, Werner Kuhn, and Chris S. Renschler. A formal model to infer geographic events from sensor observations. *International Journal of Geographical Information Science*, 29(1):1–27, 2015.

[37] Paul Doran, Valentina A. M. Tamma, and Luigi Iannone. Ontology module extraction for ontology reuse: an ontology engineering perspective. In Mário J. Silva, Alberto H. F. Laender, Ricardo A. Baeza-Yates, Deborah L. McGuinness, Bjørn Olstad, Øystein Haug Olsen, and André O. Falcão, editors, *Proceedings*

of the Sixteenth ACM Conference on Information and Knowledge Management (CIKM '07), pages 61–70. ACM, 2007. Lisbon, Portugal, November 6-10.

[38] Zlatan Dragisic, Valentina Ivanova, Patrick Lambrix, Daniel Faria, Ernesto Jiménez-Ruiz, and Catia Pesquita. User validation in ontology alignment. In *15th International Semantic Web Conference (ISWC'16)*, volume 9981 of *Lecture Notes in Computer Science*, pages 200–217, 2016. Kobe, Japan, October 17-21, 2016.

[39] Faezeh Ensan and Weichang Du. A semantic metrics suite for evaluating modular ontologies. *Information Systems*, 38(5):745–770, 2013.

[40] Pablo R. Fillottrani, Enrico Franconi, and Sergio Tessaris. The ICOM 3.0 intelligent conceptual modelling tool and methodology. *Semantic Web*, 3(3):293–306, 2012.

[41] Pablo R. Fillottrani and C. Maria Keet. Patterns for heterogeneous tbox mappings to bridge different modelling decisions. In E. Blomqvist et al., editors, *Proc. of ESWC'17*, volume 10249 of *LNCS*, pages 371–386. Springer, 2017. 30 May - 1 June 2017, Portoroz, Slovenia.

[42] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3):75–174, 2010.

[43] Linton C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215–239, 1978.

[44] A. Gangemi and V. Presutti. Ontology design patterns. In S. Staab and R. Studer, editors, *Handbook on Ontologies*, pages 221–243. Springer Verlag, 2009.

[45] Aldo Gangemi. Acting for ontology design pattern. `http://ontologydesignpatterns.org/wiki/Submissions:ActingFor` last accessed: 20 June 2017, 2010. Online (August 2010).

[46] Aldo Gangemi. Situation ontology design pattern. `http://ontologydesignpatterns.org/wiki/Submissions:Situation` last accessed: 20 June 2017, 2010. Online (March 2010).

[47] Ana Carolina Garcia, Letícia Tiveron, Cláudia Justel, and Maria Cláudia Cavalcanti. Applying graph partitioning techniques to modularize large ontologies. In *Joint V Seminar on Ontology Research in Brazil and VII International Workshop on Metamodels, Ontologies and Semantic Technologies*, volume 938 of *CEUR Workshop Proceedings*, pages 72–83. CEUR-WS.org, 2012. Recife, Brazil, September 19-21.

[48] Juan García, Francisco José García Peñalvo, and Roberto Therón. A survey on ontology metrics. In Miltiadis D. Lytras, Patricia Ordóñez de Pablos, Adrian

Ziderman, Alan Roulstone, Hermann A. Maurer, and Jonathan B. Imber, editors, *Third World Summit on the Knowledge Society, (WSKS'10)*, volume 111 of *Communications in Computer and Information Science*, pages 22–27. Springer, 2010. Corfu, Greece, September 22-24.

[49] Tom Gardiner, Dmitry Tsarkov, and Ian Horrocks. Framework for an automated comparison of description logic reasoners. In *The 5th International Semantic Web Conference (ISWC'06)*, volume 4273 of *Lecture Notes in Computer Science*, pages 654–667. Springer, 2006. Athens, GA, USA, November 5-9, 2006.

[50] Chiara Ghidini and Fausto Giunchiglia. A semantics for abstraction. In Ramon López de Mántaras and Lorenza Saitta, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI'04)*, pages 343–347. IOS Press, 2004. PAIS 2004, Valencia, Spain, August 22-27.

[51] Chiara Ghidini and Fausto Giunchiglia. A semantics for abstraction. In *Proceedings of the 16th European conference on Artificial Intelligence (ECAI-04)*, 2004. Valencia, 22-27 August 2004.

[52] Andrew Gibson, Katy Wolstencroft, and Robert Stevens. Promotion of ontological comprehension: Exposing terms and metadata with web 2.0. In *The Workshop on Social and Collaborative Construction of Structured Knowledge (CKC'07)*, volume 273 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007. Banff, Canada, May 8.

[53] Fausto Giunchiglia, Adolfo Villafiorita, and Toby Walsh. Theories of abstraction. *AI Communication*, 10(3,4):167–176, 1997.

[54] Jennifer Golbeck, Gilberto Fragoso, Frank W. Hartel, James A. Hendler, Jim Oberthaler, and Bijan Parsia. The national cancer institute's thésaurus and ontology. *Journal of Semantic Web*, 1(1):75–80, 2003.

[55] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, June 1993.

[56] N. Guarino. Formal ontology and information systems. In N. Guarino, editor, *Proceedings of the 1st International Conference on Formal Ontology in Information Systems (FOIS'98)*, Frontiers in Artificial intelligence and Applications, pages 3–15, Amsterdam, The Netherlands, 1998. IOS Press.

[57] Melissa A Haendel, Fabian Neuhaus, David Osumi-Sutherland, Paula M Mabee, Jos LV Mejino Jr, Chris J Mungall, and Barry Smith. CARO- The common anatomy reference ontology. In *Anatomy Ontologies for Bioinformatics*, volume 6 of *Computational Biology*, pages 327–349. Springer, 2008.

[58] Fayçal Hamdi, Brigitte Safar, Chantal Reynaud, and Haïfa Zargayouna. Alignment-based partitioning of large-scale ontologies. In *Advances in Knowledge Discovery and Management [Best of EGC 2009, Strasbourg, France]*, vol-

ume 292 of *Studies in Computational Intelligence*, pages 251–269. Springer, 2009.

[59] Jens Hartmann, York Sure, Peter Haase, Raul Palma, and Mari del Carmen Suárez-Figueroa. OMV - Ontology Metadata Vocabulary. In *Ontology Patterns for the Semantic Web (OPSW)*, 2005. Galway, Ireland, November.

[60] Janna Hastings, Paula de Matos, Adriano Dekker, Marcus Ennis, Venkatesh Muthukrishnan, Steve Turner, Gareth Owen, and Christoph Steinbeck. Modular extensions to the ChEBI ontology. In *Proceedings of the 3rd International Conference on Biomedical Ontology (ICBO 2012)*, volume 897 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012. Graz, Austria, July 21-25.

[61] Heinrich Herre. General Formal Ontology (GFO): A foundational ontology for conceptual modelling. In *Theory and applications of ontology: computer applications*, pages 297–345. Springer, 2010.

[62] Heinrich Herre. General Formal Ontology (GFO): A foundational ontology for conceptual modelling. In *Theory and Applications of Ontology: Computer Applications*, chapter 14, pages 297–345. Springer, Heidelberg, 2010.

[63] Ralph Hodgson and Paul J Keller. QUDT-quantities, units, dimensions and data types in OWL and XML. `http://www.qudt.org` last accessed: 20 June 2017, 2011. Online (September 2011).

[64] Robert Hoehndorf, Michel Dumontier, John H Gennari, Sarala Wimalaratne, Bernard de Bono, Daniel L Cook, and Georgios V Gkoutos. Integrating systems biology models and biomedical ontologies. *BMC systems biology*, 5(1):124, 2011.

[65] Robert Hoehndorf, Frank Loebe, Roberto Poli, Heinrich Herre, and Janet Kelso. GFO-Bio: A biological core ontology. *Applied Ontology*, 3(4):219–227, 2008.

[66] Robert Hoehndorf, Frank Loebe, Roberto Poli, Heinrich Herre, and Janet Kelso. GFO-Bio: A biological core ontology. *Applied Ontology*, 3(4):219–227, 2008.

[67] Rinke Hoekstra, Joost Breuker, Marcello Di Bello, and Alexander Boer. The LKIF core ontology of basic legal concepts. In *Proceedings of the 2nd Workshop on Legal Ontologies and Artificial Intelligence Techniques (LOAIT '07)*, volume 321 of *CEUR Workshop Proceedings*, pages 43–63. CEUR-WS.org, 2007. June 4th, Stanford University, Stanford, CA, USA.

[68] Joana Hois, Mehul Bhatt, and Oliver Kutz. Modular ontologies for architectural design. In *Proceedings of the International workshop on Formal Ontologies Meet Industry (FOMI'09)*, 2009. September 2, 2009, Vicenza, Italy.

[69] Matthew Horridge and Sean Bechhofer. The OWL API: A java API for OWL ontologies. *Semantic Web*, 2(1):11–21, 2011.

[70] Jon C. Ison, Matús Kalas, Inge Jonassen, Dan M. Bolser, Mahmut Uludag, Hamish McWilliam, James Malone, Rodrigo Lopez, Steve Pettifer, and Peter M. Rice. EDAM: an ontology of bioinformatics operations, types of data and identifiers, topics and formats. *Bioinformatics*, 29(10):1325–1332, 2013.

[71] Krzysztof Janowicz and Michael Compton. The Stimulus-Sensor-Observation Ontology Design Pattern and its Integration into the Semantic Sensor Network Ontology. In *3rd International Workshop on Semantic Sensor Networks*, volume 668 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010. November 7, Shanghai, China.

[72] Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, and Yujiao Zhou. Logmap 2.0: towards logic-based, scalable and interactive ontology matching. In *The 4th International Workshop on Semantic Web Applications and Tools for the Life Sciences, SWAT4LS 2011*, pages 45–46. ACM, 2011. London, United Kingdom, December 07-09, 2011.

[73] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, Bernardo Cuenca Grau, and James A. Hendler. Swoop: A Web Ontology Editing Browser. *Journal of Web Semantics*, 4(2):144–153, 2006.

[74] Yevgeny Kazakov, Markus Krötzsch, and Frantisek Simancik. ELK reasoner: Architecture and evaluation. In Ian Horrocks, Mikalai Yatskevich, and Ernesto Jiménez-Ruiz, editors, *1st International Workshop on OWL Reasoner Evaluation (ORE-2012)*, volume 858 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012. Manchester, UK, July 1st.

[75] C. M. Keet, A. Lawrynowicz, C. d'Amato, A. Kalousis, P. Nguyen, R. Palma, R. Stevens, and M. Hilario. The data mining optimization ontology. *Web Semantics: Science, Services and Agents on the World Wide Web*, 32:43–53, 2015.

[76] C. Maria Keet. Using abstractions to facilitate management of large ORM models and ontologies. In *OTM Workshops*, volume 3762 of *Lecture Notes in Computer Science*, pages 603–612, 2005. Agia Napa, Cyprus, October 31 - November 4.

[77] C. Maria Keet. Enhancing comprehension of ontologies and conceptual models through abstractions. In *10th Congress of the Italian Association for Artificial Intelligence (AI*IA 2007)*, volume 4733 of *Lecture Notes in Computer Science*, pages 813–821. Springer, 2007. Rome, Italy, September 10-13.

[78] C. Maria Keet, Claudia d'Amato, Zubeida Casmod Khan, and Agnieszka Lawrynowicz. Exploring reasoning with the DMOP ontology. In *3rd Workshop on Ontology Reasoner Evaluation (ORE'14)*, CEUR Workshop Proceedings, pages 64–70. CEUR-WS.org, 2014. July 1, Vienna, Austria.

[79] C. Maria Keet, A. Lawrynowicz, C. d'Amato, and M. Hilario. Modeling issues and choices in the Data Mining OPtimisation Ontology. In *8th Workshop on OWL: Experiences and Directions (OWLED'13)*, volume 1080 of *CEUR-WS*, 2013. 26-27 May 2013, Montpellier, France.

[80] Zubeida Khan and C. Maria Keet. The foundational ontology library ROMULUS. In *Third International Conference on Model & Data Engineering (MEDI'13)*, volume 8216 of *LNCS*, pages 200–211. Springer, 2013. September 25-27, 2013, Amantea, Italy.

[81] Zubeida Khan and C. Maria Keet. Feasibility of automated foundational ontology interchangeability. In *19th International Conference on Knowledge Engineering and Knowledge Management (EKAW'14)*, volume 8876 of *LNAI*, pages 225–237. Springer, 2014. 24 - 28 November 2014, Linköping, Sweden.

[82] Zubeida Casmod Khan. Evaluation metrics in ontology modules. In *29th International Workshop on Description Logics (DL'16)*, volume 1577 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016. 22-25 April 2016, Cape Town, South Africa.

[83] Zubeida Casmod Khan and C. Maria Keet. Automatic modularisation with algorithms for abstraction and expressiveness. (in preparation for submission to an international conference).

[84] Zubeida Casmod Khan and C. Maria Keet. SUGOI: automated ontology interchangeability. In Patrick Lambrix, Eero Hyvönen, Eva Blomqvist, Valentina Presutti, Guilin Qi, Uli Sattler, Ying Ding, and Chiara Ghidini, editors, *Knowledge Engineering and Knowledge Management - EKAW 2014 Satellite Events*, volume 8982 of *Lecture Notes in Computer Science*, pages 150–153. Springer, 2014. Linköping, Sweden, November 24-28, 2014. Revised Selected Papers.

[85] Zubeida Casmod Khan and C. Maria Keet. An empirically-based framework for ontology modularisation. *Applied Ontology*, 10(3-4):171–195, 2015.

[86] Zubeida Casmod Khan and C. Maria Keet. Toward a framework for ontology modularity. In *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT'15)*. ACM Conference Proceedings, 2015. 28-30 September 2015, Stellenbosch, South Africa.

[87] Zubeida Casmod Khan and C. Maria Keet. Dependencies between modularity metrics towards improved modules. In *20th International Conference on Knowledge Engineering and Knowledge Management (EKAW'16)*, Lecture Notes in Artificial Intelligence LNAI, pages 19–23. Springer, 2016. 19-23 November 2016, Bologna, Italy.

[88] Zubeida Casmod Khan and C. Maria Keet. ROMULUS: the repository of ontologies for multiple uses populated with mediated foundational ontologies. *Journal of Data Semantics*, 5(1):19–36, 2016.

[89] Zubeida Casmod Khan and C. Maria Keet. Automated ontology interchangeability towards improved modules. In *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT'17)*. ACM Conference Proceedings, 2017. 26-28 September 2017, Bloemfontein, Free State, South Africa.

[90] Zubeida Casmod Khan, C. Maria Keet, Pablo R. Fillottrani, and Karina Cenci. Experimentally motivated transformations for intermodel links between conceptual models. In *20th Conference on Advances in Databases and Information Systems (ADBIS'16)*, volume 9809 of *Lecture Notes in Computer Science LNCS*, pages 104–118. Springer, 2016. August 28-31, Prague, Czech Republic.

[91] Boris Konev, Carsten Lutz, Dirk Walther, and Frank Wolter. Semantic modularity and module extraction in description logics. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikolaos M. Avouris, editors, *Proceedings of 18th European Conference on Artificial Intelligence (ECAI'08)*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 55–59. IOS Press, 2008. Patras, Greece, July 21-25, 2008.

[92] Boris Konev, Carsten Lutz, Dirk Walther, and Frank Wolter. Formal Properties of Modularisation. In *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, volume 5445 of *Lecture Notes in Computer Science*, pages 25–66. Springer, 2009.

[93] Markus Krötzsch. OWL 2 profiles: An introduction to lightweight ontology languages. In *Reasoning Web. Semantic Technologies for Advanced Query Answering - 8th International Summer School 2012*, volume 7487 of *Lecture Notes in Computer Science*, pages 112–183. Springer, 2012. Vienna, Austria, September 3-8, 2012.

[94] O. Kutz, C. Lutz, F. Wolter, and M. Zakharyaschev. E-Connections of Abstract Description Systems. *Artificial Intelligence*, 156(1):1–73, 2004.

[95] Stephen D. Larson, Lisa Fong, Amarnath Gupta, Christopher Condit, William J. Bug, and Maryann E. Martone. A formal ontology of subcellular neuroanatomy. *Front. Neuroinform.*, 2007, 2007.

[96] Stephen D Larson, Lisa L Fong, Amarnath Gupta, Christopher Condit, William J Bug, and Maryann E Martone. A Formal Ontology of Subcellular Neuroanatomy. *Front Neuroinformatics*, 1:3, 2007.

[97] Dennis Lee, Ronald Cornet, Francis Lau, and Nicolette de Keizer. A survey of SNOMED CT implementations. *Journal of Biomedical Informatics*, 46(1):87 – 96, 2013.

[98] Frank Loebe. Requirements for logical modules. In Peter Haase, Vasant Honavar, Oliver Kutz, York Sure, and Andrei Tamilin, editors, *The 1st International Workshop on Modular Ontologies (WoMO'06)*, volume 232 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006. ISWC'06 November 5, Athens, Georgia, USA.

[99] Steffen Lohmann, Paloma Díaz, and Ignacio Aedo. MUTO: the modular unified tagging ontology. In Chiara Ghidini, Axel-Cyrille Ngonga Ngomo, Stefanie N. Lindstaedt, and Tassilo Pellegrini, editors, *Proceedings of the 7th International Conference on Semantic Systems (I-SEMANTICS '11)*, ACM International Conference Proceeding Series, pages 95–104. ACM, 2011. Graz, Austria, September 7-9.

[100] Bill MacCartney, Sheila A. McIlraith, Eyal Amir, and Tomás E. Uribe. Practical Partition-Based Theorem Proving for Large Knowledge Bases. In *Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 89–98. Morgan Kaufmann, 2003. Aug 9-15, Acapulco, Mexico.

[101] I. Mani. A theory of granularity and its application to problems of polysemy and underspecification of meaning. In A.G. Cohn, L.K. Schubert, and S.C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR98)*, pages 245–255. San Mateo: Morgan Kaufmann, 1998.

[102] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari. Ontology library. WonderWeb Deliverable D18 (ver. 1.0, 31-12-2003)., 2003. `http://www.loa.istc.cnr.it/old/Papers/D18.pdf` last accessed: 20 June 2017.

[103] Dave McComb. Gist: The minimalist upper ontology. 2010 Semantic Technology Conference, 2010. June 21-25 2010, San Francisco, CA.

[104] Peter Mika. Flink: Semantic web technology for the extraction and analysis of social networks. *Journal of Web Semantics*, 3(2-3):211–223, 2005.

[105] Eleni Mikroyannidi, Alan Rector, and Robert Stevens. Abstracting and generalising the foundational model anatomy (fma) ontology. In *Proceedings of the bio-ontologies 2009 conference*, 2009. Stockholm, June 2009.

[106] R. Mizoguchi. YAMATO: Yet Another More Advanced Top-level Ontology. In *Proceedings of the Sixth Australasian Ontology Workshop*, Conferences in Research and Practice in Information, pages 1–16, 2010. Sydney : ACS.

[107] Till Mossakowski, Oliver Kutz, and Mihai Codescu. Ontohub: A semantic repository for heterogeneous ontologies. In *Proceedings of the Theory Day in Computer Science Satellite workshop (DACS-2014)*, 2014. University of Bucharest, September 15-16, 2014.

[108] Till Mossakowski, Oliver Kutz, Mihai Codescu, and Christoph Lange. The distributed ontology, modeling and specification language. In Chiara Del Vescovo, Torsten Hahmann, David Pearce, and Dirk Walther, editors, *Proceedings of the 7th International Workshop on Modular Ontologies (WoMO'13)*, volume 1081 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013. Corunna, Spain, September 15.

[109] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, Carsten Lutz, et al. OWL 2 web ontology language profiles. *W3C recommendation*, 27:61, 2009.

[110] Mark A. Musen. The protégé project: a look back and a look forward. *AI Matters*, 1(4):4–12, 2015.

[111] David Newman, Sean Bechhofer, and David De Roure. myExperiment: An ontology for e-Research. In *Proceedings of the Workshop on Semantic Web Applications in Scientific Discourse (SWASD 2009)*, volume 523 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009. Washington DC, USA, October 26.

[112] I. Niles and A. Pease. Towards a standard upper ontology. In Chris Welty and Barry Smith, editors, *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, 2001. Ogunquit, Maine, October 17-19, 2001.

[113] Natalya Fridman Noy and Mark A. Musen. Specifying Ontology Views by Traversal. In *Third International Semantic Web Conference (ISWC'04)*, volume 3298 of *Lecture Notes in Computer Science*, pages 713–725. Springer, 2004. Nov 7-11, Hiroshima, Japan.

[114] Natalya Fridman Noy and Mark A. Musen. Traversing ontologies to extract views. In *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, volume 5445 of *Lecture Notes in Computer Science*, pages 245–260. Springer, 2009.

[115] Sunju Oh and Joongho Ahn. Ontology module metrics. In *International Conference on e-Business Engineering, (ICEBE'09)*, pages 11–18. IEEE Computer Society, 2009. Macau, China, 21-23 October.

[116] Sunju Oh, Heon Y Yeom, and Joongho Ahn. Evaluating ontology modularization approaches. In *The 8th International Conference on Frontiers of Information Technology*, page 6. ACM, 2010. December 21 -23, 2010, Islamabad, Pakistan.

[117] Sunju Oh and Heon Young Yeom. Evaluation criteria ontology modularization tools. In *The IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Workshops (WI-IAT '11)*, pages 365–368. IEEE Computer Society, 2011. Lyon, France, August 22-27, 2011.

[118] Sunju Oh, Heon Young Yeom, and Joongho Ahn. Cohesion and coupling metrics for ontology modules. *Information Technology and Management*, 12(2):81–96, 2011.

[119] Anthony M. Orme, Haining Yao, and Letha H. Etzkorn. Coupling metrics for ontology-based systems. *IEEE Software*, 23(2):102–108, 2006.

[120] P. Pandurang Nayak and A.Y. Levy. A semantic theory of abstractions. In C. Mellish, editor, *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 196–203. San Mateo: Morgan Kaufmann, 1995.

[121] Pance Panov, Saso Dzeroski, and Larisa N. Soldatova. OntoDM: An ontology of data mining. In *Workshops Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008)*, pages 752–760. IEEE Computer Society, 2008. December 15-19, Pisa, Italy.

[122] Symeon Papadopoulos, Yiannis Kompatsiaris, Athena Vakali, and Ploutarchos Spyridonos. Community detection in social media - performance and application considerations. *Data Mining and Knowledge Discovery*, 24(3):515–554, 2012.

[123] Christine Parent and Stefano Spaccapietra. An Overview of Modularity. In Heiner Stuckenschmidt, Christine Parent, and Stefano Spaccapietra, editors, *Modular Ontologies*, volume 5445 of *Lecture Notes in Computer Science*, pages 5–23. Springer Berlin Heidelberg, 2009.

[124] Jyotishman Pathak, Thomas M. Johnson, and Christopher G. Chute. Survey of modular ontology techniques and their applications in the biomedical domain. *Integrated Computer-Aided Engineering*, 16(3):225–242, 2009.

[125] Heiko Paulheim. On applying matching tools to large-scale ontologies. In Pavel Shvaiko, Jérôme Euzenat, Fausto Giunchiglia, and Heiner Stuckenschmidt, editors, *Proceedings of the 3rd International Workshop on Ontology Matching (OM'08)*, volume 431 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008. Karlsruhe, Germany, October 26.

[126] Clara Pizzuti. Community detection in social networks with genetic algorithms. In Conor Ryan and Maarten Keijzer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008)*, pages 1137–1138. ACM, 2008. Atlanta, GA, USA, July 12-16.

[127] Valentina Presutti. Type of entities ontology design pattern. `http://ontologydesignpatterns.org/wiki/Submissions:Types_of_entities` last accessed: 20 June 2017, 2010. Online (March 2010).

[128] Alan L. Rector, Jeremy Rogers, Pieter E. Zanstra, and Egbert J. van der Haring. OpenGALEN: Open source medical terminology and tools. In *American Medical Informatics Association Annual Symposium (AMIA'03)*. AMIA, 2003. Washington, DC, USA, November 8-12.

[129] Rachel L. Richesson, James E. Andrews, and Jeffrey P. Krischer. Use of SNOMED CT to Represent Clinical Research Data: A Semantic Characterization of Data Items on Case Report Forms in Vasculitis Research. *JAMIA*, 13(5):536–546, 2006.

[130] Lior Rokach and Oded Maimon. *Clustering Methods*, pages 321–352. Springer US, 2005.

[131] Ana Armas Romero, Mark Kaminski, Bernardo Cuenca Grau, and Ian Horrocks. Module extraction in expressive ontology languages via datalog reasoning. *Journal of Artificial Intelligence Research (JAIR)*, 55:499–564, 2016.

[132] Marco Rospocher. An ontology for personalized environmental decision support. In *Formal Ontology in Information Systems FOIS '14*, pages 421–426, 2014. September, 22-25, 2014, Rio de Janeiro, Brazil.

[133] Cornelius Rosse and José L. V. Mejino, Jr. A reference ontology for biomedical informatics: the Foundational Model of Anatomy. *Journal of Biomedical Informatics*, 36(6):478–500, 2003.

[134] Satya Sanket Sahoo, Samden D. Lhatoo, Deepak K. Gupta, Licong Cui, Meng Zhao, Catherine P. Jayapandian, Alireza Bozorgi, and Guo-Qiang Zhang. Epilepsy and seizure ontology: towards an epilepsy informatics infrastructure for clinical research and patient care. *Journal of American Medical Informatics Association*, 21(1):82–89, 2014.

[135] Ulrike Sattler, Thomas Schneider, and Michael Zakharyaschev. Which kind of module should I extract? In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Proceedings of the 22nd International Workshop on Description Logics (DL'09)*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009. Oxford, UK, July 27-30, 2009.

[136] Ansgar Scherp, Carsten Saathoff, Thomas Franz, and Steffen Staab. Designing core ontologies. *Applied Ontology*, 6(3):177–221, 2011.

[137] Anne Schlicht and Heiner Stuckenschmidt. Towards structural criteria for ontology modularization. In *Proceedings of the 1st International Workshop on Modular Ontologies, (WoMO'06)*, volume 232 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006. ISWC'06 November 5, 2006, Athens, Georgia, USA.

[138] Anne Schlicht and Heiner Stuckenschmidt. A flexible partitioning tool for large ontologies. In *International Conference on Web Intelligence (WI 2008)*, pages 482–488. IEEE, 2008. 9-12 December, Sydney, NSW, Australia.

[139] Anne Schlicht and Heiner Stuckenschmidt. A flexible partitioning tool for large ontologies. In *International Conference on Web Intelligence (WI'08)*, pages 482–488. IEEE Computer Society, 2008. 9-12 December, Sydney, NSW, Australia.

[140] Stefan Schulz and Martin Boeker. BioTopLite: An upper level ontology for the life sciences. evolution, design and application. In *Informatik 2013, 43. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Informatik angepasst an Mensch, Organisation und Umwelt*, volume 220 of *LNI*, pages 1889–1899. GI, 2013. 16-20. September 2013, Koblenz.

[141] Julian Seidenberg. Web ontology segmentation: Extraction, transformation, evaluation. In *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, volume 5445 of *Lecture Notes in Computer Science*, pages 211–243. Springer, 2009.

[142] Stefanie Seltmann, Harald Stachelscheid, Alexander Damaschun, Ludger Jansen, Fritz Lekschas, Jean-Fred Fontaine, Throng-Nghia Nguyen-Dobinsky, Ulf Leser, and Andreas Kurtz. CELDA - an ontology for the comprehensive representation of cells in complex systems. *BMC Bioinformatics*, 14:228, 2013.

[143] B. Smith, M. Ashburner, C. Rosse, J. Bard, W. Bug, W. Ceusters, L.J. Goldberg, K. Eilbeck, A. Ireland, C.J. Mungall, The OBI Consortium, N. Leontis, A.B. Rocca-Serra, A. Ruttenberg, S-A. Sansone, M. Shah, P.L. Whetzel, and S. Lewis. The OBO Foundry: Coordinated evolution of ontologies to support biomedical data integration. *Nature Biotechnology*, 25(11):1251–1255, 2007.

[144] Holger Stenzhorn, Elena Beisswanger, and Stefan Schulz. Towards a top-domain ontology for linking biomedical ontologies. In *Proceedings of the 12th World Congress on Health Medical Informatics - Building Sustainable Health Systems (MEDINFO'07)*, volume 129 of *Studies in Health Technology and Informatics*, pages 1225–1229. IOS Press, 2007. 20-24 August, Brisbane, Australia.

[145] Robert Stevens and Phillip Lord. Semantic publishing of knowledge about amino acids. In Alexander Garca Castro, Christoph Lange, Frank van Harmelen, and Benjamin Good, editors, *Proceedings of the 2nd Workshop on Semantic Publishing*, volume 903 of *CEUR Workshop Proceedings*, pages 45–48. CEUR-WS.org, 2012. Hersonissos, Crete, Greece, May 28th.

[146] Anselm Strauss and Juliet Corbin. Grounded theory methodology. *Handbook of qualitative research*, 17:273–85, 1994.

[147] Heiner Stuckenschmidt and Michel C. A. Klein. Structure-based partitioning of large concept hierarchies. In *Third International Semantic Web Conference (ISWC'04)*, volume 3298 of *Lecture Notes in Computer Science*, pages 289–303. Springer, 2004. Hiroshima, Japan, November 7-11, 2004.

[148] Heiner Stuckenschmidt and Michel C. A. Klein. Reasoning and change management in modular ontologies. *Data Knowledge Engineering*, 63(2):200–223, 2007.

[149] Heiner Stuckenschmidt and Anne Schlicht. Structure-Based Partitioning of Large Ontologies. In *Modular Ontologies: Concepts, Theories and Techniques*

for Knowledge Modularization, volume 5445 of *Lecture Notes in Computer Science*, pages 187–210. Springer, 2009.

[150] Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, and Mariano Fernández-López. The neon methodology for ontology engineering. In *Ontology Engineering in a Networked World.*, pages 9–34. Springer, 2012.

[151] Samir Tartir, I Budak Arpinar, Michael Moore, Amit P Sheth, and Boanerges Aleman-Meza. OntoQA: Metric-based ontology quality analysis. *IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources*, 9, 2005.

[152] Dhavalkumar Thakker, Vania Dimitrova, Lydia Lau, Ronald Denaux, Stan Karanasios, and Fan Yang-Turner. A priori ontology modularisation in ill-defined domains. In Chiara Ghidini, Axel-Cyrille Ngonga Ngomo, Stefanie N. Lindstaedt, and Tassilo Pellegrini, editors, *Proceedings the 7th International Conference on Semantic System (I-SEMANTICS'11)*, ACM International Conference Proceeding Series, pages 167–170. ACM, 2011. Graz, Austria, September 7-9, 2011.

[153] Hendrik Thomas, Rob Brennan, and Declan O'Sullivan. Using the OM2R metadata model for ontology mapping reuse for the ontology alignment challenge - a case study. In *Proceedings of the 7th International Workshop on Ontology Matching (OM'12)*, volume 946 of *CEUR-WS*, 2012. Boston, MA, USA, November 11.

[154] Dmitry Tsarkov. Improved Algorithms for Module Extraction and Atomic Decomposition. In Yevgeny Kazakov, Domenico Lembo, and Frank Wolter, editors, *The 2012 International Workshop on Description Logics (DL '12)*, volume 846 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012. Rome, Italy, June 7-10.

[155] Venkata Krishna Chaitanya Turlapati and Sreenivasa Kumar Puligundla. Efficient module extraction for large ontologies. In Pavel Klinov and Dmitry Mouromtsev, editors, *4th International Conference on Knowledge Engineering and the Semantic Web (KESW'13)*, volume 394 of *Communications in Computer and Information Science*, pages 162–176. Springer Berlin Heidelberg, 2013.

[156] Chiara Del Vescovo. The modular structure of an ontology: Atomic decomposition towards applications. In *Proceedings of the 24th International Workshop on Description Logics (DL 2011)*, volume 745 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011. Barcelona, Spain, July 13-16.

[157] Chiara Del Vescovo, Damian Gessler, Pavel Klinov, Bijan Parsia, Ulrike Sattler, Thomas Schneider, and Andrew Winget. Decomposition and Modular Structure of BioPortal Ontologies. In *10th International Conference on The International*

Semantic Web Conference (ISWC'10), volume 7031 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2011. October 23-27, Bonn, Germany.

[158] Chiara Del Vescovo, Pavel Klinov, Bijan Parsia, Ulrike Sattler, Thomas Schneider, and Dmitry Tsarkov. Empirical study of logic-based modules: Cheap is cheerful. In *Proceedings of the 26th International Workshop on Description Logics (DL 2013)*, volume 1014 of *CEUR Workshop Proceedings*, pages 144–155. CEUR-WS.org, 2013. Ulm, Germany, July 23 - 26.

[159] Chiara Del Vescovo, Bijan Parsia, Ulrike Sattler, and Thomas Schneider. The modular structure of an ontology: Atomic decomposition. In Toby Walsh, editor, *The 22nd International Joint Conference on Artificial Intelligence (IJCAI '11)*, pages 2232–2237. IJCAI/AAAI, 2011. Barcelona, Catalonia, Spain, July 16-22, 2011.

[160] Markel Vigo, Caroline Jay, and Robert Stevens. Design insights for the next wave ontology authoring tools. In *Conference on Human Factors in Computing Systems (CHI'14)*, pages 1555–1558. ACM, 2014. Toronto, ON, Canada, April 26 - May 01, 2014.

[161] Yimin Wang, Peter Haase, and Jie Bao. A survey of formalisms for modular ontologies. In *Workshop on Semantic Web for Collaborative Knowledge Acquisition*, 2007.

[162] Leo Wanner, Marco Rospocher, Stefanos Vrochidis, Harald Bosch, Ulrich Bgel, Gerard Casamayor, Thomas Ertl, Ioannis Kompatsiaris, Tarja Koskentalo, Simon Mille, Jrgen Mograber, Anastasia Moumtzidou, Maria Myllynen, Emanuele Pianta, Horacio Saggion, Luciano Serafini, Virpi Tarvainen, Sara Tonelli, and Bruno Kessler. Personalized environmental service configuration and delivery orchestration: The PESCaDO demonstrator. In *In Proceedings of the 9th Extended Semantic Web Conference (ESWC 2012)*, 2012. Heraklion, Crete, Greece, May 27-31 (demo).

[163] Patricia L. Whetzel, Natalya Fridman Noy, Nigam H. Shah, Paul R. Alexander, Csongor Nyulas, Tania Tudorache, and Mark A. Musen. BioPortal: enhanced functionality via new web services from the national center for biomedical ontology to access and use ontologies in software applications. *Nucleic Acids Research*, 39(Web-Server-Issue), 2011.

[164] Haining Yao, Anthony M Orme, and Letha Etzkorn. Cohesion metrics for ontology design and application. *Journal of Computer science*, 1(1):107, 2005.