ROLE AND APPLICATION OF ONTOLOGY DESIGN PATTERNS IN BIO-ONTOLOGIES

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2009

By
Mikel Egaña Aranguren
School of Computer Science

Contents

List of Tables

List of Figures

Abstract

Knowledge Representation (KR) languages such as OWL (Web Ontology Language), having precise semantics, offer the possibility of computationally exploiting biological knowledge, by codifying it in the axioms of bio-ontologies. Bio-ontologies are widely used in life sciences for knowledge management. Knowledge is, however, often represented in bio-ontologies without following rigorous principles of modelling and the resulting bio-ontologies are axiomatically lean. Therefore knowledge cannot be computationally exploited for integrity checking, hypothesis generation, consistency maintenance, integration, or rich querying.

A solution that can contribute to the rigorous modelling and axiomatic richness of bio-ontologies is the use of Ontology Design Patterns (ODPs). ODPs are thoroughly documented and efficient solutions for recurrent problems encountered when building ontologies. Therefore ODPs act as guides on how to use KR languages for creating ontology fragments that have well known advantages and side effects.

In order for ODPs to be efficiently accessed by bio-ontologists, an online catalogue of ODPs has been created, describing different ODPs using a consistent documentation schema. Such ODPs, apart from being accessed, can be applied automatically with the Ontology PreProcessor Language (OPPL), as OPPL makes it possible to encapsulate ODPs in scripts to be executed on OWL ontologies, making the application of ODPs replicable and flexible.

The infrastructure for applying ODPs formed by the catalogue and OPPL has been used for applying ODPs in bio-ontologies like the Cell Type Ontology. The results of such application have been evaluated to assess the applied ODPs and the change on ontology quality. Side effects of the usage of ODPs have been spotted, but in general terms, when ODPs are applied, bio-ontology quality improves, especially in the areas of structure, functionality, and maintainability. Therefore it is concluded that the use of ODPs contributes to the creation of more rigorous and axiomatically richer bio-ontologies, providing new possibilities for knowledge management in life sciences.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Dedication

Amarentzat

Uztaila hontan beteko ditut
nik nahi baino urte geiho
ilea galtzearren nik ez dut
ematen gizon serio
ene hitz garbienak ere ez du
inon ezer ez balio
nire buruari sekulan ez dio
inork jarriko prezio.

Gau ixil hontan ikusten zaitut
iparraldeko izarra
hamabost urteko neska baten
ametsa bezain ederra
urruti zaude mundu hontatik
librea eta bakarra
inoren menpe kantatzen duzu ta
zure boza da zilarra.

Bernardo Atxaga - Ruper Ordorika

Acknowledgements

This work was funded by the University of Manchester and the Engineering and Physical Sciences Research Council (EPSRC). Other institutions and projects have additionally funded different parts of the work: the Marie Curie-Early Stage Training programme funded my visit to the VIB (Vlaams Instituut voor Biotechnologie), the Ontogenesis Network funded the meetings for the Normalisation experiment, and the European Network of Excellence in Semantic Mining funded my visit to the EBI (European Bioinformatics Institute).

I got prompt and vital technical help from Nick Drummond, Chris Wroe and Olivier Dameron. I have used the services provided by `SinDominio.Net` for my own benefit without contributing as I should. I have used and adapted excellent free software during these years, especially GNU/Linux, and I would like to thank the work of the developers, which usually do not get the due public acknowledgements.

In academic terms, I would like to thank foremost the support and understanding of my supervisor, Robert Stevens, who was also encouraging and challenging. I would like to thank group leaders who welcomed me to their excellent groups: Midori Harris of the GO editors' team at the EBI, and especially Martin Kuiper of the Computational Biology group at the VIB.

Plenty of people contributed with challenging discussions or interesting ideas to this work: Jesualdo Tomás Fernández Breis, Amelia Ireland, Chris Mungall, Alan Rector, Martin Kuiper, Alan Ruttenberg, Chris Wroe, Helen Parkinson, James Malone, Andy Gibson, Simon Jupp, Luigi Iannone, Phil Lord, Uli Sattler, Vladimir Mironov, Rinke Hoekstra, Matthew Horridge, Bijan Parsia, David Shotton, Midori Harris, Jon Bard and many more. Especial acknowledgements go to Erick Antezana for being a challenging, trustworthy and inspiring colleague from whom I have learned countless new concepts in computer science.

My friends have not forgotten me, even though I have seldom been available, and I am really grateful to them for that. My friends from Bilbo have been there for me in these difficult years and we have shared wonderful experiences: Barandi, Ibai, Santi, Aurre, and everyone else. I would like to also thank my Mancunian friends, Dan, Phil and Conor, for checking whether I was alive every now and then.

My family was very supportive during these years, especially my father Juan Ramón and my brother Eneko. My mother, Amaia, had an amazing faith in me till her very last moment, and she was always supportive even though she had to confront a terminal illness.

My biggest acknowledgement goes to my partner and friend Maider, who was supportive and patient, even to the extent of politely pretending to be interested in the explanations about this work. She has shown me during these years what makes real changes, and what truly worthy people are made of.

The author

Mikel Egaña Aranguren graduated from the University of Basque Country with a degree in biology in 2004. Whilst an undergraduate he was awarded an Erasmus scholarship by the European Union, which allowed him to spend half a term in Canterbury Christ Church University College, UK, studying environmental biology.

In 2004-2005, he completed an MSc in Bioinformatics, at the University of Manchester, UK, obtaining a distinction mark. The MSc thesis was entitled "Improving the Structure of the Gene Ontology" and it has been cited in various journal publications[1].

He has been doing research since 2005 as a PhD student in the Bio Health Informatics Group at the University of Manchester, culminating in this thesis. During this period he completed two research visits: a 5 months visit to the Computational Biology group of the Vlaams Instituut voor Biotechnologie (VIB), Belgium, funded by the Marie Curie-Early Stage Training program, and a one month visit to the GO editors' team of the European Bioinformatics Institute (EBI), UK, funded by the European Network of Excellence in Semantic Mining.

He has been working in the following areas:

The Gene Ontology Next Generation[2] (GONG) project provides a pipeline for dissecting the syntactic structure of GO term names to add new, richer axioms and as a result exploit Automated reasoning, as described in [73]. In the same context, a conceptual bridge between the OBO (Open Biomedical Ontologies) format and OWL, as a result of analysing the semantics and assumptions behind each of them, was presented in [72].

The Cell Cycle Ontology[3] (CCO) is a domain ontology that was built to represent the aspects of the cell cycle that were absent in GO, and to connect such aspects to

[1] `http://scholar.google.com/scholar?hl=es&lr=&cites=12900445016847716696`

[2] `http://www.gong.manchester.ac.uk`

[3] `http://cellcycleontology.org`

other resources such as UniProt [7]. ONTO-PERL[4] is a Perl API for working with OBO ontologies, developed for automating the creation and management of CCO, as described in [5].

Ontology Design Patterns (ODPs) are documented and tested best practices of ontology engineering. A description of the concept of ODPs and their application on CCO was presented in [10], and their application on GO was presented in [30]. Some ODPs for modelling biological knowledge were collected in an ODPs public catalogue[5]. The limitations and capabilities of OWL for representing biological knowledge were analysed in [108], including some ODPs.

The Ontology PreProcessor Language[6] (OPPL), a scripting language for programmatically modifying OWL ontologies, was presented in [29]. Its usage for applying different ODPs for modelling modifiers was described in [30].

Semantic Systems Biology[7] (SSB) is a project that explores the use of Semantic Web technology to enhance knowledge management on Systems Biology research. BioGateway[8], which is part of SSB, is an RDF triple store that integrates many different bioinformatics resources [6].

He has given various talks (University of Basque Country 2005, Bio-ontologies SIG at ISMB 2007, OWLed 2008 DC, EKAW 2008, University of Murcia 2008) and has taught OWL advanced tutorials for the biological domain[9]. He has done reviewing work for ONTORACT 2008.

[4]`http://search.cpan.org/~easr/`

[5]`http://odps.sf.net/`

[6]`http://oppl.sourceforge.net/`

[7]`http://www.semantic-systems-biology.com/`

[8]`http://www.semantic-systems-biology.org/biogateway`

[9]`http://www.co-ode.org/resources/tutorials/bio/`

Publications

0.1 Journals

Erick Antezana, Ward Blondé, **Mikel Egaña**, Alistair Rutherford, Robert Stevens, Bernard De Baets, Vladimir Mironov, Martin Kuiper. Structuring the Life Science Resourceome for Semantic Systems Biology: Lessons from the BioGateway Project. *BMC bioinformatics*, accepted for publication.

Erick Antezana, **Mikel Egaña**, Bernard De Baets, Ward Blondé, 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* 2009, 10(5):R58.

Mikel Egaña Aranguren, Erick Antezana, Martin Kuiper, Robert Stevens. Ontology Design Patterns for bio-ontologies: a case study on the Cell Cycle Ontology. *BMC bioinformatics* 2008, 9(Suppl 5):S1.

Mikel Egaña Aranguren, Chris Wroe, Carole Goble, Robert Stevens. *In situ* migration of handcrafted ontologies to Reason-able Forms. *Data & Knowledge Engineering* 2008, 66, 147-162.

E. Antezana, **M. Egaña**, B. De Baets, M. Kuiper and V. Mironov. ONTO-PERL: An API supporting the development and analysis of bio-ontologies. *Bioinformatics* 2008, 24(6):885-887.

Mikel Egaña Aranguren, Sean Bechoffer, Phillip Lord, Ulrike Sattler and Robert Stevens. Understanding and using the meaning of statements in a bio-ontology: recasting the Gene Ontology in OWL. *BMC Bioinformatics* 2007, 8:57.

Robert Stevens, **Mikel Egaña Aranguren**, Katy Wolstencroft, Ulrike Sattler, Nick Drummond and Matthew Horridge. Using OWL to Model Biological Knowledge. *International Journal of Human Computer Studies* 2006, 65:7, 583-594.

0.2 Conference proceedings

Mikel Egaña, Alan Rector, Robert Stevens, Erick Antezana. Applying Ontology Design Patterns in bio-ontologies. EKAW 2008, LNCS 5268, pp. 7-16.

0.3 Workshops

Mikel Egaña, Robert Stevens, Erick Antezana. Transforming the Axiomisation of Ontologies: The Ontology Pre-Processor Language. OWLed 2008, Washington DC, USA.

Jesualdo Tomás Fernández-Breis, **Mikel Egaña Aranguren**, Robert Stevens. A Quality Evaluation Framework for Bio-Ontologies. ICBO 2009, Buffalo, USA. Accepted.

About this document

0.4 Electronic version

An electronic version of this document is available online[10], containing clickable hyperlinks and bookmarks for navigating the document.

0.5 Abbreviations

ODP: Ontology Design Pattern.

KR: Knowledge Representation.

WWW: World Wide Web.

OWL: Ontology Web Language.

LSSW: Life Sciences Semantic Web.

GO: Gene Ontology.

OBO: Open Biomedical Ontologies.

SO: Sequence Ontology.

OBI: Ontology for Biomedical Investigations.

OPPL: Ontology PreProcessor Language.

CL: Cell Type Ontology.

CCO: Cell Cycle Ontology.

[10]`http://www.gong.manchester.ac.uk/thesis.pdf.zip`

W3C: World Wide Web Consortium.

HTML: HyperText Markup Language.

W3C HCLS: Semantic Web Health Care and Life Sciences Interest Group.

QCR: Qualified Cardinality Restriction.

XML: eXtensible Markup language.

RDF: Resource Description Framework.

URI: Uniform Resource Identifier.

SPARQL: SPARQL Protocol and RDF Query Language.

RDFS: Resource Description Framework Schema.

API: Application Programming Interface.

MOS: Manchester OWL Syntax.

DL: Description Logics.

TBox: Terminological Box.

ABox: Assertional Box.

UNA: Unique Name Assumption.

OWA: Open World Assumption.

GOA: Gene Ontology Annotation.

KB: Knowledge Base.

BFO: Basic Formal Ontology.

RO: Relations Ontology.

OBOL: OBO Language.

LODP: Logical Ontology Design Pattern.

CODP: Concept Ontology Design Pattern.

CODEP: Content Ontology Design Pattern.

UML: Unified Modelling Language.

W3C BPD: Semantic Web Best Practices and Deployment Working Group.

XSLT: eXtensible Stylesheet Language Transformations.

GrOWL: Graphical OWL.

GONG: Gene Ontology Next Generation.

nCL: normalised Cell Type Ontology.

PATO: Phenotypic Quality Ontology.

0.6 Glossary

The following definitions have been adapted from [72]: Knowledge Representation (KR) Language, Automated reasoning, Soundness, Completeness, Description Logics (DL), Monotonic.

Knowledge Representation (KR) Language: A language used to represent information about the world or a particular domain. KR Languages are often formal languages, with a precise semantics for the operators in the language, allowing for the use of deduction or automated reasoning (see Automated reasoning). One of the most used KR languages is the Web Ontology Language (OWL).

Ontology: A representation of a concrete knowledge domain, created using a KR language and implemented in a format that can be computationally processed. The term "model", as an ideal representation of some real objects, is usually used as a synonym of ontology, and "modelling" as a synonym of creating or maintaining an ontology. However, the term "model" has a different and very precise meaning in the context of model-theoretic semantics (see Semantics) [31].

Axiom: An axiom is a logical statement relating different entities of a particular knowledge domain, expressed using a KR language. Axioms can be logically entailed by other axioms, in which case they will be inferred by a reasoner (See Automated reasoning). The more expressive a KR language, the more complex and hence computationally expensive axioms can be created. An ontology is composed by a collection of axioms, and hence an ontology with complex axioms has a rich axiomisation: it richly describes the knowledge domain.

Semantics: In model-theoretic semantics, the statements are mapped to a set-theoretic structure with precise formal properties. Thus, the "meaning" of an statement is provided by a formalism shared by different scientists, instead of relying on such meaning being arbitrarily interpreted by each scientist [31]. In OWL, the relation between syntax and semantics is given by interpretations that consist of an interpretation domain and an interpretation function. The interpretation function maps the entities and axioms of the syntactic statement to the interpretation domain. If an interpretation satisfies all the axioms of a given ontology, it is said to be a "model" of the ontology [58].

Automated reasoning: The application of a set of rules or processes in order to derive conclusions from a set of statements. For example, "consistency checking" can determine whether a set of facts are self-consistent or contain a contradiction. Automated reasoning can be used to realize a number of tasks in KR, for example computing implicit classification hierarchies from collections of definitions or query answering.

Soundness: Soundness is a property of an inference procedure or proof theory for a language that is in terms of the underlying semantics of the language. A Sound procedure is one which is guaranteed to only produce correct conclusions. See also Completeness.

Completeness: Completeness relates an inference procedure or proof theory for a language with the underlying semantics of the language. A Complete procedure is one which is guaranteed to find all valid inferences. See also Soundness.

Description Logics (DL): DLs are a family of KR Languages tailored for expressing knowledge about concepts and concept hierarchies. They are usually given a declarative semantics, which allows them to be seen as sub-languages of predicate logic. They are considered an important formalism, unifying and giving a logical basis to the well known traditions of frame-based systems, semantic networks and KL-ONE-like languages, object-oriented representations, semantic data models, and type systems. DL systems have been used for building a variety of applications including conceptual modelling, information integration, query mechanisms, view maintenance, software management systems, planning systems, configuration systems, and natural language understanding. In general, DL languages are “well-behaved”, with sound and complete procedures for inference. See also Soundness, Completeness.

Monotonic: In a monotonic logic, conclusions can not be falsified by the addition of new information. Thus, “additional axioms can lead to additional inferences but they cannot annul previous inferences” [84]. In a non-monotonic logic, in contrast, the addition of new hypotheses can cause previously derived conclusions to become false.

Rigour: Rigorous modelling exploits precise, consistent and explicit modelling (ontological) principles. For example, the part-of relationship in a concrete ontology can be defined as being transitive, and only relating physical independent entities. Expressed in a KR formalism, rigorous modelling can be computationally exploited. However, “rigorous” is not a synonym of “formal”, as “rigorous” relates to the content of an ontology, and “formal” relates to a means of computationally expressing such rigour through semantics.

Schema: A Schema, in general, is a concrete, explicit and agreed description of the way some data should be presented, in terms of the data types used, the relations between such types, *etc.* In different fields this term has different but related meanings, *e.g.* a database schema is different from an XML schema, but they are implementations of the same general notion. Ontologies are sometimes regarded as schemas.

URI: An URI, Uniform Resource Identifier, is a string with a concrete syntax that identifies a resource on the internet. A URL, Uniform Resource Locator, is a string, also with a concrete syntax, that gives the location of such resource. A URL is a type of URI.

Chapter 1

Introduction

Life sciences, and especially molecular biology, have generated vast amounts of data since the advent of technologies like DNA sequencing, requiring a whole new discipline, bioinformatics, for scientists to be able to computationally exploit the information derived from such data [110]. Such exploitation of information is performed by querying it, filtering it, and combining it with other information to obtain new results. For example, a scientist can use ortholog information and other sources of information (*e.g.* protein interactions, cellular location) to deduce the hypothetical function of a given protein.

The bottleneck for an efficient scientific activity in life sciences research has shifted from experimental data obtention, which is exponentially growing, to the exploitation of the information derived from the data, which is not growing at the same pace. This is due to two main reasons: the nature of biological information and the strategy that has been adopted for its exploitation. The biological information available in public resources is complex, heterogeneously represented, fine-grained and stored in high quantities. On the other hand, the exploitation of such information has been traditionally focused on human intervention, and implemented with tools like relational databases, text-based browsing and *ad hoc* creation of customised programs. The problem with such approach is that it results in a limited exploitation of the information. For example, it is difficult for a single scientist to relate different items of information in a principled manner (so other scientists can reuse such relations), and, most importantly, queries to the information yield incomplete results. Therefore plenty of information is not processed and adequately related, so as to allow its maximum exploitation by scientists.

One of the steps that has been taken to improve the information management in life sciences is to represent information using ontologies (bio-ontologies). However, many of the current bio-ontologies lack the necessary quality for fully functional information management. This thesis proposes the use of Ontology Design Patterns (ODPs) to develop bio-ontologies of higher quality.

The chapter is organised as follows. Section 1.1 provides a brief introduction to the research problem: the lack of quality in current bio-ontologies. A solution to such lack of quality, based in the use of ODPs, is proposed in Section 1.2. The research hypothesis and the research questions are explained in Section 1.3. The contributions of this work to the community are reviewed in Section 1.4. Finally, the structure of the thesis is outlined in Section 1.5.

1.1 Bio-ontologies

One way of improving the exploitation of information by life scientists is using the aid of computers. Computers, contrary to human users, are able to cope with the complexity and large quantity of biological information, provided that the information is codified in a way that computers can efficiently manage. Thus, the information needs to be "translated" to a form that allows the computer to manage its meaning. Such translation is provided by precise semantics, a mathematical formalism that allows to state information unambiguously via axioms.

Knowledge Representation (KR) languages are employed by human users to exploit precise semantics and create ontologies that store the information, thus to state the information computationally so that the computer manages it for the human users. An ontology is a computational representation, with precise semantics, of the concepts extracted from a concrete domain of knowledge. Such concepts are codified with identifiers and connected with relationships (axioms), creating a structure. The use of precise semantics allows algorithms to process the statements made in an ontology and retrieve results in a process called "automated reasoning".

Ontologies that represent information about biology, *i.e.* bio-ontologies, are widely used in current bioinformatics [44]. One of the main benefits of using bio-ontologies is to exploit automated reasoning, *e.g.* to generate new hypotheses about biological data, to query the information, to classify a new item against the bio-ontology or to check the consistency of the information held in public resources [78]. A bio-ontology is a consensus representation of a domain of knowledge, and, as such, it can be used to integrate different resources (as different items of different resources point to the same key concept in the ontology) or share information in an agreed computational language.

Combining automated reasoning with the integration of resources scattered over the internet, that is, combining semantics with the World Wide Web (WWW), a powerful environment for distributed information management is obtained. Such a combination of semantics with WWW technology has been proposed for enhancing the current WWW, in the so called Semantic Web. The Semantic Web is an ideal next generation WWW, where information is processed by computers, not only by humans. The Semantic Web is based on codifying the information using KR techniques as well as traditional WWW techniques. One of those KR techniques is to use a KR language to create ontologies in a WWW context; the Web Ontology Language, OWL, is one of the most used KR languages for creating such ontologies. Bio-ontologies are part of the so called Life Sciences Semantic Web (LSSW), the application of Semantic Web technology to the problem of information management in life sciences [49].

The majority of current bio-ontologies differ from the ideal of a fully functional LSSW due to mainly three aspects: rigour, axiomatic richness, and usage.

A rigorous ontology follows clear and consistent principles for defining how to represent information in the ontology. There are KR formalisms that offer the possibility of *practically* exploiting such rigour, *e.g.* languages that offer the possibility of sound and complete automated reasoning[1], like OWL.

The axiomisation of an ontology indicates what has been stated in the ontology. An ontology with a rich axiomisation represents complex knowledge with a high resolution, exploiting complex axioms. An ontology with a lean axiomisation represents knowledge in a simpler manner, exploiting simpler axioms. The level of axiomisation that can be achieved when building an ontology depends on the expressivity of the KR language used: the more expressive the language, the more complex axioms can be codified in the ontology.

Rigour and axiomatic richness are independent aspects of bio-ontologies. There are bio-ontologies implemented with rigour and with lean axiomisation and, on the other hand, bio-ontologies with rich axiomisation implemented without rigour. The OWL version of the Gene Ontology (GO) [40] is an example of the former, as it is implemented exploiting a rigorous formalism (OWL), but a limited fragment of the

[1]Even though sound and complete automated reasoning can be guaranteed in theory, performing automated reasoning can be difficult in some ontologies, depending on the complexity of the ontology and the available computational resources [27].

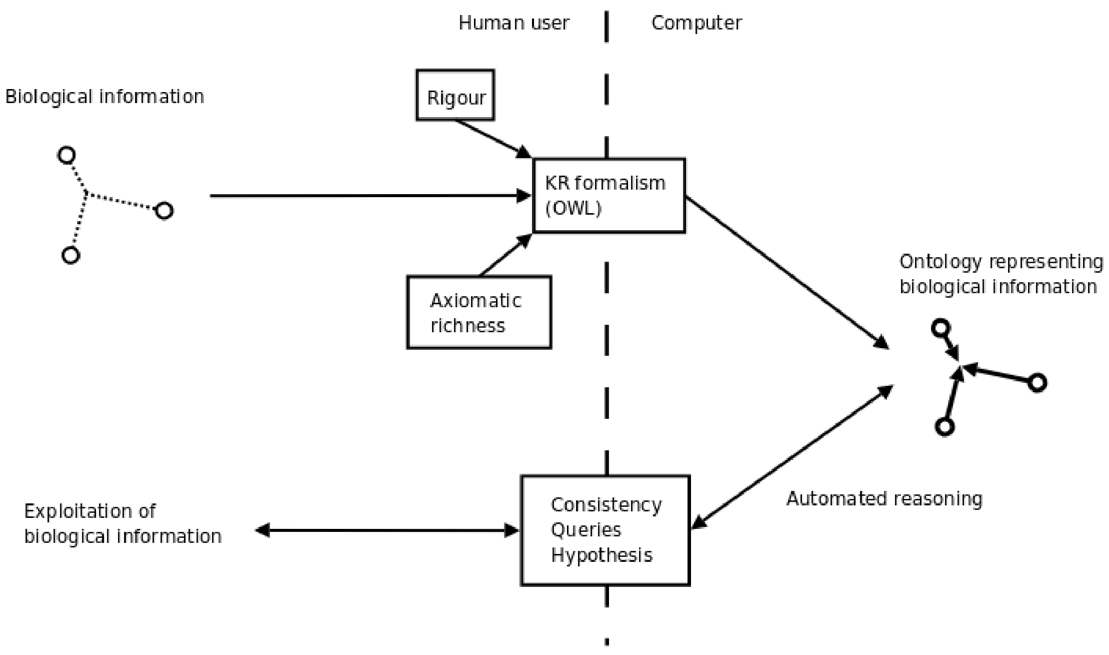


Figure 1.1: Rigour and axiomatic richness are used to create bio-ontologies for computational exploitation of the information. Rigour and axiomatic richness are most efficiently implemented through a KR formalism like OWL. That is, OWL can be used to create rigorous and axiomatically rich bio-ontologies. Those bio-ontologies can be computationally exploited, *via* automated reasoning, to perform various tasks like checking the consistency of the information, rich querying of the information, or to generate new hypotheses about the information.

expressivity of OWL is used in axioms. On the other hand, the OBO[2] version of the Sequence Ontology (SO) [63] is an example of the latter, as it is axiomatically rich (*e.g.* symmetric properties and intersections can be found in SO) but such axioms are implemented in a non-rigorous manner, without exploiting the precise semantics of a formalism like OWL. However, when axiomatic richness is implemented in a rigorous formalism with sound and complete automated reasoning, a powerful knowledge representation is achieved, as the combination allows computational tools to process the axioms, *via* automated reasoning, instead of only being (inefficiently) processed by human users (Figure 1.1).

To what extent rigour and axiomatic richness are needed by the ontologists' community depends on a third aspect of ontologies, the function that such ontologies will perform. Such function ranges from acting as a controlled vocabulary (a list of terms with a loose structure) to capturing the knowledge of a concrete domain in a domain

[2]OBO (Open Biomedical Ontologies) is a KR language used exclusively for bio-ontologies, described in Chapter 2.

ontology of the highest possible resolution. In a controlled vocabulary axiomatic richness and rigour are not important, but in a domain ontology they are paramount for efficient automated reasoning and hence maintenance, querying, and hypothesis generation. One of the problems of the majority of current bio-ontologies is that both extremes are neither clear for users nor developers. As a result, most bio-ontologies are used as domain ontologies and controlled vocabularies at the same time but they are developed only as controlled vocabularies, without using complex axioms or following principles of rigorous modelling.

One of the main reasons for such lack of rigour and axiomatic richness is that rigorous and axiomatically rich bio-ontologies are difficult to build. The complex axioms needed to effectively represent complex knowledge are unlikely to be “discovered” by biologists in the whole available expressivity of a KR formalism like OWL. Also, such axioms are difficult to understand and sometimes non-intuitive, especially with regards to the conceptual implications of using an axiom or the consequences of applying automated reasoning after adding the axiom. Rigour is also difficult to comprehend, as the formal definitions are abstract and obscure.

Besides the intrinsic difficulty of building ontologies, the application of KR technics to biological knowledge is relatively new, and bio-ontology engineering has been demonstrated to be more difficult for biologists than expected [92, 129]. As a result, however successful and widely used, many bio-ontologies have not been implemented exploiting the best that KR languages like OWL can offer in terms of axiomatic richness and rigour. There are some exceptions, like OBI[3] (Ontology of Biomedical Investigations), yOWL [118], and the Phosphatases ontology [125], but the majority of bio-ontologies lack axiomatic richness and rigour. Therefore, many bio-ontologies represent biological knowledge in a limited and lean manner, without exploiting precise semantics, hence hampering the prospective exploitation of such knowledge by computers and ultimately by humans.

1.2 Ontology Design Patterns (ODPs) for bio-ontologies

This section describes an ontology engineering technique, the use of Ontology Design Patterns (ODPs), that facilitates the process of building axiomatically rich and rigorous bio-ontologies.

In software engineering, the problem of how to build powerful artefacts minimising

[3] `http://purl.obofoundry.org/obo/obi`

the engineering effort and maximising the exploitation of the languages' expressivity has been widely explored. As software engineering is an older and more widespread discipline than ontology engineering, some of the techniques employed to overcome the problem have been thoroughly tested and demonstrated to work in the development process of widely used software.

One of the most widespread techniques is the use of design patterns in the development of software [33]. A design pattern is a solution to a common modelling problem that appears when designing different systems. The design pattern for such a problem is an efficient solution, as it has been already widely used and tested. It is also thoroughly documented and uniquely identified. A software engineer can use a collection of design patterns to design an efficient system in a principled and hence reusable manner and with reduced effort.

The same technique can be applied in ontology engineering, thus, design patterns can be defined for tackling common pitfalls and modelling problems that arise when building ontologies. Such design patterns are known as Ontology Design Patterns (ODPs). ODPs encapsulate the complex expressivity and precise semantics of KR languages in self-contained efficient solutions, properly documented and thoroughly tested by other, more experienced, ontology developers. Therefore, ODPs are "off the shelf" solutions to problems faced when creating and maintaining ontologies. ODPs can be regarded as "modelling units", thus discrete fragments of modelling, formed by a concrete set of axioms.

For example, it could be that an ontologist wants to model the fact that water has a standard boiling temperature under a certain pressure and using certain units. There is a simplistic ODP that shows how to model such information, the Nary DataType ODP[4], that consists of an OWL class (`standard_water_boiling_point`) that holds all the numeric values: temperature value (100), temperature unit (Celsius), pressure value (1), and pressure unit (Atm). This simplistic example shows the rationale behind the use of ODPs: it could be that an ontologist does not come up with the idea of naming a class to hold all the relationships (`temperature_value`, `temperature_unit`, `pressure_value`, `pressure_unit`), and the Nary DataType ODP acts as a guide towards such idea.

Therefore, if the expressivity of a KR language is regarded as a hypothetical multidimensional space (called "expressivity space" in this thesis), an ODP reduces the search for an optimal solution, which is a concrete point in such space that fulfils the

[4] http://www.gong.manchester.ac.uk/odp/html/Nary_DataType_Relationship.html

ontologist's modelling requirements. As the modelling requirements that the ODP solves are clearly explained in its documentation, the ODP encapsulates a point on the expressivity space with an identifier and the requirements, making it easier to find. Using ODPs in the development of the ontology, the actual modelling becomes explicit. Also, the modelling can be shared in the form of ODPs, as ready to apply modelling units. Finally, the modelling is principled, making such modelling efficient to reuse.

As mentioned in Section 1.1, bio-ontologies lack the axiomatic richness and rigour that KR languages such as OWL can offer. As ODPs guide the ontologists in exploiting the capabilities of KR languages to solve concrete problems, they can be used by biologists to create axiomatically rich and rigorous bio-ontologies [10]. By using ODPs the biologists should be able to develop rigorous, robust and axiomatically rich bio-ontologies, hence being able to efficiently exploit precise semantics, and do so more efficiently and faster.

1.3 Research hypothesis and research questions

The hypothesis of this thesis is that by using ODPs, bio-ontologies of higher quality can be created with reduced development effort. Higher quality bio-ontologies, compared to the majority of current bio-ontologies, are axiomatically richer and more rigorous. Axiomatically richer bio-ontologies allow for more diverse computations through automated reasoning (inference): more descriptive queries, more thorough analysis of biological information, consistency checking of the represented knowledge, hypothesis generation, and deeper integration of the information. More rigorous bio-ontologies can be more consistently and therefore more efficiently maintained. By using ODPs, all that richness and robustness can be obtained with reduced effort and the bio-ontologists' effort can be focused in modelling even more biological knowledge that needs to be captured in bio-ontologies.

The validation of the hypothesis consists of evaluating three areas: ODP quality, ontology engineering and ontology quality. ODP quality analyses ODPs as standalone units, focusing on the features and drawbacks of an ODP independently of its concrete application on an ontology. An ODP with a high quality is beneficial because it encapsulates a highly expressive or especially complex modelling in a clear way and with the least possible drawbacks. Ontology engineering analyses how each ODP affects the ontology development process, in terms of making it more efficient and more principled. Ontology quality analyses the quality improvement of bio-ontologies as a consequence of applying ODPs, in terms of axiomatic richness, rigour, and use of the ontology.

The hypothesis and the evaluation strategy result in the following research questions, answered in the remaining chapters of this thesis:

What are ODPs? (Chapter 2).

How can we obtain ODPs? (Chapter 3).

How can we apply ODPs? (Chapter 4).

How can we assess ODP quality? (Chapter 5, Chapter 6).

How can we assess the impact of ODPs in bio-ontology engineering? (Chapter 5, Chapter 6).

How can we assess the change of quality of bio-ontologies as a result of applying ODPs? (Chapter 5).

How does the use of ODPs change the quality of concrete bio-ontologies? (Chapter 5, Chapter 6).

1.4 Contributions

The work performed for this research has resulted in the following contributions:

Explanation of the concept of ODPs. The exploitation of ontology engineering techniques such as the use of ODPs is relatively new in bio-ontology development: the idea of ODPs and their advantages has been thoroughly explained in this thesis, as well as in publications [10, 30, 108] and a tutorial[5].

Public catalogue of ODPs. An online catalogue of ODPs[6] has been created in order to collect, classify and consistently describe ODPs [10]. The bio-ontologists can use the catalogue to explore and retrieve ODPs.

Ontology PreProcessor Language (OPPL). OPPL[7] is a scripting language for automatically changing the axioms of OWL ontologies [30, 29, 10]. OPPL can be used to codify and apply ODPs, by representing ODPs in OPPL scripts.

[5]`http://www.co-ode.org/resources/tutorials/bio/`

[6]`http://odps.sf.net/`

[7]`http://oppl.sf.net`

Evaluation framework for ontology quality. An evaluation framework has been developed to assess the relative ontology quality of bio-ontologies in comparison to each other, based on the ISO 9126 software quality standard [32].

Improved ontological artefacts. Different bio-ontologies have been improved during this work (GO, Cell Type Ontology –CL– [14], and Cell Cycle Ontology[8] –CCO–) and several OPPL scripts, containing those improvements, have been generated.

1.5 Thesis outline

Chapter 2 (ODPs for bio-ontologies) provides the background of the work. The chapter describes the current state of bioinformatics and the need for precise semantics and hence bio-ontologies. The notion of ontologies and the different KR languages that can be used to implement them are also described, focusing on OWL. The quality problems of current bio-ontologies are reviewed, presenting the use of ODPs as a solution. Finally, a definition for ODPs and their advantages for bio-ontology engineering are provided.

Chapter 3 (A Catalogue of ODPs) provides a description of the online catalogue of ODPs. The catalogue provides a centralised resource for efficiently exploring and retrieving ODPs, as all the ODPs are consistently described. The chapter gives an overview of the design and implementation of the catalogue.

Chapter 4 (Ontology PreProcessor Language) describes OPPL, a scripting language for executing scripts that change the axiomisation of OWL ontologies, and therefore able to encapsulate ODPs in OPPL scripts and apply them in OWL ontologies.

Chapter 5 (Evaluation framework) describes the evaluation framework proposed for testing the results, consisting of ODP quality, ontology engineering, and ontology quality.

Chapter 6 (Evaluation results) describes the use cases, thus the application of ODPs in bio-ontologies, and the results of evaluating such use cases with the evaluation

[8] `http://www.cellcycleontology.org/`

framework from Chapter 5. The use cases are the following: Upper Level Ontology ODP[9] in CCO, Sequence ODP[10] in CCO, Entity-Quality ODP[11] in GO, Selector ODP[12] in GO, and Normalisation ODP[13] in CL.

Chapter 7 (Conclusions) reviews the whole thesis in the light of the evaluation results, providing the conclusions, a review of outstanding issues and pointers for future research.

Appendix A (Public Catalogue of ODPs) is a copy of the online catalogue of ODPs.

Appendix B (Ontology quality values for CL and nCL) provides the detailed values of the ontology quality evaluation performed in Chapter 6, comparing CL and nCL.

[9] `http://www.gong.manchester.ac.uk/odp/html/Upper_Level_Ontology.html`
[10] `http://www.gong.manchester.ac.uk/odp/html/Sequence.html`
[11] `http://www.gong.manchester.ac.uk/odp/html/Entity_Quality.html`
[12] `http://www.gong.manchester.ac.uk/odp/html/Selector.html`
[13] `http://www.gong.manchester.ac.uk/odp/html/Normalisation.html`

Chapter 2

ODPs for bio-ontologies

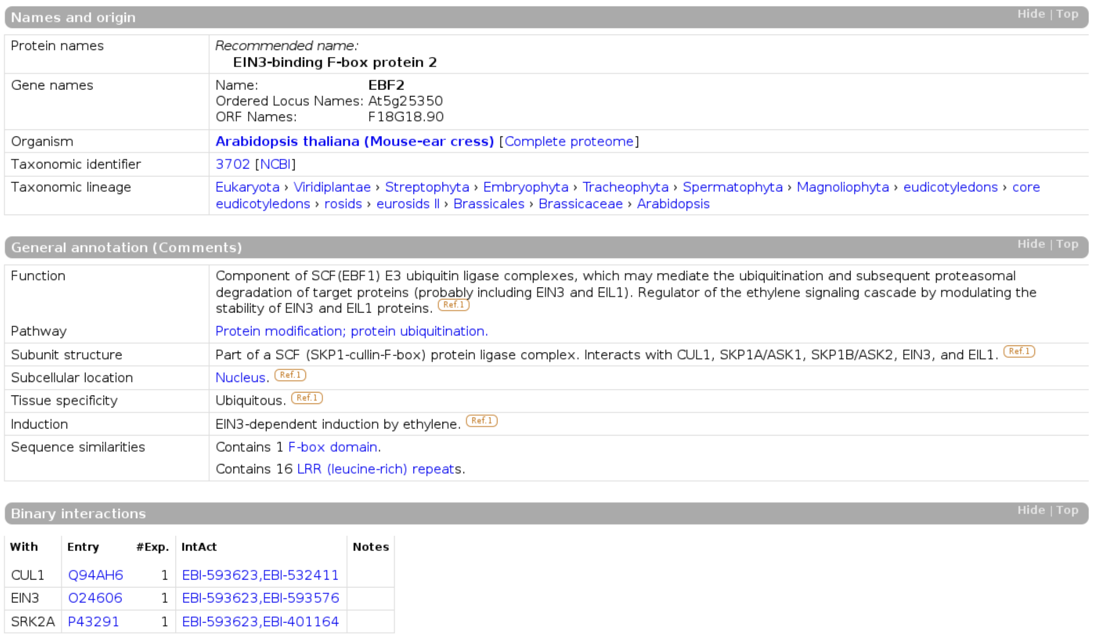
This chapter provides an answer to the research question *What are ODPs?*

The chapter starts by describing the problems of current bioinformatics, focusing on the need for precise semantics and the full implementation of a Life Sciences Semantic Web, in Section 2.1. Ontologies are the most efficient and widespread tool for exploiting precise semantics, but different communities consider different artefacts to be ontologies, especially regarding expressivity (axiomatic richness) and rigour, as described in Section 2.2. Ontologies are implemented using KR languages, and the KR language chosen highly affects the axiomatic richness and rigour of the resulting ontology. Therefore, Section 2.3 describes different KR languages, focusing on OWL. Ontologies that have been used to represent life sciences information, bio-ontologies, are reviewed in Section 2.4. As part of that review the quality problems of current bio-ontologies are also described in Section 2.4, focusing on lack of rigour and lean axiomisation. ODPs can be used to better exploit KR languages, therefore improving the rigour and axiomisation of bio-ontologies: the idea of ODPs is described in detail in Section 2.6.

2.1 Bioinformatics and the Life Sciences Semantic Web (LSSW)

2.1.1 Current bioinformatics and the need for precise semantics

Molecular biology techniques have advanced fast since the 1980s, especially DNA sequencing, and new techniques have appeared, like microarrays. This has resulted in an exponential growth of data [92, 75]. Bioinformatics provides the techniques to


Names and origin
Hide | Top

Protein names	*Recommended name:* **EIN3-binding F-box protein 2**
Gene names	Name: **EBF2** Ordered Locus Names: At5g25350 ORF Names: F18G18.90
Organism	**Arabidopsis thaliana (Mouse-ear cress)** [Complete proteome]
Taxonomic identifier	3702 [NCBI]
Taxonomic lineage	Eukaryota › Viridiplantae › Streptophyta › Embryophyta › Tracheophyta › Spermatophyta › Magnoliophyta › eudicotyledons › core eudicotyledons › rosids › eurosids II › Brassicales › Brassicaceae › Arabidopsis

General annotation (Comments)
Hide | Top

Function	Component of SCF(EBF1) E3 ubiquitin ligase complexes, which may mediate the ubiquitination and subsequent proteasomal degradation of target proteins (probably including EIN3 and EIL1). Regulator of the ethylene signaling cascade by modulating the stability of EIN3 and EIL1 proteins. Ref.1
Pathway	Protein modification; protein ubiquitination.
Subunit structure	Part of a SCF (SKP1-cullin-F-box) protein ligase complex. Interacts with CUL1, SKP1A/ASK1, SKP1B/ASK2, EIN3, and EIL1. Ref.1
Subcellular location	Nucleus. Ref.1
Tissue specificity	Ubiquitous. Ref.1
Induction	EIN3-dependent induction by ethylene. Ref.1
Sequence similarities	Contains 1 F-box domain. Contains 16 LRR (leucine-rich) repeats.

Binary interactions
Hide | Top

With	Entry	#Exp.	IntAct	Notes
CUL1	Q94AH6	1	EBI-593623,EBI-532411	
EIN3	O24606	1	EBI-593623,EBI-593576	
SRK2A	P43291	1	EBI-593623,EBI-401164	

Figure 2.1: Fragment of the UniProt entry for the protein Q708Y0. In this fragment the organism in which the protein can be found, the function, subcellular location, and interactions with other proteins are shown.

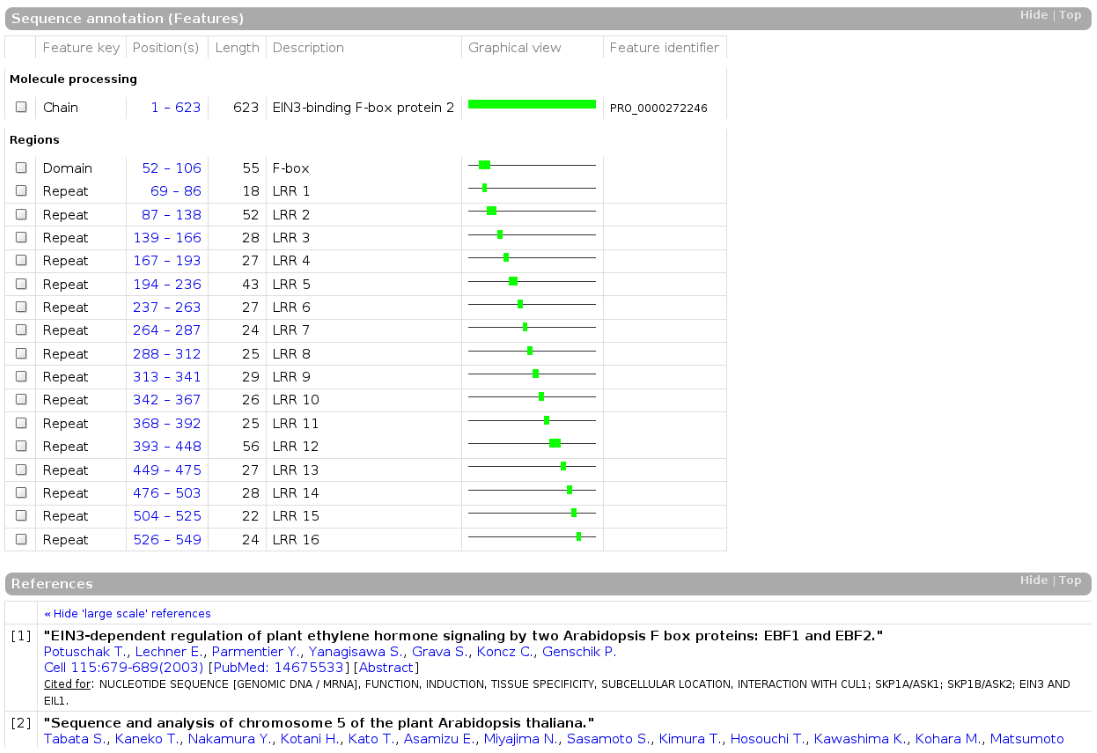
manage, store, process, and analyse such data [110].

The molecular biology data is diverse: gene expression data, protein structures, literature, sequences, *etc.* The data is processed algorithmically (*e.g.* sequence alignments) and also information is attached to it. The majority of such information is presented as "annotations" that capture statements *about* or *deduced from* the data. For example, in the case of the protein sequence identified in UniProt[1] as Q708Y0[2], the information on the annotations includes the organism in which the protein can be found, the function, subcellular location, molecular interactions with other proteins, sequence features, articles that back the statements of the annotations, references to other databases where the same protein appears, *etc.* The biggest part of that information is provided as either natural language statements or links to other databases and resources (Figures 2.1, 2.2, and 2.3).

One of the biggest assets of the biologists' community is the information that can be found on annotations, and therefore biology is considered to be a "knowledge based

[1]UniProt is a widely used public database that stores protein sequences and their associated annotations [116].

[2]`http://www.uniprot.org/uniprot/Q708Y0`



Sequence annotation (Features) Hide | Top

	Feature key	Position(s)	Length	Description	Graphical view	Feature identifier
Molecule processing						
☐	Chain	1 – 623	623	EIN3-binding F-box protein 2		PRO_0000272246
Regions						
☐	Domain	52 – 106	55	F-box		
☐	Repeat	69 – 86	18	LRR 1		
☐	Repeat	87 – 138	52	LRR 2		
☐	Repeat	139 – 166	28	LRR 3		
☐	Repeat	167 – 193	27	LRR 4		
☐	Repeat	194 – 236	43	LRR 5		
☐	Repeat	237 – 263	27	LRR 6		
☐	Repeat	264 – 287	24	LRR 7		
☐	Repeat	288 – 312	25	LRR 8		
☐	Repeat	313 – 341	29	LRR 9		
☐	Repeat	342 – 367	26	LRR 10		
☐	Repeat	368 – 392	25	LRR 11		
☐	Repeat	393 – 448	56	LRR 12		
☐	Repeat	449 – 475	27	LRR 13		
☐	Repeat	476 – 503	28	LRR 14		
☐	Repeat	504 – 525	22	LRR 15		
☐	Repeat	526 – 549	24	LRR 16		

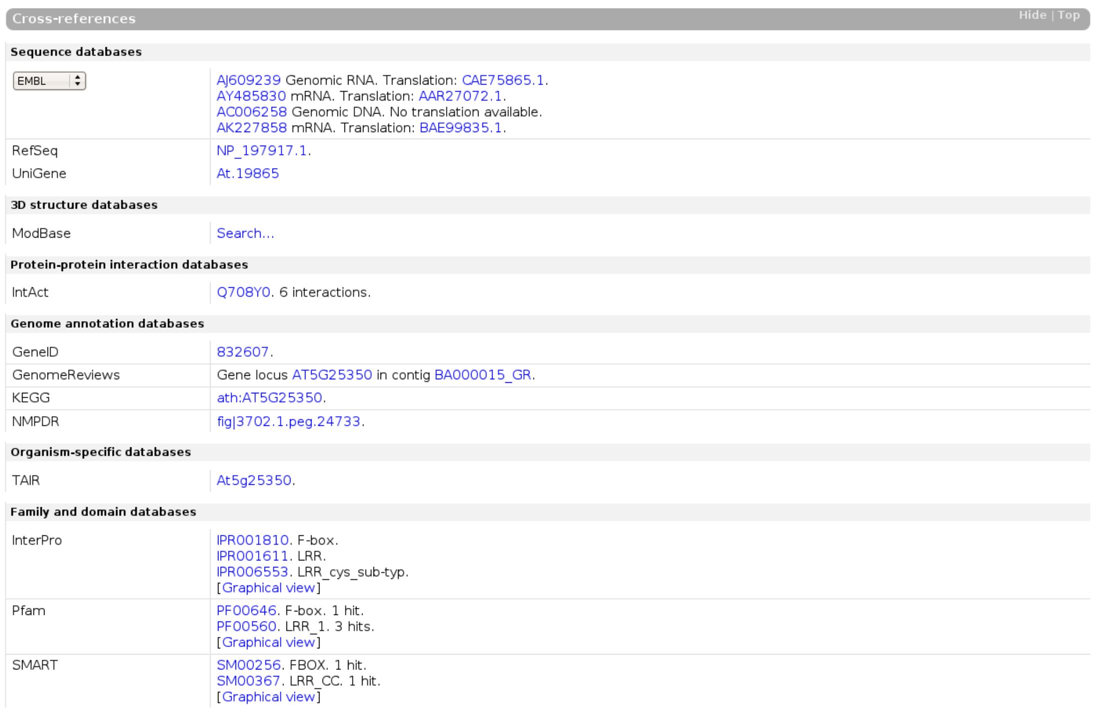
References Hide | Top

« Hide 'large scale' references

[1] **"EIN3-dependent regulation of plant ethylene hormone signaling by two Arabidopsis F box proteins: EBF1 and EBF2."**
Potuschak T., Lechner E., Parmentier Y., Yanagisawa S., Grava S., Koncz C., Genschik P.
Cell 115:679-689(2003) [PubMed: 14675533] [Abstract]
Cited for: NUCLEOTIDE SEQUENCE [GENOMIC DNA / MRNA], FUNCTION, INDUCTION, TISSUE SPECIFICITY, SUBCELLULAR LOCATION, INTERACTION WITH CUL1; SKP1A/ASK1; SKP1B/ASK2; EIN3 AND EIL1.

[2] **"Sequence and analysis of chromosome 5 of the plant Arabidopsis thaliana."**
Tabata S., Kaneko T., Nakamura Y., Kotani H., Kato T., Asamizu E., Miyajima N., Sasamoto S., Kimura T., Hosouchi T., Kawashima K., Kohara M., Matsumoto

Figure 2.2: Fragment of the UniProt entry for the protein `Q708Y0`. In this fragment the sequence features and articles that demonstrate the statements of the annotations are shown.



Cross-references Hide | Top

Sequence databases	
EMBL	AJ609239 Genomic RNA. Translation: CAE75865.1. AY485830 mRNA. Translation: AAR27072.1. AC006258 Genomic DNA. No translation available. AK227858 mRNA. Translation: BAE99835.1.
RefSeq	NP_197917.1.
UniGene	At.19865
3D structure databases	
ModBase	Search...
Protein-protein interaction databases	
IntAct	Q708Y0. 6 interactions.
Genome annotation databases	
GeneID	832607.
GenomeReviews	Gene locus AT5G25350 in contig BA000015_GR.
KEGG	ath:AT5G25350.
NMPDR	fig|3702.1.peg.24733.
Organism-specific databases	
TAIR	At5g25350.
Family and domain databases	
InterPro	IPR001810. F-box. IPR001611. LRR. IPR006553. LRR_cys_sub-typ. [Graphical view]
Pfam	PF00646. F-box. 1 hit. PF00560. LRR_1. 3 hits. [Graphical view]
SMART	SM00256. FBOX. 1 hit. SM00367. LRR_CC. 1 hit. [Graphical view]

Figure 2.3: Fragment of the UniProt entry for the protein Q708Y0. In this fragment the references to other databases where the same protein appears are shown.

discipline" [18]. This means that, unlike disciplines like physics, where abstract equations describe complex systems, in biology all the information needs to be captured and described, usually in the form of annotations that represent the community's knowledge about an entity or process. Considering that the flow of data (especially high throughput data) is growing in volume and exponentially faster, the bottleneck for efficient information management in bioinformatics has shifted from data to annotations [80].

The sheer quantity and complexity of such knowledge makes it impossible to be efficiently managed by a scientist using current techniques and data formats. For example, a scientist may want to do the following analysis given a concrete protein like Q708Y0: (1) get the orthologs of the protein for *A. thaliana*, (2) if any ortholog is located in the nucleus, get the proteins that interact with such orthologs *via* phosphorilation, (3) get the regulation processes in which those proteins participate, and compare them. Such analysis is usually done manually, thus a scientist collects the pertinent information from annotations, manipulates the output of each step to fit the input of the next step, and after completing all the steps obtains the result. Doing such analysis if, *e.g.* there are various proteins in the initial set, becomes too complex to be carried out by a scientist, and tracking all the information on each step laborious.

Another problem with the information in bioinformatics, apart from its complexity and quantity, is that it is generally provided in isolated resources with unique schemas and different formats [92, 44]. Therefore the integration of information is a difficult task, as *ad hoc* tools must be created for integrating concrete resources with their own schemas. This poses big problems for the biologists. For example, it is difficult to make queries about this disparate knowledge: a biologist must traverse different resources, collecting the necessary information, in a process that will most probably provide incomplete results (Figure 2.4).

A considerable part of the information is "buried" in literature, which makes the situation even worse, as an article needs to be read each time in order to extract the pertinent information. For example, as many as 30,000 articles have been published in recent years solely on the subject of cell apoptosis [68]. The problem of information being computationally "hidden" in resources like articles is so important that a whole subdiscipline of bioinformatics, text mining, aims at developing reliable techniques for automatically extracting information from the articles.

Computers, contrary to scientists, are able to do highly complex analysis involving lots of related but disparate information items, like the analysis described above that starts with the single protein `Q708Y0`, but starting with many related proteins. In order for computers to be able to perform such analysis, the knowledge needs to be properly codified in them, using precise semantics *via* KR techniques. Such "knowledge" is not the same knowledge that scientists hold in their minds, despite being denoted by the same term. However, it is useful and safe for scientists to *assume* that it is the same knowledge, as that allows scientists to delegate to the computer tasks that would be too complex. Precise semantics makes it possible to delegate the task of performing such analysis to computers, as the scientist has the guarantee that what he understands is the same as what the computer "understands".

The same task-delegation to the computer takes place, with, for example, a computer program that executes some arithmetic operation such as *1979 divided by 29*. The scientist has the guarantee that what he understands by *1979 divided by 29* is the same as what the computer understands by *1979 divided by 29*, and, more importantly, the result will be the same, the only difference being the time needed to perform the operation. This is so despite the fact that the knowledge of *1979 divided by 29* is not the same: in a computer it is a collection of bits in memory, and in the mind of the scientist it is a complex neurophysiological phenomenon with different implications for different scientists (*e.g.* the resulting number could mean a lot for someone born in 1979 whose age is 29). However, in the same way that different scientists unambiguously agree on the exact meaning of *1979 divided by 29*, both the computer and the scientist agree on the result through the use of a formalism, in this case, arithmetic, that allows a precise interpretation of what *1979 divided by 29* means. And, more importantly, different computers agree on the meaning of *1979 divided by 29*, which allows them to exchange and manipulate results without human intervention.

KR languages are an example of the same delegation principle, but using concepts, in the form of logic, instead of arithmetic [27]. For example scientists and computers can precisely agree on the meaning (semantics) of the subclass relationship, as the relationship between two sets where all the instances of one set belong also to the other set, but not the other way around[3]. Describing information using a KR language that has precise semantics, scientists can exploit a feature that computers have and humans lack: computational "brute force". For example, given thousands of instances and their attributes, a computer can work out all the subclass relationships, something difficult for a scientist. Therefore, precise semantics provide a means for the computational manipulation of representations of knowledge.

Ontologies are the most common tool for exploiting precise semantics and delegating inferences to the computer. Bio-ontologies (ontologies that describe biological knowledge) are useful in bioinformatics because there are many concepts in annotations that can (and should) be codified using the precise semantics of a KR language like OWL, and hence, ideally, be managed as easily as the operation *1979 divided by 29*.

2.1.2 Life Sciences Semantic Web

Bio-ontologies are part of the Life Sciences Semantic Web (LSSW) [49], an endeavour aiming at providing a semantic framework making it possible to efficiently integrate and manage biological information through the web. Ideally, in a LSSW context, a biologist should be able to process data and annotations automatically and reliably, in the same way that is already possible, for example, with sequences and sequence alignment algorithms using the tools that exploit such algorithms. This means that, in the example of Figure 2.4, a simple query and exploitation of transitivity would answer the scientist's complex question. Such actions could be implemented, for example,

[3]What the computer "understands" is the structure resulting from combining various sets, regardless of the names given to the instances or the sets: the names are only useful for humans.

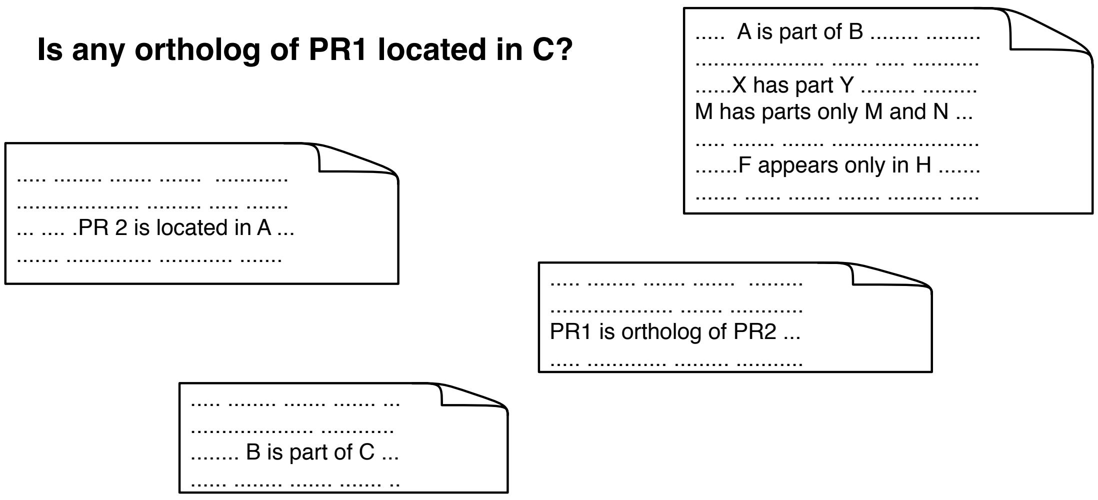

Is any ortholog of PR1 located in C?
..... A is part of B ....
......X has part Y ....
M has parts only M and N ...
.......F appears only in H ....
... . .PR 2 is located in A ...
PR1 is ortholog of PR2 ...
.... B is part of C ...

Figure 2.4: Information is disperse in bioinformatics resources. For example, if a biologist wants to know whether an ortholog of PR1 is located in C, he will have to go through the following path: he will realise that PR2 is an ortholog of PR1, then he will go to another resource and get the location of PR2 (A), then he will go to another resource that states that A is part of B, and finally, as there is another resource that states that B is part of C, he will answer his query with *yes*.

in an automatic digital assistant that would be able to perform analysis like the one described in Section 2.1.1, starting with the protein `Q708Y0` [13].

The LSSW is an application of the very technology that has been proposed to create the next generation of the WWW, the Semantic Web. The Semantic Web is one of the activities of the World Wide Web Consortium[4] (W3C), an organisation set up to recommend standards for an open and interoperable WWW. The Semantic Web vision[5] proposes a web made of computationally processable *statements*, instead of the current *syntactic* web made of *documents* [16]. In other words, documents like HTML pages are inappropriate containers for concepts, as it is necessary to process the document to retrieve a concept, and the Semantic Web proposes a web made of concepts instead of documents. This would result in a world wide network of interlinked concepts and data that could be exploited by human users and automated agents alike, exploiting the already described principle of delegating to the computer the task of performing complex inferences, *via* precise semantics.

Such a Semantic Web could be used for finding the exact information needed, in queries like *find the names of all the mammal species that are predators, live in Africa,*

[4]`http://www.w3.org/`
[5]`http://www.w3.org/2001/sw/`

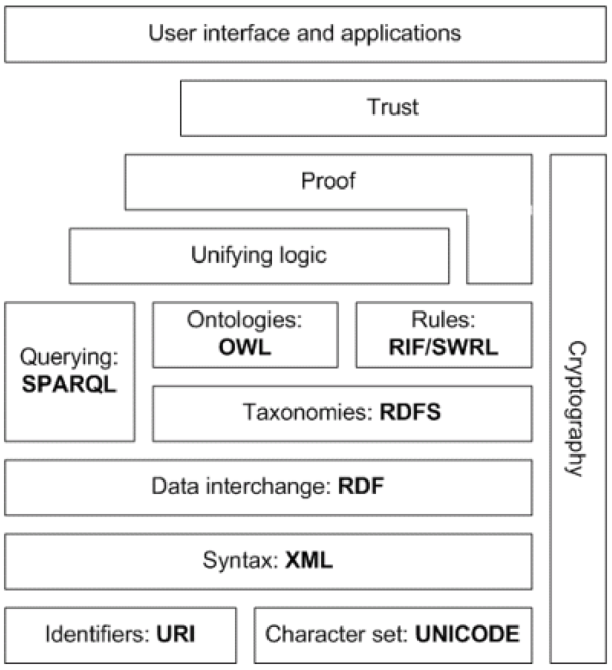


Figure 2.5: The Semantic Web stack. Each layer adds new functionalities but also exploits the functions of the layer bellow: for example, RDF exploits XML functionalities but adds a data model. XML, RDF, SPARQL, RDFS and OWL are explained in Section 2.3.

are not lions, and have been mentioned in the New York Times by any author who has been working there for more than 5 years. In the current syntactic web such a general query is impossible: a user must traverse through different resources and collect/compare the information. Even a simple query like *is Rome located in Italy?* is difficult to do automatically. The Semantic Web could also be used by an automatic agent to, for example, book a journey that fits certain criteria the user has previously entered: dates, itinerary of cities, maximum prices, hotels, events, *etc.* Using such an agent, the users save time by not performing tedious and meaningless tasks. The key to the Semantic Web is to publish data in a machine processable manner, for example, by representing data through ontologies. In this manner, the agent mentioned above would be able to exploit automated reasoning using different ontologies (*e.g.* a hotels ontology, an events ontology, *etc.*) and deliver the needed result automatically.

The Semantic Web is based in a collection of standards that extend the current web technologies, instead of replacing them, in the Semantic Web Stack, which is shown in Figure 2.5 (Image taken from the Wikipedia[6]).

[6]`http://en.wikipedia.org/wiki/Semantic_Web_Stack`

Even though Semantic Web technologies have not been completely adopted in life sciences, the use of such technologies is growing [49]. Biology is an ideal test-ground for Semantic Web technologies, as biological information presents a combination of attributes not found in other fields that make it a demanding use case [110, 124]. Biological information is highly complex, with many different types of information about the same entity, and it is stored in large volumes; it grows fast and it rapidly changes; it is distributed in different resources and hence represented with disparate schemas; it needs to be represented at the highest resolution possible. On the other hand, there is a long tradition of classification of entities in biology [110], dating back to Linnaeus, which can be seen as a precursor of today's bio-ontologies discipline. One of the latest incarnations of such tradition is the growing group of scientists that are ready to computationally codify biological information (an effort already committed to generating annotations), a situation not found in other disciplines. This is why the W3C created the W3C Semantic Web Health Care and Life Sciences Interest Group[7] (W3C HCLS), being aware of the potential of life sciences as a demanding test-case for the Semantic Web technologies.

2.2 Ontologies

The original meaning of the term "Ontology" refers to the philosophical discipline that identifies the kinds of things that actually exist. However the same term is used in computer science to refer to a computational representation of a domain of discourse by a community[8] [9]. The most cited definition for an ontology is "a formal, explicit specification of a shared conceptualisation" [111], but what is an ontology remains a controversial issue. As mentioned, ontologies have different aspects or axes that can be used to analyse them: rigour, axiomatic richness and usage are the most important ones. Different communities emphasise different points in each of those axes, therefore producing artefacts that are considered like ontologies by one community and not by other communities.

Rigour allows the scientists to use a KR formalism to share the meaning of concepts with the computers in a precise way, therefore making it possible for scientists to delegate to the computer analyses that are too big or complex, *via* inferences. The axis

[7]http://www.w3.org/2001/sw/hcls/

[8]Some authors make a difference between the term "Ontology" (uncountable, starting with upper case) for the philosophical discipline and "an ontology" (singular, starting with lower case) for the computational artefact.

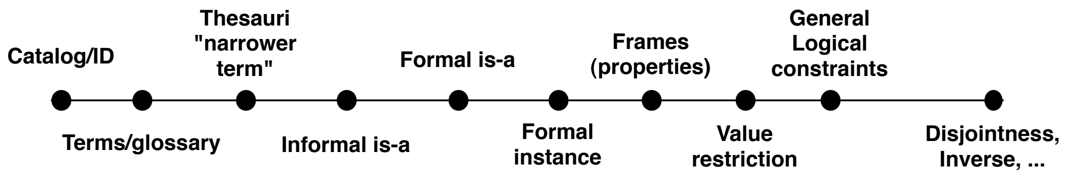


Figure 2.6: Ontology spectrum. Rigour increases from left to right: on the left extreme simple lists of terms can be found, and on the right extreme fully rigorous ontologies that exploit precise semantics.

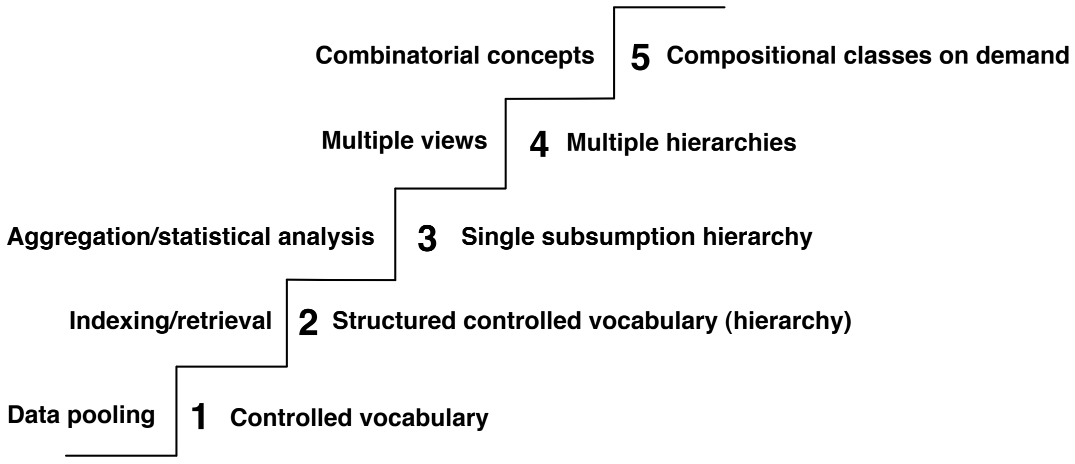

Combinatorial concepts
5 Compositional classes on demand
Multiple views
4 Multiple hierarchies
Aggregation/statistical analysis
3 Single subsumption hierarchy
Indexing/retrieval
2 Structured controlled vocabulary (hierarchy)
Data pooling
1 Controlled vocabulary

Figure 2.7: Feature escalator. The richer an ontology, the wider the range of uses: a controlled vocabulary is used for data pooling, whereas an ontology with classes that can be combined in rich ways (*e.g.* an OWL ontology) can be used for more tasks besides data pooling (*e.g.* generate multiple views of the data).

of rigour was fist represented by the “ontology spectrum” [123] (Figure 2.6). On one extreme we can find simple lists of terms and on the other fully rigorous ontologies.

The axes of axiomatic richness and usage can be seen in the “feature escalator”, first described in [73] (Figure 2.7). Axiomatic richness allows the scientists to delegate the management of a big portion of his domain of knowledge to the computer: the richer the ontology, the more the user can delegate to the computer. Therefore, the richer an ontology, the more functions it can perform at the same time, especially if such expressivity is backed by a rigorous formalism.

Considering the needs of each community, the requirements that an ontology should fulfil will stand in different points of those axes, resulting in a demand for an ontology with a concrete set of capabilities. The capabilities of an ontology largely depend on the KR language used to implement it.

2.3 Knowledge Representation (KR) languages

KR languages are diverse but this section will be focused only on the ones that are Semantic Web oriented (XML, RDF, RDFS, and OWL) or biology oriented (OBO format), due to the context of this thesis. XML is not strictly a KR language but it is included in this section because it forms the implementation base for RDF, RDFS and OWL in the Semantic Web[9] (Figure 2.5). KR languages differ mainly in the areas of semantics, syntax, expressivity and automated reasoning [10]:

Semantics: What well formed statements mean, thus the set of concrete situations that are consistent with a sentence. Precise semantics allows unambiguous interpretation of the sentences, by the computer and by the human users.

Syntax: The set of rules that are used for creating a well formed statement. Different statements can express the same situation when they are semantically interpreted: syntax is completely independent of semantics.

Expressivity: The ability of the language to distinguish different situations. For example, a language with simple cardinality restrictions, like the first version of OWL, is less expressive than a language with Qualified Cardinality Restrictions (QCRs), like OWL 2 (Figure 2.8). A simple cardinality restriction would be, for example, the following axiom: `HasAppendix max 5 owl:Thing`. Having `owl:Thing` as filler means that any class can be the filler. However, a QCR would read as follows: `HasAppendix max 5 finger`. The QCR allows us to distinguish between an entity that has at most 5 fingers as appendices (a hand: `HasAppendix max 5 finger`) and an entity that has, for example, at most 5 tentacles as appendices (a mutant octopus: `HasAppendix max 5 tentacle`). However, with the simple cardinality restriction, both entities have at most 5 appendices that are fingers, tentacles or whatever (`owl:Thing`). Expressivity is related to both syntax and semantics. For example, if a language's syntax allows the expression of minimum cardinality and negation, maximum cardinality can be expressed if the semantics of the language are such that there is an equivalent maximum cardinality equivalent to the negated minimum cardinality (*e.g.* `not HasAppendix min 4` is equivalent to `HasAppendix max 3`). There is

[9]RDF, RDFS and OWL are modelling languages and they do not depend on XML (they can be serialised –written in files– using any other syntax), but their most used implementation and the implementation for the Semantic Web heavily depends on XML.

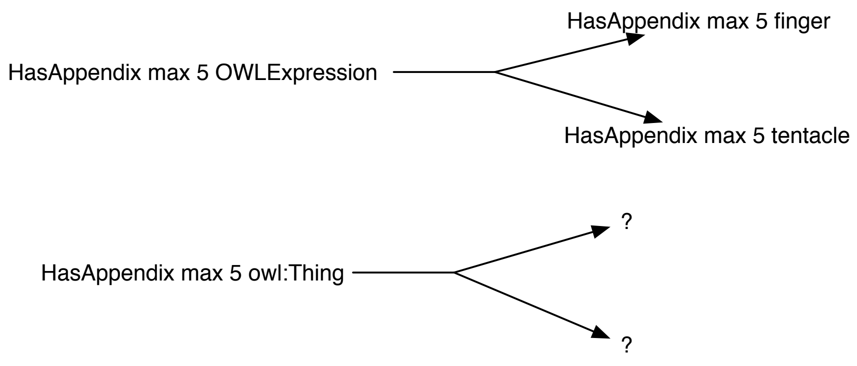

HasAppendix max 5 OWLExpression
HasAppendix max 5 finger
HasAppendix max 5 tentacle
HasAppendix max 5 owl:Thing
?
?

Figure 2.8: Different levels of expressivity in OWL. A QCR (top) makes further distinctions compared to a simple cardinality restriction (bottom).

an intuitive albeit not strictly linear expressivity gradient that goes from XML (least expressive), to RDF, to RDFS, and finally to OWL (most expressive) [9]. Expressivity is closely related to computational tractability, which measures the amount of resources needed for obtaining an answer: the more expressive a language, the less tractable [27]. Expressivity is also related to decidability. There could be a question formulated in a KR language, depending on its expressivity, for which there is no algorithm that provides a correct answer, and hence the question is said to be undecidable.

Automated reasoning: Answering some semantic based query, such as determining if one statement follows from another or if there is a consistent model according to what we have stated. Automated reasoning makes implicit knowledge explicit, not new knowledge. Thus, axioms that are entailed by the stated axioms are shown to the ontologist.

Depending on different factors like competency of ontology curators, resolution and size of domain knowledge, knowledge retrieval requirements, *etc.*, a given KR language will be more appropriate than others in one or more of the mentioned four areas. OWL is one of the most suitable KR languages for biological knowledge representation [108], and its use is rapidly expanding within and outside bioinformatics [50], so it is the language of choice for implementing the ODPs presented in the online catalogue of ODPs. Therefore it will be explained in more detail than the other KR languages in this section (XML, RDF, RDFS and OBO). A detailed explanation of OWL is also justified because the aim of this research is to make an expressive KR language like OWL more usable for bio-ontologists, so an overview of such expressivity must be provided in order to judge whether the proposed solution (usage of ODPs) fulfils that aim.

2.3.1 Extensible Markup Language (XML)

XML[10] is an official W3C recommendation[11] for defining markup languages. Thus, using XML one can define a concrete format to create structured documents that conform to a given set of rules, using XML Schema[12]. XML does not impose any semantics to the defined documents, but it imposes a concrete structure (a tree of document elements), so it is widely used for representing semi-structured information.

In bioinformatics, many databases provide entries in XML, and languages like SBML[13] (Systems Biology Markup Language) have been developed using XML. XML provides the syntactic base for RDF, RDFS, and OWL, which can be, and usually are, codified as valid XML documents. Using XML as the base makes the implementation of tooling for such languages more efficient than if they were codified in different formats. RDF, RDFS and OWL exploit XML functionalities and they add their own semantics, RDF on top of XML and RDFS and OWL on top of RDF (Figure 2.5).

2.3.2 Resource Description Framework (RDF)

RDF[14] was conceived to represent metadata in the Semantic Web, and it is used to describe resources and how they relate to each other. The model behind RDF is based in the subject-predicate-object triple pattern (Figure 2.9): a subject relates to an object *via* a predicate. The subject is described by the predicate (the property) and the object (the value that such property takes). For example in RDF we can make statements like `this thesis is authored by Mikel Egaña`, where `this thesis` is the subject, `is authored by` the predicate and `Mikel Egaña` the object: `is authored by Mikel Egaña` describes the entity `this thesis`.

A “graph” is obtained by combining these triples (Figure 2.10). RDF uses URIs[15]

[10]`http://www.w3.org/XML/`

[11]`http://www.w3.org/TR/2004/REC-xml11-20040204/`

[12]`http://www.w3.org/XML/Schema`

[13]`http://sbml.org`

[14]`http://www.w3.org/RDF/`

[15]URIs, being part of the base of the internet, have been reused for identifying entities in RDF, RDFS and OWL in order for such languages to exploit capabilities available in the Semantic Web Stack. The

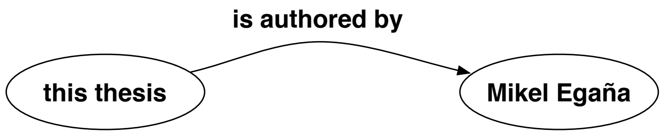


Figure 2.9: RDF triple. A subject (`this thesis`) is linked to an object (`Mikel Egaña`) *via* a predicate (`is authored by`).

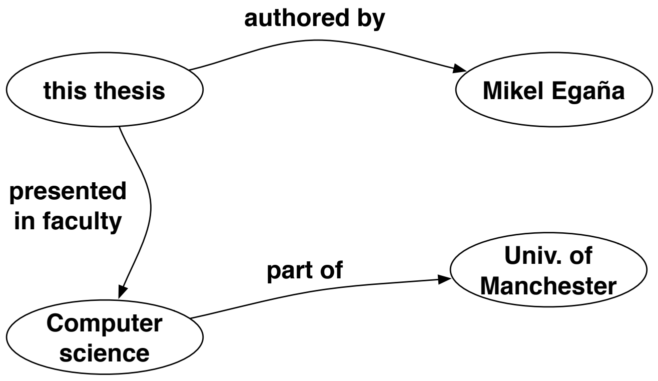


Figure 2.10: An RDF graph is obtained by combining RDF triples. The same entity can be the subject or the object of different triples, *e.g.* `this thesis` is the subject of two triples.

to identify entities (subjects, predicates or objects), so graphs can be combined over the internet. RDF is used in many resources, and there is a language called SPARQL[16] that offers a powerful, simple and flexible way of querying RDF graphs.

2.3.3 Resource Description Framework Schema (RDFS)

RDF Schema[17] provides constructs for grouping RDF resources in classes. A resource belongs to a given class (`rdfs:Class`) *via* the `rdf:type` construct and classes can be subclasses of other classes in a hierarchy, using the `rdfs:subClassOf` construct. RDFS also provides constructs for creating property hierarchies (`rdf:Property` and `rdfs:subPropertyOf`), and other constructs. RDFS is the first step towards an ontology language [9], as it allows the expression of general attributes of the sets (classes) of entities rather than only working with concrete entities in RDF triples. RDFS uses URIs to identify entities.

"fragment" of the complete URI is usually used to render OWL entity names in tools like Protégé: *e.g.* from the entity with the URI `http://www.gong.manchester.ac.uk/go.owl#GO_0000001`, the fragment `GO_0000001` is used to render the class name.

[16]`http://www.w3.org/TR/rdf-sparql-query/`

[17]`http://www.w3.org/TR/rdf-schema/`

2.3.4 Web Ontology Language (OWL)

OWL is a W3C official recommendation and it is being developed by the W3C OWL Working Group[18], aiming at providing a powerful and expressive KR language for publishing ontologies on the WWW. OWL ontologies are designed also to be part of software. For example, the OWL API[19] (Application Programming Interface) allows the building of applications that make a heavy use of ontologies as data models, and it offers the possibility of directly using reasoners such as Pellet [100] and FaCT++ [115]. Such API has been used on the creation of ontology editors like Protégé 4[20], an OWL 2 focused successor of Protégé 3[21]. There are other OWL ontology editors like Swoop[22] or TopBraid Composer[23].

2.3.4.1 OWL syntax and semantics

As mentioned, KR languages have four components: syntax, semantics, expressivity and automated reasoning. This section provides a brief overview of the syntax and semantics of OWL. The full expressivity, automated reasoning and other technical details of OWL are out of the scope of this thesis; for more information the reader should refer to the official W3C website about OWL[24] or a tutorial like the one provided by the CO-ODE group[25]. The overview that follows is aimed at providing a broad idea of OWL's design and capabilities; more about the OWL's expressivity can be appreciated by exploring the ODPs in the online catalogue of ODPs. In order for the reader to be able to understand some OWL semantics examples, its syntax is first reviewed.

OWL can be expressed in a range of syntaxes, divided in two groups: computer-readable syntaxes, designed for efficient parsing of OWL files by computers, and human-readable syntaxes, designed for maximum readability by human users. The most common computer-readable syntax is RDF/XML[26], although OWL/XML[27] is also used. The most common human-readable syntax is the Manchester OWL Syntax

[18]http://www.w3.org/2007/OWL/wiki/OWL_Working_Group
[19]http://owlapi.sourceforge.net/
[20]http://www.co-ode.org/downloads/protege-x/
[21]http://protege.stanford.edu/
[22]http://code.google.com/p/swoop/
[23]http://www.topbraidcomposer.com/
[24]http://www.w3.org/2004/OWL/
[25]http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial-p4.0.pdf
[26]http://www.w3.org/TR/rdf-syntax-grammar/
[27]http://www.w3.org/TR/owl-xmlsyntax/

```
<owl:Class rdf:about="#nucleus">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#part_of"/>
      <owl:someValuesFrom rdf:resource="#cell"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Figure 2.11: RDF/XML representation of the OWL class `nucleus`. The class has the restriction `part_of some cell` as a superclass.

```
nucleus subClassOf part_of some cell
```

Figure 2.12: MOS representation of the OWL class `nucleus`. This representation is equivalent in its semantics to the one of Figure 2.11.

(MOS) [57], which is used in Protégé 4. For example, the representation of an hypothetic OWL class called `nucleus`, with a restriction along the `part_of` property to the class `cell` (a nucleus must have at least one part of relationship to a cell), is shown in RDF/XML syntax in Figure 2.11 and in MOS in Figure 2.12.

OWL has precise semantics, provided by Description Logics (DL), based in entities (identified by their URI) and axioms. Axioms are used to make statements about the domain of knowledge, relating entities. Entities can be of three kinds (Figure 2.13): individuals, classes or properties. Individuals are the objects from the domain of knowledge, classes are sets of individuals, and properties link pairs of individuals in relationships. The TBox (Terminological Box), formed by the classes, provides the vocabulary that must be used to describe the knowledge domain. The ABox (Assertional Box), formed by the individuals, asserts facts about the knowledge domain using such vocabulary.

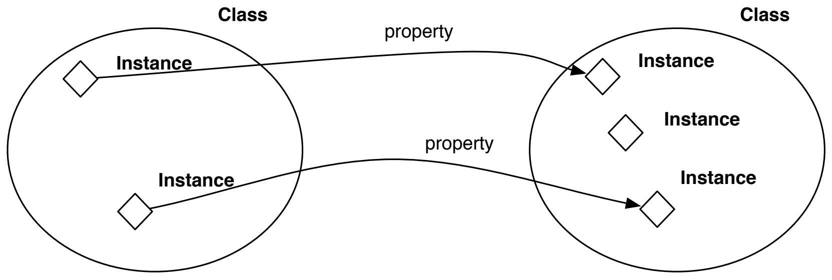


Figure 2.13: OWL entities and their role in the OWL model. A Class (circle) is a set of individuals (diamonds). Properties link individuals in binary relations (arrows).

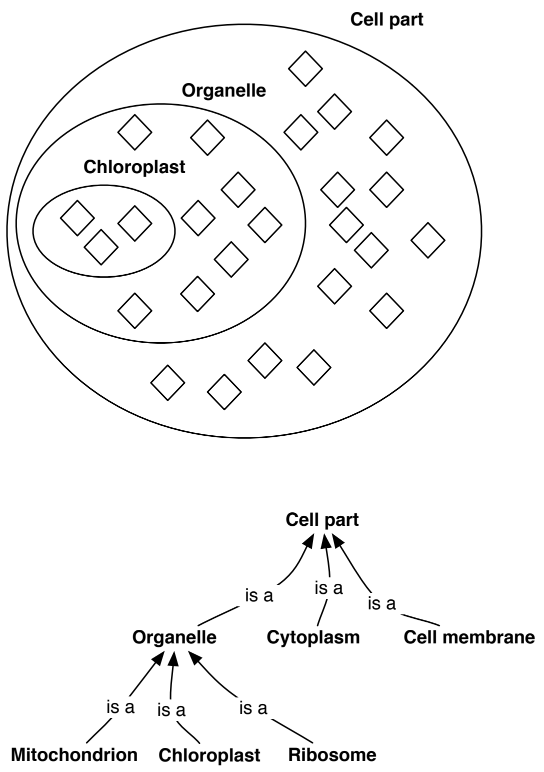


Figure 2.14: OWL classes and subclasses. The set of individuals `Chloroplast` is a subset of the set `Organelle`, which is a subset of `Cell part` (top). There are other organelles that are not chloroplasts (*e.g.* ribosomes), therefore `Chloroplast` is a subclass of `Organelle`, and as there are other cell parts that are not organelles (*e.g.* cytoplasm), `Organelle` is a subclass of `Cell part`. An is-a hierarchy is built by combining different subsumption relationships (bottom).

Individuals are grouped in categories, forming classes: a class is a set of individuals that share some attributes. A class is a subclass of another class if, and only if, all the individuals of the subclass are also individuals of the superclass, but not all the individuals of the superclass are individuals of the subclass. The superclass *subsumes* the subclass. Therefore categories become subcategories of (are subsumed by) other categories: the class called `Chloroplast`, containing all the chloroplast individuals, is a subcategory of `Organelle` (all the chloroplasts are organelles but the class `Organelle` contains other types of individuals, like `Nucleus`, `Mitochondrion`, *etc.*). Combining all the class-subclass relationships of the ontology, the well known and intuitive is-a hierarchy or taxonomy is built (Figure 2.14).

The attributes of a given class apply to all of its individuals. To define such attributes, the relationships that the individuals should have are defined, in terms of how many relationships and to which individuals. Therefore the attributes of classes are usually *restrictions* on the relationships that the individuals have, and the restrictions limit the individuals that belong to the class. A restriction like `part_of some cell` is composed of three elements: the property (`part_of`), the quantifier (`some`), and the filler (`cell`). Such restriction can be read as *at least one relationship along the `part_of` property to an individual of the class `cell`* . The `some` quantifier is known as "existential" (the example would be an "existential restriction"): the other OWL quantifiers are `only` or "universal" (if there is a relationship with the given property it must be to an individual of the filler class, or nothing else), `exactly` (exact number of relationships), `min` (minimum number of relationships), `max` (maximum number of relationships), and `value` (a relationship to a concrete individual).

Restrictions and named classes can be combined using operators (`not`, `and`, `or`) to build highly expressive attributes or "class expressions". For example the filler of a restriction can be another restriction or a combination of various restrictions with named classes, or even restrictions with restrictions as fillers, creating nested restrictions. The complexity that can be achieved in class expressions is one of the strong points of OWL, and it allows the building of classes "on the fly" by combining other classes and restrictions. For example, the following restriction can be read as *part of at least one thing that participates only in reproduction and feeding* (an existential restriction that has another universal restriction as its filler, which has the union of two named classes as its filler): `part_of some (participates_in only (reproduction or feeding))`[28].

A class expression is an anonymous class (a class without URI) formed by the individuals that fulfil its conditions. Therefore, the class being defined using the class expression can be a subclass of it (*e.g.* `nucleus subClassOf part_of some cell`) or equivalent to it (*e.g.* `nucleus equivalentTo part_of some cell`).

The `subClassOf` and `equivalentTo` relationships to anonymous classes (class expressions) are known as "necessary" and "necessary and sufficient" conditions, respectively. A necessary condition (*e.g.* `nucleus subClassOf part_of some cell`) is a condition that it is necessary to fulfil but is not enough in itself to assume that an individual belongs to a class: all the nuclei are part of cells, and being part of a cell is a necessary condition for being a nucleus, but if we find an individual that is part of a cell, that fact on its own is not enough to consider the individual as a nucleus (it could be a chloroplast, which are also part of cell) (Figure 2.15). A necessary and sufficient

[28]Note the difference between the natural language "and" (as in *reproduction and feeding*) and the OWL `or` (as in `reproduction or feeding`): in both cases their meaning is the union of sets. In OWL, `and` means the intersection of sets. This leads to confusion on newcomers to OWL, as pointed in [85].

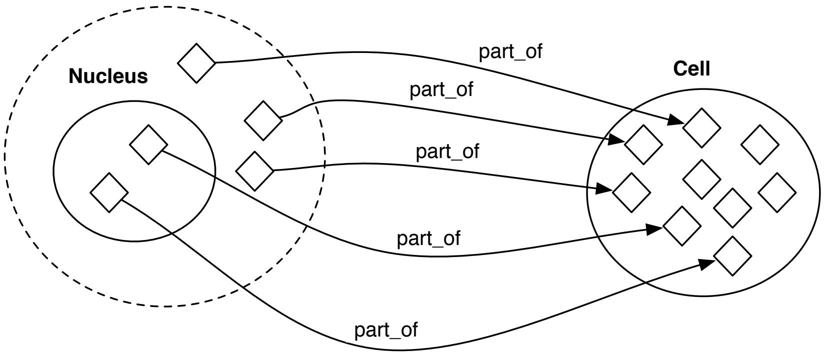


Figure 2.15: Necessary condition on the class `nucleus` made with an existential restriction (`nucleus subClassOf part_of some cell`). The named class `nucleus` is a subclass of the anonymous class (doted circle) formed by the individuals that have at least one `part_of` relationship to an individual of the named class `cell`.

condition (*e.g.* `nucleus equivalentTo has_part some nucleolus`) is a condition that suffices to determine that an individual belongs to a class: having a nucleolus is a fact that suffices to identify an organelle as nucleus, because, as nuclei are the only organelles with nucleoli, if we find an organelle which has a nucleolus as one of its parts we can assume it to be a nucleus (Figure 2.16).

The properties that link individuals in relationships (and therefore can be used in restrictions in class expressions) can be assigned different characteristics: transitive, functional, inverse functional, symmetric, asymmetric, reflexive and irreflexive. For example flagging a property as transitive means that if individual B is related to individual C and individual C is related to individual D, then individual B is also related to individual D. In another example, if a property is flagged as functional and individual A is related using such property to individuals B and C, it will be assumed that B and C are the same individual, thus if B and C are stated to be different in another axiom, an inconsistency will be flagged by the reasoner.

Properties can be arranged in hierarchies using `subPropertyOf` axioms[29]: if a property links two individuals in a relationship, then they are also related by the super-property (but not the other way around, as in the subsumption between two classes). Domain and ranges in properties define to which classes the properties should be applied.

Properties that link individuals are called Object Properties, but there is another

[29]Properties can also be disjoint or equivalent, they can be coupled in property chains, and they can be assigned inverse properties.

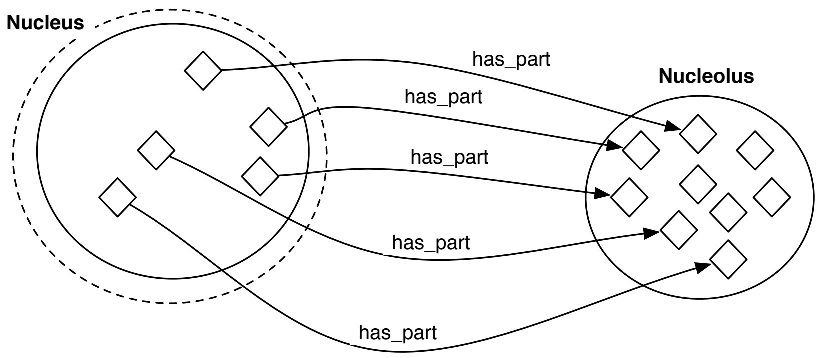


Figure 2.16: Equivalent condition on the class `nucleus` made with an existential restriction (`nucleus equivalentTo has_part some nucleolus`). The named class `nucleus` is equivalent to the anonymous class (doted circle) formed by the individuals that have at least one `has_part` relationship to an individual of the named class `nucleolus`: note that both the anonymous class and `nucleus` have exactly the same extent of individuals.

type of properties, called Datatype Properties, that can be used to link individuals to literal values (*e.g.* strings, dates, numeric values, *etc.*). OWL offers the possibility of adding annotations that are not processed by the reasoners, for storing human readable information, using Annotation Properties, the third and last type of property found in OWL. Entities (classes, properties or individuals) or axioms can be annotated with annotation values. Different types of annotation properties can be used: the RDFS annotation properties (`rdfs:label`, `rdfs:comment`, *etc.*), Dublin core properties[30], or any properties created by the ontologist.

The identity of named entities in OWL can be expressed by asserting that a given entity is different from other entities, using `sameAs` and `differentFrom` for individuals and `disjointFrom` and `equivalentTo` for classes. OWL does not work with the Unique Name Assumption (UNA), thus the fact that two entities have different names (URIs) does not imply that they are different entities: they could be equivalent classes or the same individual. However entities with the same URI *are* the same entity.

Another key assumption of OWL is the Open World Assumption (OWA). The OWA means that facts that are not asserted to be true are assumed to be *unknown* rather than false. The following assertion is used as an example of how open world and closed world systems differ: *DNA is located in the nucleus.* In a closed world system like a database, when the system is queried to know whether DNA is located in mitochondria, the answer is *no*, thus the system answers with negation as failure. However, in an

[30]`http://dublincore.org/`

OWL based system, the answer would be *unknown*: it could be that DNA is located in the nucleus and also other parts of the cell (*e.g.* mitochondria), as the only stated knowledge is that DNA is located in the nucleus. In OWL, negation has to be explicitly stated: it has to be asserted that DNA is *not* located in mitochondria (and make nucleus and mitochondria disjoint, if they were not, due to the lack of UNA). Therefore the closed world assumption in OWL can be achieved, but it must be done so explicitly.

OWA and the lack of UNA in OWL result in a modelling process that can be thought of as "trimming" a first ontology where everything is possible and every entity is equal. The ontologist trims the ontology by limiting the possibilities: entities are asserted to be different, *e.g.* with `differentFrom`, and axioms that limit the possible facts are introduced, *e.g.* by using restrictions. The ontologist trims the all-possible ontology until a satisfactory representation of the domain of knowledge is reached.

Precise semantics and a limited expressivity in OWL allow sound, complete and (in many cases) efficient automated reasoning. Using automated reasoning the following tasks can be performed:

DL queries: Anonymous classes can be defined and the reasoner can answer how the classes and individuals of the ontology relate to those classes, which is equivalent to "asking" questions about the knowledge represented in the ontology. For example we can define the query `part_of some cell` (which represents the set of individuals that have at least one `part_of` relationship to an individual from the class `cell`) and ask the reasoner to retrieve its named superclasses (and ancestor classes), equivalent classes, subclasses (and descendant classes), or individuals.

Subsumption hierarchy: The reasoner infers the structure made of the class-subclass relationships, even if not all of them are explicitly stated. For example if the class `organelle` has the restriction `part_of some cell` as an equivalent condition (*anything that has at least one part of relationship to a cell is an organelle*) and the class `mitochondrion` has the same restriction as a superclass condition (*all the mitochondria have at least one part of relationship to a cell*), the class `mitochondrion` will be inferred to be a subclass of the class `organelle` (if all the mitochondria are part of a cell, and anything that is a part of a cell is an organelle, then a mitochondrion is an organelle). Further subsumption relationships (*e.g.* subclasses of the class `mitochondrion`, like `plant mitochondrion` and `animal mitochondrion`) are also inferred, rebuilding the whole taxonomy.

Realisation: Given an individual with relationships, the reasoner can infer to which named classes it belongs. For example if the individual `Organelle1` has a relation `part_of cell`, it will be inferred to be a member of the class `organelle`.

Consistency checking: The reasoner can check whether the ontology is consistent, thus there are no contradictions. An inconsistent ontology does not have any model, thus, there is no interpretation that satisfies every axiom of the ontology. The reasoner can also check whether classes in the ontology are satisfiable: an unsatisfiable class cannot have any instances in any model of the ontology, thus it is interpreted as an empty set in every model of the ontology [58], and flagged as a subclass of `owl:Nothing`. For example if the class `GeneticMaterial` is defined as `subClassOf part_of some mitochondrion and nucleus`, and `mitochondrion` and `nucleus` are disjoint, the reasoner will flag the class `GeneticMaterial` as unsatisfiable, as there cannot be any instance of `GeneticMaterial`. Unsatisfiable classes usually indicate an error in the modelling.

OWL semantics is monotonic, and hence new axioms can be introduced and the prior axioms must remain valid. Thus additional axioms can result in new inferences, but never cancel the prior inferences: *e.g.* if a class is flagged as unsatisfiable after adding new axioms, that is a new inference and does not invalidate prior inferences, although it indicates a likely error in the modelling [84].

An OWL ontology can reuse descriptions in other ontologies, being able to refer to the entities of the other ontologies *via* axioms (the other ontologies are "imported" by the ontology). For example, in the class definition `nucleus subClassOf ro:part_of some cl:cell`, there are two references to entities of other ontologies: `part_of` (`ro` ontology) and `cell` (`cl` ontology).

2.3.4.2 OWL in relation to RDF and RDFS

As mentioned, compared to RDF, OWL and RDFS are designed as ontology languages. Whereas RDF represents metadata, OWL and RDFS are aimed at representing knowledge about those metadata. In the same perspective, RDF can be seen as an OWL ontology with only individuals. An RDF model about a concrete family would have, for example, some names (subjects) and some relationships (predicates) (Figure 2.17): `Adams_family has_parent John`, `Adams_family has_child Mary`, `Adams_family has_child Simon`, *etc.* However, the knowledge about the usual composition of a

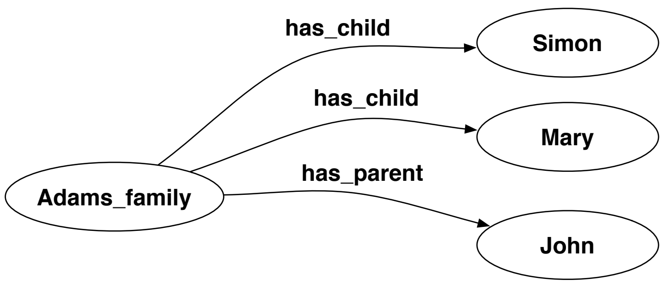


Figure 2.17: RDF graph describing a concrete family. The adams family has a parent called John, and two children, Simon and Mary.

catholic family (*e.g.* a male father, a female mother and at least 5 children) should be encoded in OWL, as it describes classes of entities with general attributes (the class composed by the set of catholic families). The class `catholic_family` would have different restrictions stating the number of children, the types of parents, *etc.* (Figure 2.18). Such restrictions should be added as superclasses, as not all the families with such attributes are catholic, but all the catholic families have such attributes. `John` would be an individual of the class `male`, `Simon` would be an individual of the class `child`, and the instance `Adams_family` would be classified (or not) as belonging to the class `catholic_family` (Figure 2.19).

OWL, in comparison with RDFS, provides a richer vocabulary for modelling and precise semantics, which comes from restricting the complete syntactic freedom of RDFS, *e.g.* in OWL (DL) a cardinality restriction cannot be applied to the subsumption relationship.

OWL (RDF/XML) includes RDFS and RDF elements in its syntax, and OWL, RDFS and RDF are intended to be syntactically compatible; a tool able to process RDF serialised in XML should be able to process OWL serialised in RDF/XML, albeit not considering its semantics, thus treating an OWL model as an (unnecessarily convoluted) RDF graph (Figure 2.5). For example, in Figure 2.11, there are RDFS constructs like `<rdfs:subClassOf>`, RDF triples can be noted, and everything is valid XML.

2.3.4.3 OWL and LSSW

OWL was designed for the Semantic Web, and four of its features stand out:

OWL is self-descriptive: The data and the schema of the data are delivered together [92], so data from different resources can be integrated without schema reconciliation, albeit semantic reconciliation may be necessary.

```
Class: catholic_family
    SubClassOf:
        has_parent only (female or male),
        has_parent exactly 1 female,
        has_parent exactly 1 male,
        has_parent exactly 2 (female or male),
        has_child min 5 child
```

Figure 2.18: MOS rendering of the OWL class catholic_family. The class catholic_family is a subclass of the anonymous class made of the families that have exactly one female parent, exactly one male parent, exactly 2 parents, in case of having a parent he or she must be from the class female or the class male (or none), and they have a minimum of 5 relationships to the individuals of the class child.

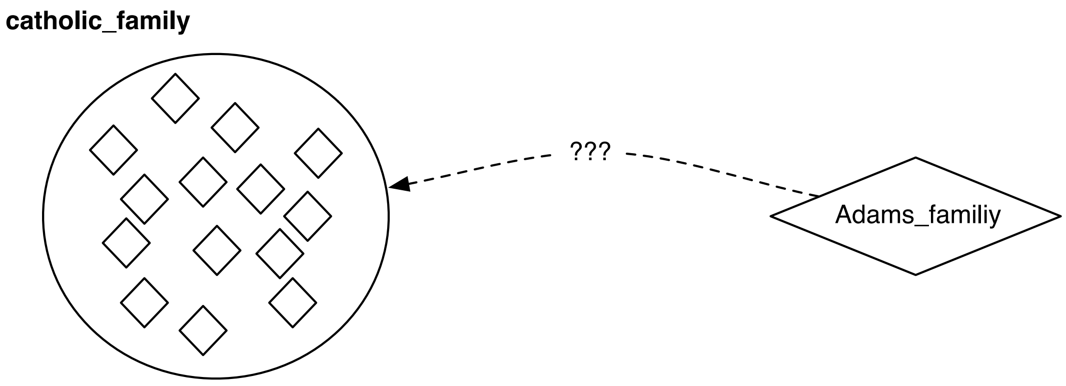


Figure 2.19: The OWL class catholic_family (left) and the individual Adams_family (right). The individual Adams_family will be (or not) inferred to be a member of the class catholic_family, which includes all the families that match the attributes defined in Figure 2.18.

Open World Assumption: To work in the Semantic Web, we need to assume that the knowledge we hold is necessarily incomplete: more knowledge from new resources will be most probably added at later stages to our ontology.

Absence of Unique Name Assumption: In the Semantic Web it could be that different entities are equivalent, *e.g.* different resources could have assigned the names "cat", "gato" and "katua" to the same concept, *Felix catus*. Therefore the lack of UNA is important to make the Semantic Web work.

URIs: URIs globally identify resources on the internet and URLs locate them. OWL uses URIs to identify entities, so the already existing internet infrastructure for URIs can be exploited.

The features that make OWL suitable for using it on the implementation of the Semantic Web also make it suitable for representing the current distributed biological knowledge in the LSSW. For example the Open World Assumption takes into account the inherent incompleteness of biological knowledge, as new resources and data need to be continuously added to the existing ontologies. Also, the importing mechanism and the fact that OWL is self-descriptive fits with the distributed nature of biological knowledge, with independent resources publishing their own data. Finally, URLs can be used to identify entities from the biological knowledge domain [49].

On the other hand, OWL is limited in some aspects with regards to expressing biological knowledge, as described in [108]. For example OWL cannot represent relations of higher arity than 2, there are no notions for exceptions, temporal automated reasoning cannot be applied, *etc.* However, even with limitations on what can be expressed, OWL presents what is a currently optimal balance between expressivity and tractability, it is a standard (facilitating interoperability), and it is included in the Semantic Web Stack, offering many functionalities "for free". The development of OWL is active, and new features of expressivity are expected in the future.

2.3.5 Open Biomedical Ontologies (OBO) format

The OBO format[31] was first designed for implementing the GO, but it has become the *de facto* KR language for implementing many bio-ontologies. However, it is not used as a KR language outside the life sciences domain.

[31] `http://www.geneontology.org/GO.format.obo-1_2.shtml`

The OBO format was originally developed as a flat file format. Although there is a XML based syntax for OBO[32], the most used syntax is the flat file. A considerable bio-ontologists community has evolved around OBO, creating ontology editors like OBO-Edit [24], Phenote[33] or COBrA-CT[34], and APIs like ONTO-PERL[35] or GO-PERL[36].

OBO does not map to any precise semantics [72], but OBO to OWL translations have been implemented [46] and DL automated reasoning has been applied with certain adaptations of the OBO ontologies [73, 45]. Besides automated reasoning through adaptations to OWL, other types of algorithmic processing of OBO ontologies have been implemented, like the OBOL project (OBO Language) [77] and the OBO-Edit reasoner[37]. OBOL exploits the grammar implicit in GO term names to infer new relationships. The OBO-Edit reasoner performs an already defined set of inferences, or "rules" (*e.g.* "Rule 1: transitivity"), thus the inference is implemented in the logic of the program instead of using already existing (and thoroughly tested) reasoners that offer a wider range of inferences, like Pellet or FaCT++. For example, the OBO-Edit reasoner cannot execute a query of arbitrary complexity, or check the consistency of the whole model, as it does not exploit any formal semantics. Therefore, the OBOL and OBO-Edit attempts cannot be considered as examples of automated reasoning.

In comparison to OWL, OBO has a limited expressivity. For example, OBO does not allow for nested expressions, and the distinction of necessary and necessary and sufficient conditions is not present, nor the distinction of properties (Object Properties, DataType Properties, and Annotation properties).

2.4 Current bio-ontologies

The use of bio-ontologies is "mainstream" now in bioinformatics, as shown by events like the bio-ontologies Special Interest Group[38] at the international conference of Intelligent Systems for Molecular Biology (ISMB) or institutions like the the National Center for Biomedical Ontology[39] (NCBO). Most of the current bio-ontologies are

[32]http://www.geneontology.org/GO.format.shtml#OBO-XML
[33]http://www.phenote.org/
[34]http://www.aiai.ed.ac.uk/project/cobra-ct/
[35]http://search.cpan.org/~easr/ONTO-PERL-1.13/
[36]http://search.cpan.org/~cmungall/go-perl/
[37]http://oboedit.org/docs/html/The_OBO_Edit_Reasoner.htm
[38]http://www.bio-ontologies.org.uk/
[39]http://bioontology.org/

gathered under the Open Biomedical Ontologies Foundry [102], a foundry that sets design principles to make the member bio-ontologies non-overlapping, of high quality and interoperable[40]. Current bio-ontologies are used to fulfil different tasks [109], listed as follows:

Reference ontology: An ontology can act as a reference of the domain knowledge, since it has been built through a consensus in order to obtain a canonical model of such domain.

Controlled vocabulary: An ontology is a defined set of terms that users commit to in order to transmit information. A controlled vocabulary can be regarded to be delivered by an ontology: an ontology is a structure where objects are classified in categories, and once the community has agreed on such categories and their labels, a controlled vocabulary is in place.

Schema and value reconciliation: Ontologies can be used to reconciliate the schemas and the instances of different databases that refer to the same domain.

Consistent query: The structure or simply the controlled vocabulary of an ontology can be exploited for enhancing querying.

Knowledge acquisition: An ontology can be used to generate forms for adding instances to the model, as the ontology stores the attributes of the classes to which such instances pertain.

Clustering and similarity: Data can be clustered using the semantic similarity of the classes against which such data is annotated in an ontology.

Indexing and linking: A controlled vocabulary can also work as an index to retrieve objects or data.

Results representation: Ontologies can be used as schemas that scientists must comply with in order to represent some results, *e.g.* results from microarray experiments.

Classifying instances: Given a concrete instance, an ontology can be used to classify it into the correct category, by applying automated reasoning to its attributes.

[40]`http://www.obofoundry.org/crit.shtml`

Text analysis: An ontology can be used in text mining either as a controlled vocabulary or exploiting the structure of the ontology.

Guidance and decision trees: An ontology, since it is a representation of the objects of a knowledge domain, can guide a user when making decisions.

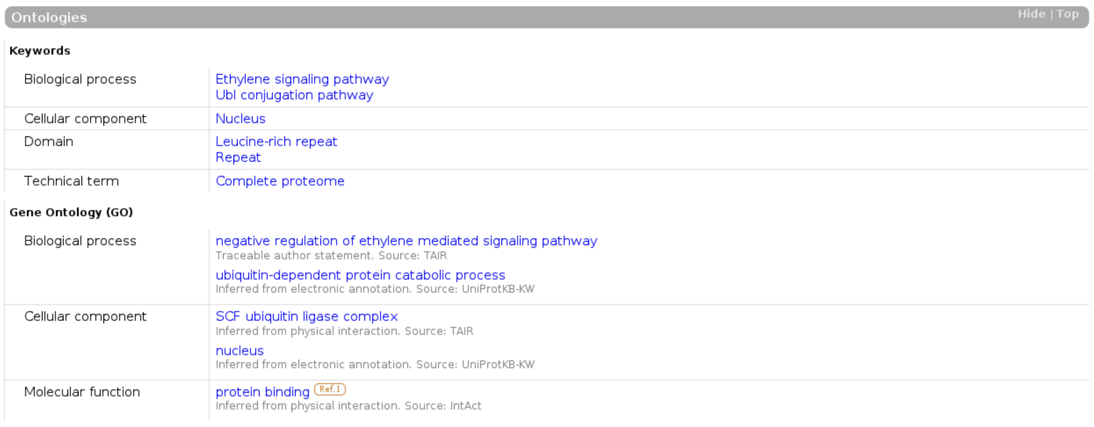
Much of the controversy on how to build and maintain bio-ontologies rises from the fact that an ontology can perform one or more of the tasks described above. Therefore, current bio-ontologies vary in (besides the already mentioned axiomatic richness, rigour and usage) content type, scope, size, querying mechanisms, interoperability, *etc.* An overview of the most important current bio-ontologies follows.

GO is the most successful (domain) bio-ontology [11]. GO provides a controlled vocabulary to describe the cellular location, the molecular function and the biological process of gene products, providing a *de facto* integration mechanism for several gene product databases. The terms of GO are organised in a structure formed by the following relationships: `is_a`, `part_of`, `regulates`, `positively_regulates` and `negatively_regulates`. An example of how GO can be used for integration is shown in Figure 2.20, in which the protein `Q708Y0` is annotated with terms from GO. Many other proteins from other databases are also annotated with GO terms, which allows the retrieval of all the proteins from disparate resources that are annotated, *e.g.* with the GO term `Negative regulation of ethylene mediated signalling pathway`, or to exploit GO's structure for a more expressive querying of the resources. The annotations of entities of databases with GO terms are collected in the GOA (Gene Ontology Annotation) project [20].

GO has become a *reference* bio-ontology and it is used for other tasks besides integration of database contents, like analysis of high throughput data [65, 128] or enhanced web browsing of biology resources [15]. There are other important bio-ontologies that describe different domains, like CL for canonical cell types, SO for sequence features or ChEBI [26] for biochemical entities.

Other bio-ontologies are more focused in concrete applications and information exchange rather than collecting canonical knowledge about a concrete biological knowledge domain. Two examples stand out, namely BioPAX[41] and OBI, providing a language for describing biochemical pathways, and life science experiments, respectively.

[41]`http://www.biopax.org/About.html`



Ontologies — Hide | Top

Keywords	
Biological process	Ethylene signaling pathway Ubl conjugation pathway
Cellular component	Nucleus
Domain	Leucine-rich repeat Repeat
Technical term	Complete proteome
Gene Ontology (GO)	
Biological process	negative regulation of ethylene mediated signaling pathway Traceable author statement. Source: TAIR ubiquitin-dependent protein catabolic process Inferred from electronic annotation. Source: UniProtKB-KW
Cellular component	SCF ubiquitin ligase complex Inferred from physical interaction. Source: TAIR nucleus Inferred from electronic annotation. Source: UniProtKB-KW
Molecular function	protein binding Ref.1 Inferred from physical interaction. Source: IntAct

Figure 2.20: Fragment of the UniProt entry for the protein `Q708Y0`. In this fragment the references to GO terms are shown.

Other bio-ontologies are used to build Knowledge Bases (KBs). A KB is a storage system that includes an ontology as a schema (TBox) and the instances that can be captured in such schema (ABox). The instances are usually the entities of the knowledge domain, and the ontology describes the classes of entities. For example, in a KB storing information about people, the class `human` would have a lot of instances (`Mikel`, `Robert`, `Simon`, *etc.*), as the class `human` holds the attributes that a human is supposed to have. A KB differs from a database in a crucial point: a KB makes inferences about the stored information, exploiting precise semantics, besides retrieving it, whereas a database only retrieves the information [27]. Databases are suited for regular and complete information, as extending a database schema requires intense work. Contrary to databases, KBs are suited for storing changing and incomplete information, as extending the schema is less laborious. Therefore KBs fit the requirements for representing biological knowledge, as mentioned in Section 2.3.4.3.

There has been an increase in the number of KBs that store biological knowledge in recent years, due to the improvement in performance of RDF and OWL storing systems, reasoners and querying languages (especially in the case of RDF). In many of those systems OWL, RDF and RDFS are used together, or also ontologies are mixed with databases. The KB built using CCO (CCO KB) is an example of an OWL and RDF mixed system. The CCO KB stores information about proteins (UniProt), molecular interactions (IntaCt [64]) and other data gathered using a Perl pipeline [5]. The KB can be accessed using SPARQL[42]. Other examples of this mixed approach include

[42] `http://www.cellcycleontology.org/query/sparql`

Thea-online [79] and SWAN [37]. The BioGateway resource[43] includes the CCO KB and many other databases and OBO ontologies in an RDF KB (Uniprot also provides all its entries in RDF, but no official SPARQL endpoint is provided to the data). Pure OWL KBs include FungalWeb [13] (a KB of fungal enzymes) and yOWL [118] (a KB for data about yeast). Ontology-database approaches have been implemented in projects like Phosphabase [125], a system for storing and applying automated reasoning over information about protein phosphatases.

Finally, there are ontologies that are used in the life sciences to provide a framework for bio-ontologies integration. Those ontologies are not focused on representing a domain, but rather in providing a conceptual scaffold where the actual modelling can rest. The Basic Formal Ontology (BFO) [51] is one of the most important Upper Level Ontologies used for bio-ontologies. It describes general classes of concepts (*e.g.* process, physical entity) making the modelling more robust and allowing for integration of different bio-ontologies. The Relation Ontology (RO) [101] provides a set of rigorously defined relationships that all the bio-ontologies should use for integration purposes. For example the `part_of` relationship in the GO `cellular component` subtree is the same `part_of` relationship as the `part_of` relationship of other anatomical ontologies, and therefore they should all use the same `part_of` from RO, which has a rigorous semantic definition.

2.5 Quality problems of current bio-ontologies

Many current bio-ontologies are not fine grained and robust representations of biological knowledge, as most of them have been implemented without exploiting the capabilities of KR languages, especially with regards to OWL.

The main reason for such lack of quality is that KR languages, and especially OWL, are difficult and anti-intuitive [85, 92, 129], and hence are not exploited as they should be. The Open World Assumption and automated reasoning are difficult to comprehend, especially the consequences of changing axioms in relation to the outcome of automated reasoning. At the same time, biological knowledge is difficult to model [110], especially with the fine grained resolution necessary for automated reasoning to yield meaningful results. Therefore, the costs (difficult modelling) are perceived as bigger than the benefits (automated reasoning, Semantic Web oriented integration, *etc.*).

[43]`http://www.semantic-systems-biology.com/biogateway/querying`

Another reason for the lack of quality lies in the perception that many biologists have of bio-ontologies. For example, a lot of bio-ontologies are seen as controlled vocabularies for annotations rather than representations of knowledge [104, 42, 70]; thus, the current emphasis on bio-ontology development is in human consumption of the information instead of computer consumption (automated reasoning). There is the perception that if a bio-ontology suffices to collect important terms from annotations in a minimally structured fashion, that should be enough. This perception is prevalent due to the fact that bio-ontologies must deliver solutions for integration *here* and *now* [43, 106], and keep up to date in relation to the content from databases. GO, for example, was quickly released and demonstrated to be useful from the very beginning, even with a minimal structure.

The quality problems of current bio-ontologies lie in two areas: lack of rigour and lean axiomisation. Lack of rigour also affects axiomisation, as axioms cannot be exploited for automated reasoning without rigorously describing them in a KR formalism with precise semantics. For example, even though potential inferences are described in initiatives like [71], the axiomisation is not rich enough for meaningful automated reasoning (*e.g.* defined classes are present in the ontology but no new superclasses are inferred: the asserted and the inferred hierarchies have exactly the same form).

The modelling behind many bio-ontologies is not rigorous [106]. For example the semantic interpretation of GO is informally defined in documentation, if defined at all [46]. This means that the reading of the `part-of` relationship can only be interpreted by humans from the documentation [72], as it is not explicitly stated in the model [105]. The GO documentation states that the `part-of` relationship in GO should be regarded as *necessarily is part*, that is, whenever the part exists it does so as part of the whole, but the whole does not necessarily always have the part as its part[44]. Another consequence of the lack of rigour is that the relation `is-a` is not used in concordance with its intended semantics [105]. There are other examples of non-rigorous modelling in GO. For example the representation of biological taxa in GO does not follow a formal theory like the one presented in [96], which creates conflicts around the usage of the word "sensu" [103, 127]. Different levels of granularity are also mixed in GO [67]. These problems can be extrapolated to the majority of the OBO ontologies.

Besides limited rigour, most of the OBO ontologies also have a limited axiomisation [42]: only a few relationships are used and there is no qualification of relationships. In the OWL versions of many OBO ontologies, only existential restrictions are

[44]`http://geneontology.org/GO.usage.shtml#partof`

used, without even disjoints[45].

Most of the biological knowledge found in bio-ontologies is "buried" in labels instead of being codified in axioms [73, 30, 104, 2]. This is due to the fact, that, as mentioned, the priority for many bio-ontologies is efficient annotation integration rather than knowledge modelling *per se*. Thus, the bio-ontologies are designed to be mainly human readable. Plenty of semantic content is also buried in the logic of *ad hoc* programs that are written to integrate and process data. For example, as part of the CCO pipeline, molecular interactions from IntAct had to be modified in order fit in the CCO schema: such equivalences are not explicit.

Another example of implicitly defined axiomisation, coming back to GO, is the propagation of the relationship `regulates` along `part-of` and `is-a`, which is only mentioned in the documentation but not implemented in the ontology[46].

Having a rich axiomisation expressed in a rigorous manner using a KR formalism like OWL allows the exploitation of automated reasoning, which has different benefits. Automated reasoning allows for querying: *e.g.* in the current GO the user cannot query for processes that happen during other processes (*e.g.* cytokinesis during cell cycle) as `during` is not codified in a property. The richer the axiomisation, the more complex queries can be done. For example a transitive superproperty of a functional object property in the ontology allows for modifying the extent of a query (*e.g.* the Sequence ODP[47]).

Automated reasoning can also be used for maintenance and consistency checking. GO is currently `is-a` complete [41], which means that every term has got at least one `is-a` relationship to a root term (`molecular function`, `biological process`, or `cellular component`), assuming `is-a` to be transitive. Maintaining such structure in an ontology of over 20,000 terms is difficult and incomplete, as shown by the GONG project [73]. In the case of consistency checking, if an ontology is sufficiently rich automated reasoning can be applied to check the definitions of newly added terms and hence ensure consistency, or to check the consistency of the whole model [8].

A richer axiomisation would result, for example, in better analysis of high throughput data [127] or deeper analysis of annotations [2]. Rich axiomisation, coupled with OWL imports, also means that composition of terms is more explicit and hence easier to test, which is a clear target for OBO ontologies, as they are all supposed to be

[45]The same applies for most of the existing ontologies outside the biological knowledge domain [9].

[46]`http://geneontology.org/GO.doc.shtml#relationship-transitivity`

[47]`http://www.gong.manchester.ac.uk/odp/html/Sequence.html`

orthogonal to each other [102]. As bio-ontologies can be used as the data model behind applications, a richer axiomisation allows a wider range of functionalities on the software side [84].

There have been different attempts to improve the axiomisation and the rigour of bio-ontologies. For example *ad hoc* consistency checking and enrichment of GO has been attempted in different projects. In those projects GO has been translated to other formalisms, consistency has been checked and the results have been fed back to GO. The work presented in [127] performed such consistency checking using frames in Protégé 3. The GONG [73] and OBOL [77] projects performed consistency checks using OWL and the OBOL language, respectively. However, GONG and OBOL went further and added more axioms to GO as a result of syntactically dissecting GO labels. Alignment discovery between different OBO ontologies was performed in the work presented in [12].

The problem with the initiatives described above for improving bio-ontologies is that they are bespoke solutions, and hence hardly reusable. There is a need in bio-ontology engineering for principled and abstract solutions that enhance axiomatic richness and rigour. Such solutions should be able to be reused in different bio-ontologies, and created and shared by the bio-ontologists, as is the case in other engineering disciplines. Such solutions should be able to be used off-the-shelf, without *ad hoc* implementations, and not only for enrichment, but in the whole bio-ontology life cycle (design, implementation, maintenance). Also, they should facilitate the usage of OWL in bio-ontologies, as that is one of the main reasons for their current low quality. The use of Ontology Design Patterns (ODPs) in the development of bio-ontologies is a solution that fulfils such requirements.

2.6 ODPs

Abstraction in design has been a constantly respected principle in every new engineering field. This includes software engineering through the abstraction offered by the software design patterns widely used in object oriented programming [33]. Using abstraction, the expressive power of a language can be encapsulated in discrete models that have well understood properties and are easier to use and share, efficient, and well documented, therefore resulting in a more efficient engineering process that generates higher quality software artefacts. ODPs represent the same principle in the field of ontology engineering. However, given that ontology engineering is a newer field compared to software engineering, ODPs' definition, representation and application methods lack the same level of consensus that the software design patterns enjoy.

2.6.1 ODPs in the literature

The literature about ODPs can be divided in two areas: literature that discusses the notion of ODPs and literature that presents concrete ODPs for tackling concrete design problems in ontologies.

The literature about the notion of ODPs begins with the paper about Ontological Design Patterns [88], in 2000. Such ODPs are best practices for molecular biology metadata modelling. Later, in 2001 and 2003 respectively, the ideas of Semantic Patterns [107] and Knowledge Patterns [23] were presented, as reusable components for building KBs. In 2004 an idea of ODPs was briefly mentioned in [112]. In 2005 a distinction between Logical Ontology Design Patterns (LODPs) and Concept Ontology Design Patterns (CODPs) was made in [35]. LODPs provide examples for solving modelling problems within the realm of the KR languages used, without any reference to concrete entities or models. For example, a LODP shows how to use the equivalence axiom in OWL, regardless of the identity of the equivalent classes. Also in 2005, the notion of OWL Macros for ontology engineering was presented in [121]. Finally in 2007 the notion of Content Ontology Design Patterns (CODEPs) was introduced, as instantiations of LODPs in concrete domains, in [34]. In [34], further types of best practices were presented (*e.g.* architectural design patterns, syntactic patterns, *etc.*).

Regarding the literature that presents concrete ODPs, most of the attempts have been individual examples of best practices, like the well known and thoroughly researched problem of mereology representation [90, 1, 94] and the related problem of propagation along transitive roles [93, 82, 97, 95, 98]. Granularity [81], default knowledge [54], possibilities [53], modularisation best practices [83, 87] and Upper Level Ontologies [86, 51] have also been proposed in the literature.

Outside the literature on KR, the W3C Semantic Web Best Practices and Deployment Working Group[48] (W3C BPD) has collected some best practices of ontology engineering, *e.g.* the n-ary relationship[49] and the value partition[50].

[48]http://www.w3.org/2001/sw/BestPractices/
[49]http://www.w3.org/TR/swbp-n-aryRelations/
[50]http://www.w3.org/TR/swbp-specified-values/

2.6.2 ODPs in this work

2.6.2.1 Comparison to other initiatives

The notion of ODPs used for this thesis differs and has some similarities with some of the other attempts described in Section 2.6.1.

Some of those attempts are equivalent to the idea of ODPs in this thesis, like the idea of ODPs described in [112], the OWL Macros described in [121], the CODPs described in [35], and the CODEPs described in [34]. Other initiatives are different to the ODPs of this thesis. For example the Ontological Design Patterns [88], the Semantic Patterns [107] and the Knowledge Patterns [23] have the drawback of being represented in a formalism that needs to be translated into actual KR languages such as OWL, making them difficult to use; ODPs in this work are represented directly in OWL. LODPs [34] are also different to the notion of ODPs used for this work, as they can be too simplistic in some cases, *e.g.* the subClassOf LODP is completely uninformative as a best practice (in this work, there is a minimum of expressivity that an ODP must have in order to be considered as such and included in the online catalogue of ODPs). The hierarchical model presented in [34], albeit formally correct, makes the usage of those best practices an unnecessarily complex task for the aim of this work. For example, LODPs are sometimes semantically equivalent to CODEPs, which adds unnecessary complexity as they can be used in the same way. The ODPs of this thesis are instances that implicitly represent a more abstract structure, instead of making the division between logical and content levels, thus making the ODPs more usable.

There are two areas that are lacking in the literature about ODPs reviewed above: application mechanisms (except the OWL Macros described in [121]), and, in the case where concrete ODPs are presented (*e.g.* [34]), specific ODPs for modelling biological knowledge.

Regarding the literature that presents concrete ODPs, some of the best practices described are equivalent to ODPs of the online catalogue of ODPs: appropriate references have been included in the online catalogue in such cases. Examples include lists [28, 108], exceptions and n-ary relationships [108]. Also, one of the OWL Macros presented in [121] is equivalent to the Closure ODP[51]. The Closure ODP represents a simple but necessary and often omitted modelling step [85]: closing a restriction so that it only points to a given class. Many newcomers to OWL assume that using

[51] `http://www.gong.manchester.ac.uk/odp/html/Closure.html`

an existential restriction suffices to assert that the class can only have one relationship along a given property. For example, many users assume that the expression `person subClassOf has_head some head` means that people have only one head. What it really means is that people must have at least one head, but can have more than one, which is obviously incorrect modelling. For example, if a DL query of the form `query equivalentTo has_head some head` is performed, an alien with more than one head (`alien subClassOf has_head min 5 head`) would be retrieved altogether with the human, both as subclasses of `query`. Therefore, the existential restriction must be combined with an universal restriction to obtain a closed relationship: `person subClassOf (has_head some head) and (has_head only head)`. The universal restriction asserts that, if a person has a `has_head` relationship, it must point to the `head` class, and nothing else. The universal restriction, on its own, is not sufficient either, so a maximum cardinality constraint needs to be added (`person subClassOf (has_head some head) and (has_head only head) and (has_head exactly 1 head)`). It is a simple axiomisation, but it is not obvious [85].

The two examples from the W3C BPD are represented in the online catalogue of ODPs as the Nary Relationship ODP[52] and the Value Partition ODP[53].

The main difference between those best practices and the ODPs presented in this work is the target knowledge domain, which in the case of this work is biological knowledge. In fact many of the examples of the online catalogue of ODPs are applications of ODPs in real bio-ontologies like GO or SO. There is, however, a emerging trend in literature on KR best practices specific to biological knowledge, exemplified by the recent publication of a pattern for representing biological taxa [96].

Another difference with the mentioned best practices is that the ODPs presented in this work are systematically described and classified in a centralised online catalogue of ODPs and an application procedure is provided, based on OPPL.

[52]http://www.gong.manchester.ac.uk/odp/html/Nary_Relationship.html
[53]http://www.gong.manchester.ac.uk/odp/html/Value_Partition.html

2.6.2.2 A working definition for ODPs

The intuitive definition of an ODP is straight forward: a well known and tested "recipe" that helps the ontology engineer to create a richer ontology or maintain it more efficiently, by providing a solution to a common modelling problem. For example, a recurrent problem in life sciences is how to model things that are duplicated along a symmetry axis, like the right hand and left hand in humans. The Selector ODP[54] presents a solution to that problem, explained as follows. The entity is the same (hand), the only difference being in which position of the symmetry axis (sagittal plane) it stands: right or left. The Selector ODP consists of using only a `hand` class (instead of `right hand` and `left hand`), and adding `left` and `right` as independent values (those values can also be used *e.g.* for left and right leg). Whenever a reference to a concrete hand needs to be added, like an infection happening only in the right hand, a restriction of the form `happens_in only (hand and has_laterality some right)` should be used. Using such a procedure the amount of classes of the ontology considerably decreases.

In this thesis the intuitive definition is assumed to be enough, because the aim of this work is not to improve KR *per se*; the aim is to help bio-ontology curators to better represent biological knowledge through the *use* of ODPs, and therefore a *working* definition for ODPs is provided.

ODPs can be presented to bio-ontologists as abstract models or concrete instances. As the focus of this research is to provide easy to apply ODPs, they are presented as instances instead of abstract models. Another reason for doing so is that there is no easy to use abstract graphical representation for OWL. The difference between the abstract model and the concrete instances can be appreciated by comparing Figures 2.21, 2.22 and 2.23 (the UML to OWL mapping used is described in Figure 3.3). The Value Partition ODP[55] is used as a running example. The Value Partition ODP is used to model the fact that a parameter can only take certain values, thus, the parameter is said to be "exhausted" by such values. For example, it could be that in an ontology about people, a person can only be tall, medium or short. In such ontology this ODP is used, *e.g.* to avoid the introduction of any extra value by another developer (*e.g.* very tall) or to make it possible for a reasoner to infer that a person that is not tall and has a height must be either medium or short (provided that some axioms of the ontology enforce the fact that a person must have a height). The structure of this ODP consists of the parameter class (*e.g.* `Height`) and the values of the parameter as subclasses of

[54]`http://www.gong.manchester.ac.uk/odp/html/Selector.html`
[55]`http://www.gong.manchester.ac.uk/odp/html/Value_Partition.html`

the parameter class (*e.g.* `Tall`, `Medium`, and `Short`). The value classes are disjoint. The key element is a "covering axiom": the parameter class is equivalent to the union of the value classes, therefore if a new subclass is added to the parameter class the reasoner will flag the parameter class to be inconsistent. Figure 2.21 is a completely abstract representation of the Value Partition ODP. Figure 2.22 is an instance of such representation, because, *e.g.* two classes are used for the values (`V1` and `V2`), whereas in the abstract model an arbitrary amount of values is represented. Figure 2.23 is an even more concrete representation of the abstract model, as concrete values are used (`Tall`, `Medium`, and `Short`).

Therefore, we define ODPs in this thesis as concrete instances that exemplify more abstract structures. Such structures are well known and thoroughly documented best practices of bio-ontology engineering. ODPs are represented in OWL, a concrete language, instead of using a more abstract formalism. As defined for this thesis, ODPs differ from software design patterns because software design patterns are not fragments of source code (they are abstract structures not defined in the same language as the target language), whereas ODPs are presented as fragments of OWL ontologies or "mini-ontologies". This is done that way because there is not a satisfactory mechanism for graphically representing abstract OWL structures.

The limit between an ODP and a small concrete ontology is fuzzy. The difference lies in that an ODP describes a way in which ontologies should be built, whereas a small ontology is an end on its own, as it describes a part of the knowledge domain. For example, even though there are different Upper Level Ontologies available, there is an Upper Level Ontology ODP, because using top-level distinctions is a good practice, regardless of which values they take. However, RO is not an ODP because it is a concrete representation of the knowledge domain, describing the properties that relate entities of biological interest.

ODPs also differ from ontology building methodologies like Methontology [47] in the scale of the solution: ODPs should be used to tackle concrete modelling problems in concrete ontologies, whereas methodologies provide more general strategies on how to build ontologies.

2.6.2.3 Advantages of using ODPs

This section describes the advantages of the use of ODPs for ontology engineering [10]. Some of the advantages have been already demonstrated in the literature, other advantages are reasonable assumptions based in the author's and other colleagues'

$P_1 \equiv V_1 \sqcup ... \sqcup V_n$
$V_1 \sqsubseteq P_1$,..., $V_n \sqsubseteq P_1$
$V_1 \sqsubseteq \neg\, V_n$,..., $V_{n-1} \sqsubseteq \neg\, V_n$

Figure 2.21: DL notation of the abstract structure of the Value Partition ODP. The structure of the Value Partition ODP is completely abstract because there is not a set number of values (V_1 ,..., V_n), yet the structure of the ODP is completely described by this notation.

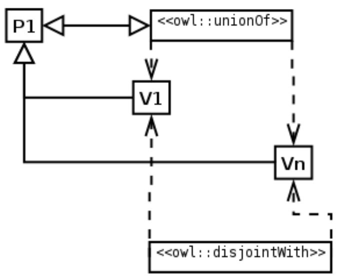


Figure 2.22: Instance of the structure of the Value Partition ODP. This is an instance of the structure shown in Figure 2.21, as two values (instead of any number of values) are used, even though they are not concrete values.

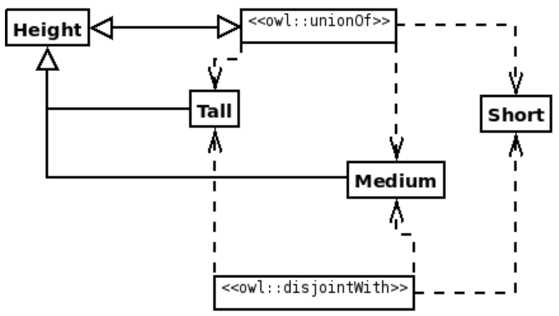


Figure 2.23: Instance of the structure of the Value Partition ODP (more concrete than Figure 2.22). This is the most concrete structure, as three given values are used: `Tall`, `Medium`, and `Short`.

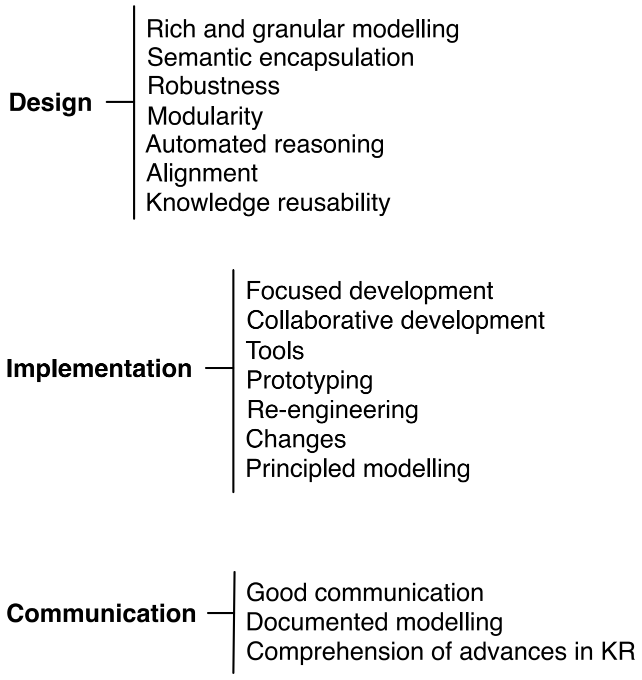


Figure 2.24: Advantages of using ODPs. The advantages are divided in three areas: design, implementation and communication.

experience in bio-ontology development, and finally other advantages have been experimentally confirmed by the use cases of Chapter 6, as it is explained in Chapter 7. The advantages are divided into three areas, as summarised in Figure 2.24: design, implementation, and communication.

Design

Rich and granular modelling: ODPs allow for a more fine-grained representation of the knowledge, as they help in exploiting the expressivity of KR languages to obtain the highest possible resolution in modelling. On the other hand, the representational needs of biological knowledge, in terms of granularity, are increasing as the knowledge that needs to be captured grows in complexity [92].

Semantic encapsulation: ODPs reduce complex semantics to a discrete abstraction with a name, which helps in finding a solution faster than without ODPs. Not only for finding solutions, encapsulation also reduces the expressivity space and hence gives the ontologist a sense of the expressive limitations of the language (*e.g.* the Nary Relationship ODP[56] indicates

[56]`http://www.gong.manchester.ac.uk/odp/html/Nary_Relationship.html`

that OWL does not support higher arity relations).

Robustness: Some ODPs help in producing bio-ontologies that are less likely to errors when modified, or make errors more explicit.

Modularity: Some ODPs help in producing bio-ontologies that are easier to modify as they are created with independent modules. In a domain like biological knowledge, where ontologies are potentially huge, this feature is especially interesting [129].

Automated reasoning: The rich axiomisation needed for a meaningful automated reasoning process is obtained with reduced effort using ODPs.

Alignment: The number of bio-ontologies is increasing, and efficient methods for aligning them are needed [129]. Although ODPs do not represent an alignment method *per se*, the consistency inherent in modelling using ODPs facilitates the semantic alignment of different ontologies.

Knowledge reusability: One of the aims of KR is to make knowledge reusable, thus to define it once and reuse it in different systems, especially in the context of the Semantic Web [3]. ODPs facilitate knowledge reusing as they are independent modelling units.

Explicit modelling: ODPs help in explicitly stating, *via* axioms, the knowledge that needs to be modelled, instead of leaving it "buried" in term labels. That allows developers to exploit other tools like automatic OWL explanations[57] for debugging bio-ontologies. Also, the decision to choose an ODP and make knowledge explicit forces the ontologists to challenge assumptions about the domain of knowledge [109].

Implementation

Focused development: ODPs reduce development time, allowing the focus of the modelling effort to be on specific (more difficult) areas of the domain.

Collaborative development: By agreeing on the use of ODPs, different developers can more efficiently work in the same ontology.

Tools: ODPs can be programmatically codified, *e.g.* facilitating the creation of tools for guiding the ontologist in the process of building an ontology. Tools can also exploit ODPs to provide easier ways of building especially

[57] `http://owl.cs.manchester.ac.uk/explanation/`

complex or difficult areas of an ontology. On the other hand, ODPs facilitate the design of the software that will interact with the ontology, when using the ontology as a driving model in an application.

Prototyping: A prototype can be built rapidly as an assembly of already existing ODPs, in the early stages of development, saving time: without ODPs the building of such prototype is more laborious.

Re-engineering: ODPs are applied in the beginning as well as during the development of an ontology, *e.g.* allowing to refactor inconsistent portions of an ontology.

Changes: Consequences of changes are more predictable and changes are more traceable using ODPs.

Principled modelling: Ontology development is still more an art than an engineering discipline, making room for "guruization" phenomena where doctrine imposes over engineering principles [18]. ODPs offer a framework for defining what is correct modelling for a given set of requirements and for assessing the modelling more precisely, helping in the process of making ontology building more an engineering discipline than a craft.

Communication

Good communication: The developers recognise and understand faster the modelling decisions, as they are made with ODPs which are thoroughly documented. Also, ODPs provide a vocabulary for discussing different representations of the same knowledge [30]. Even in the case that the application of an ODP is rejected, it challenges the assumptions of the developers and it makes it more reasonable for other developers to demand a detailed explanation from them on why the ODP should be rejected. Thus, if an ODP is presented by one of the developers instead of presenting some vague modelling, the counter arguments for rejecting it will have to be more sophisticated and therefore the discussion much richer.

Documented modelling: The design decisions made during development can be registered by tracing which ODPs were applied.

Comprehension of advances in KR: KR languages, especially OWL, are evolving fast [50]. ODPs are examples of how to use the new features of KR languages, and therefore help in grasping the advances in KR.

2.6.2.4 Methods for applying ODPs

There are different methods for applying ODPs:

Manually: The structure of the ODP can be recreated, step by step, in the target ontology, following the example from the online catalogue of ODPs.

OWL imports: ODPs can also be directly incorporated by importing them, as they are OWL small ontologies. Using OWL importing, however, the ODP can only be used "as is", because renaming entities is not possible when importing. A more direct method of incorporating the ODP and being able to rename its entities is by simply saving it locally, and starting to build the ontology around the ODP, renaming its entities as necessary.

Protégé wizards: The Protégé wizards[58] guide the modeller in the step by step application of the ODP; it is faster than the **manually** method, but a wizard should exist in the first place for the desired ODP.

OWL Macros: OWL Macros can be used to directly transform OWL ontologies in RDF/XML format [121]. This method requires XSLT knowledge, and interacting with the ontology at markup level misses all the semantics; *e.g.* queries to the reasoner cannot be done in order to retrieve a set of entities to which to apply the ODP.

Programmatic application: The OWL API or the Protégé script tab[59] can be used to codify an ODP and automatically apply it. The drawback of this method is that it demands programming knowledge from the ontologist, something not always possible. Also, programmatically codifying complex axioms is cumbersome and not flexible.

Ontology PreProcessor Language: OPPL is a scripting language for automatically modifying the axioms of OWL ontologies. OPPL can be used for applying ODPs, as in the **programmatic application** method, but differing in that the user need not know detailed programming.

[58]`http://www.co-ode.org/downloads/wizard/`

[59]`http://www.ea3888.univ-rennes1.fr/dameron/protegeScript/`

2.7 Conclusions

The main problem of current bioinformatics is the excess of computationally unmanageable information. The LSSW aims at applying KR techniques, and especially Semantic Web technologies like RDF and OWL, to solve this problem. One of the main components of the solution is the use of bio-ontologies to computationally represent biological knowledge. Many of the current bio-ontologies, however, do not exploit all the capabilities of KR languages like OWL, especially with regards to axiomatic richness and rigour. Axiomatic richness and rigour allow to exploit automated reasoning, which can be used for sophisticated knowledge management like rich querying, consistency checking, ontology maintenance and hypothesis generation.

A solution that contributes to the creation of axiomatically richer and more rigorous bio-ontologies is the use of ODPs, as ODPs, being already tested and thoroughly documented solutions to concrete modelling problems, act as guides to KR languages like OWL. The contribution of the chapter is a working definition of ODPs, comparing it to previous attempts, and a description of the advantages of the use of ODPs for bio-ontology engineering, therefore providing an answer to the research question *What are ODPs?*

Chapter 3

A Catalogue of ODPs

This chapter provides an answer to the research question *How can we obtain ODPs?*

In order for bio-ontologists to be able to efficiently explore and retrieve ODPs, an online public catalogue of ODPs[1] has been created during this research [10]. The catalogue focuses on ODPs that facilitate biological knowledge modelling. The ODPs are consistently described using a set documentation schema, facilitating the comparison of different ODPs, and hence the exploration of the catalogue.

The chapter provides all the necessary information about the catalogue, like motivation, design, implementation and usage. The catalogue itself should be explored online or in Appendix A, which is a copy of the online catalogue.

The chapter starts by describing the motivation and requirements for an ODPs public catalogue in Section 3.1. Section 3.2 describes the design of the catalogue, presenting the details of the documentation schema and the justification for the classification of ODPs. Section 3.3 describes the implementation details of the catalogue, which is based on describing each ODP in an OWL file, making it possible to exchange ODPs with their documentation attached. Section 3.4 describes how to use and contribute new ODPs to the catalogue. Finally, other catalogues of ontology engineering best practices are reviewed in Section 3.5.

3.1 Motivation and requirements

As mentioned in Chapter 1, the expressivity of a KR language like OWL allows an ontologist to create an infinite number of models, each model being an unique combination of axioms. If those models are arranged in a hypothetical expressivity space, the

[1] `http://odps.sf.net/`

ontologist can be imagined as an explorer of such a space. Points of rich axiomisation on the expressivity space are the optimums where the ontologist should tend to, and ODPs highlight such points. As the aim of this research is to facilitate the usage of KR technology to the bio-ontologists, it is assumed that, *a priori*, the ontologist is not aware of any such points. Therefore they must be collected and presented to ontologists in a centralised repository that can be used as a guide to explore the expressivity space. The repository is a "tour" of the expressivity space of the KR language, highlighting different points of the expressivity space (ODPs), which is the main motivation for an online catalogue of ODPs.

The ontologist should be able to get the greatest amount of information possible from exploring the online catalogue: the aim of ODPs is to make KR languages more usable, and hence the documentation needs to be easy to understand by someone that does not necessarily have KR expertise. Therefore, the documentation of each ODP should be organised as much as possible and separated in clear sections. Such documentation schema also facilitates the exploration of the repository by making it possible to compare different ODPs. For example the Nary relationship is described in various publications or resources in different ways (and cross referenced): in the W3C BPD[2], in [34], and in [108]. There is no way of comparing the different descriptions or comparing the Nary relationship to other ODPs using a common schema. With a centralised repository of ODPs in place, the bio-ontologists need not waste precious development time exploring best practices from different resources, with different description schemas.

The more ODPs in the repository, the more complete the exploration of the expressivity space will be. Therefore, the online catalogue needs to be easy to extend by adding new ODPs, whilst retaining consistency in the descriptions. ODPs should be added by the ontologist's community.

The requirements described above determine the design of the online catalogue, which is described in Section 3.2.

3.2 Design

In essence, the online catalogue of ODPs consists of a collection of ODPs described using a documentation schema and classified in three groups (Tables 3.2, 3.3, and 3.4). The documentation schema consists of a list of parameters or sections that should

[2]http://www.w3.org/TR/swbp-n-aryRelations/

be filled with information or diagrams, like `name`, `aim`, `structure`, *etc.* Some of the sections are optional and some are required (Table 3.1). An example of how the documentation schema is used to describe the Value Partition ODP[3] can be seen in Figure 3.1.

In order to easily navigate the online catalogue, a classification of ODPs is needed. The classification should also help the bio-ontologist in choosing the appropriate ODP. Therefore a classification based on the functionality of ODPs is used for this online catalogue, dividing the ODPs in three main groups:

Extension ODPs (Table 3.2): ODPs that by-pass the limitations of the KR language, in this case OWL.

Good Practice ODPs (Table 3.3): ODPs that should be applied to obtain a more robust, cleaner and easier to maintain ontology.

Domain Modelling ODPs (Table 3.4): ODPs that offer ways of modelling concrete requirements of the domain being represented.

Most of the ODPs applied to biological knowledge tend to belong to the domain modelling category, due to the special requirements and complexity of the biological knowledge domain [110], but they can also be found in the other categories. The different categories fulfil different requirements that bio-ontologists have when building bio-ontologies. For example if a bio-ontologists finds that the binary relations of OWL are a limitation in terms of modelling, he will ideally look for a solution in the Extension category, and find a suitable solution in the N-ary Relationship ODP.

Another important distinction is the one between ODPs that exploit automated reasoning (Dynamic ODPs) and ODPs than do not (Static ODPs). Static ODPs do exploit automated reasoning but only for querying and consistency checking; dynamic ODPs also exploit it for obtaining the desired structure, besides querying and consistency checking. For example in the Normalisation ODP[4], automated reasoning is necessary to obtain the desired structure (a taxonomy with multiple inheritance), apart from querying and consistency checking, whereas in the Sequence ODP automated reasoning is only used for querying the sequence and making sure that the sequence is consistent.

[3]`http://www.gong.manchester.ac.uk/odp/html/Value_Partition.html`
[4]`http://www.gong.manchester.ac.uk/odp/html/Normalisation.html`

Section name	**Explanation**	**Status**
Name	Unique identifier	Required
Also known as	Any other name	Optional
URL	An URL that points to an OWL file	Required
Classification	One of “Extension”, “Good practice” or “Domain Modelling”	Required
Motivation	A general situation where the ODP might be necessary	Required
Aim	The concrete objective of the ODP, the detailed solution it provides	Required
Elements	The properties, classes and instances of the ODP	Required
Structure	How the elements are combined (UML)	Required
Sample	The application of the ODP in a real ontology (UML)	Required
Implementation	Detailed procedure for applying the ODP	Required
Result	The main (desired) changes on the target ontology after application of the ODP and sometimes after automated reasoning	Required
Side effects	Any non-desired or non-obvious effects of applying the ODP	Required
Known uses	Any public ontology where the ODP has been successfully applied	Optional
Related ODPs	Any ODP that reuses this ODP	Optional
References	Any resource where the ODP was previously described	Optional
Additional information	Anything that does not fit in the other sections	Optional

Table 3.1: Documentation schema for describing each ODP. The column on the left provides the list of sections, the column in the middle a brief explanation for each section and the column on the right the status (optional or required).

Different classification schemas for ODPs have been proposed in the literature. In [17] the following categories are presented for classifying patterns in ontology engineering: Application Patterns, Architecture Patterns, Design Patterns, Semantic Patterns and Syntactic Patterns. The ODPs presented in this thesis focus on the modelling of the ontology, which makes them equivalent to Semantic Patterns, but as they have different functions there are further divided into the three categories of the online catalogue (Extension, Good Practice, Domain Modelling). Another classification is the one presented in [34], where ODPs are classified as Logical ODPs, Architectural ODPs and Content ODPs. Again, the ODPs presented would be equivalent to the idea of Content ODPs but they need further classification in order to be efficiently explored.

3.3 Implementation

The implementation of the online catalogue of ODPs is based on storing each ODP in an OWL file (Figures 3.4, 3.5, and 3.6). The semantic content of the ODP, the structure, is codified in OWL axioms, and the documentation sections are codified in annotation values. All the OWL files of the online catalogue, each of them describing an ODP, follow the same convention in their basic structure (Figure 3.2):

NAME: Value Partition.

ALSO KNOWN AS: Enumeration, if it is built using individuals instead of classes.

CLASSIFICATION: Good Practice.

MOTIVATION: Reality is full of attributes of elements. For example, a person can be defined as being short, medium or tall, and the attribute height can just get those values. Height is said to be covered or exhausted by those values; the possible heights are only those three. Biology is full of such situations: metabolism can only be anabolism or catabolism, membrane transport can only be uniport, sinport or antiport, regulation is always positive, negative, and so forth.

AIM: To model values of attributes. In this example we model biological regulation, being negative or positive. PositiveRegulationOfCellKilling, from GO, is linked to the appropriate value.

STRUCTURE: See Figure A.33.

SAMPLE: See Figure A.34.

ELEMENTS: The main elements are the classes that make up the Value Partition itself: a class for the attribute and the subclasses for the values. In this case, Regulation, Positive, and Negative, respectively. The most important relationship is the one that links each element of the knowledge domain with the values of the Value Partition. In this case, IsRegulationOfType (functional).

IMPLEMENTATION: Identify the attributes every element must be described with. For each attribute, create a class under Modifier (or the pertinent upper level distinction that it is used in the ontology). In each attribute class create a subclass for every value and make them disjoint. Create a covering axiom defining the attribute class. Create the restrictions pointing to the values of the Value Partition.

RESULT: The attributes and the elements that are described or modified by the attributes get untangled: whenever a new element enters the domain (e.g. another regulation phenomenon) it is only a matter of adding a restriction pointing to the pertinent Value Partition class. The values that can be given to a certain attribute are constrained, enforcing a better modelling.

ADDITIONAL INFORMATION: The Value Partition built with classes offers an advantage over the Enumeration (a Value Partition built with individuals): new subpartitions can be built for each of the value classes (e.g. very tall).

REFERENCES:

- `http://www.w3.org/TR/swbp-specified-values`
- `http://www.co-ode.org/resources/tutorials/bio/`

URL: `http://www.gong.manchester.ac.uk/odp/owl/Good_Practice_ODP/Value_Partition.owl`

Figure 3.1: Description of the Value Partition ODP using the documentation schema. In the case of the `structure` and `sample` sections, a diagram is shown in the online catalogue, not shown here due to space limitations.

Extension ODPs	
Exception (Dynamic)	OWL cannot represent default knowledge: this ODP can be used to simulate it, including exceptions of defaults
Nary Relationship (Static)	OWL can only represent binary relationships (*e.g.* "patient has condition temperature"): this ODP can be used to represent relations between more than two entities (*e.g.* "patient has condition increasing temperature")
Nary DataType Relationship (Static)	This ODP applies the same principle as the Nary Relationship ODP to data type values instead of entities; therefore it can be used to model composite data type values (*e.g.* a temperature in certain units and at a certain pressure)

Table 3.2: Extension ODPs in the online catalogue. The classification according to exploitation of automated reasoning is shown close to the ODP name, on the left column. On the right column a brief explanation of the utility of the ODP is provided. ODPs taken from the literature or other online resources (refer to the online catalogue for references).

Good Practice ODPs	
Entity-Quality (Static)	Model entities and their qualities using a single generic property: *e.g.* a building (entity) has a quality (generic property) of being tall (quality). The generic property is reused for other entities: *e.g.* a car has a quality of being white
Entity-Property-Quality (Static)	Model entities and their qualities using a property for each quality: *e.g.* a car (entity) has a colour (property) white (quality)
Entity-Feature-Value (Static)	Model entities and their qualities, further defining the qualities: *e.g.* a car (entity) has a colour (feature) white (value) with a brightness (further value) and a saturation (further value)
Selector (Static)	Represent qualities that can be used to select an entity, *e.g.* along a symmetry axis (right or left)
Value Partition (Static)	Represent the complete set of values that a parameter can take: *e.g.* the gender of a person can only be male or female
Defined Class Description (Dynamic)	Simulate and if-then data structure
Normalisation (Dynamic)	Use the reasoner to maintain an ontology with multiple inheritance, *i.e.* an ontology in which each class has more than one superclass
Upper Level Ontology (Static)	Represent the main types of entities of an ontology (*e.g.* processes or entities), to facilitate integration with other ontologies
Closure (Static)	Close a relationship so that it can only point to instances of a concrete class

Table 3.3: Good Practice ODPs in the online catalogue. The classification according to exploitation of automated reasoning is shown close to the ODP name, on the left column. On the right column a brief explanation of the utility of the ODP is provided. Boldfaced ODPs have been created by the author: the rest have been taken from the literature or other online resources (refer to the online catalogue for references).

Domain Modelling ODPs	
List (Dynamic)	Simulate a list data structure, so that a reasoner can compare different such structures
Adapted SEP (Static)	Implement a selective propagation of properties along a transitive property like part-of: *e.g.* an infection of the finger is an infection of the hand, but a broken finger is not a broken hand
Interactor-Role-Interaction (Static)	Represent interactions, interacting entities, and the roles of entities independently: *e.g.* an interacting agent will have different roles in different interactions
Sequence (Static)	Model phases of processes sequentially
Composite Property Chain (Dynamic)	A property chain made of combining two chains (*e.g.* "the son of the brother of my father is my cousin")

Table 3.4: Domain Modelling ODPs in the online catalogue. The classification according to exploitation of automated reasoning is shown close to the ODP name, on the left column. On the right column a brief explanation of the utility of the ODP is provided. Boldfaced ODPs have been created by the author: the rest have been taken from the literature or other online resources (refer to the online catalogue for references).

the class named `... Domain` (`...` stands for any ODP name) is a direct subclass of `owl:Thing` and it holds all the documentation in annotation values (`name`, `aim`, `implementation`, *etc.*). Everything under the class `... Domain` represents the semantics (structure) of the ODP. For example, in the case of the Value Partition ODP, the class is called `ValuePartitionDomain`. `ValuePartitionDomain` holds the documentation (Figure 3.4) in its annotation values (*e.g.* the annotation property `name` takes the value `ValuePartition`, the annotation property `classification` takes the value `Good Practice`, *etc.*). Under `ValuePartitionDomain` there is also a sample of the structure of a typical value partition (Figure 3.5): a class (`regulation`) covered by two disjoint subclasses, `positive` and `negative`.

The fact that the documentation and the semantics (the structure) of each ODP are bundled together in an OWL file allows for the exchange of ODPs with all the necessary information in a single modelling unit. This makes the communication between developers more fluid. It also allows semantics to be added and to document such addition in the same OWL file, making the creation of ODPs that conform to the documentation schema straight away: as the semantic structure is built, the documentation can be added. By using a convention and a given structure (the `... Domain` class, and a given set of annotation properties, provided by the documentation schema), new ODPs and their descriptions can be added consistently. Also, as the OWL files follow a defined convention for structure, they can be automatically parsed to generate other

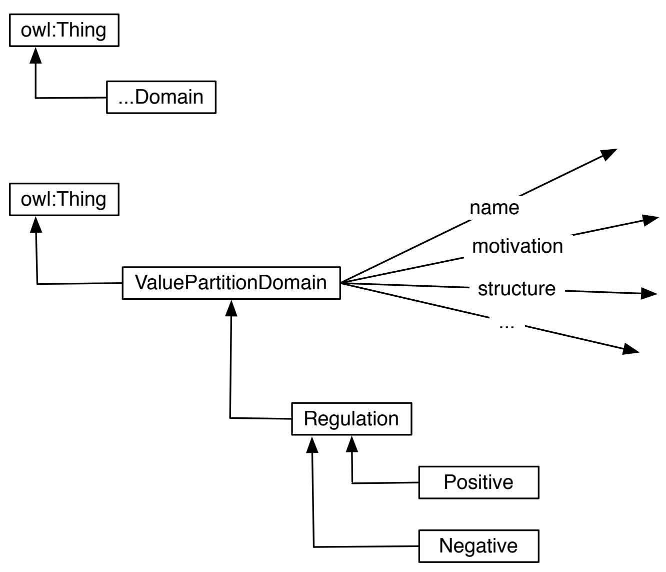


Figure 3.2: Basic structure of an OWL file that stores an ODP. On the top, the generic structure, where a class holds the documentation and the structure of the ODP. On the bottom, an instance of the generic structure, describing the Value Partition ODP. Simple arrows represent `subClassOf` axioms, named arrows annotation properties (`name`, `motivation`, *etc.*). Only `subClassOf` axioms are shown, thus the covering axiom and the disjoint axioms of the Value Partition ODP are not shown, for the sake of simplicity.

formats: the HTML[5] and LaTeX 2ε (Appendix A) versions of the catalogue were generated following such procedure. The HTML version of the catalogue entry for the Value Partition ODP, generated from the OWL file, can be seen in Figures 3.7 and 3.8.

There is not a consensus on how to graphically represent ODPs as there is for software design patterns (UML) [34]. Usually UML is also used for OWL due to tool support and familiarity, as UML is well known by programmers and ontologists alike. UML does not represent OWL semantics in an adequate manner, as, *e.g.* it is not a compact representation of OWL axioms, but OWL-native graphical representations like GrOWL (Graphical OWL) [66] are not sufficiently mature and widespread. As it can be seen in Figures 3.7 and 3.8, UML is used for graphically representing the ODPs. The UML profile chosen to represent ODPs in the online catalogue (Figure 3.3) was presented in [19].

[5]`http://odps.sf.net/`

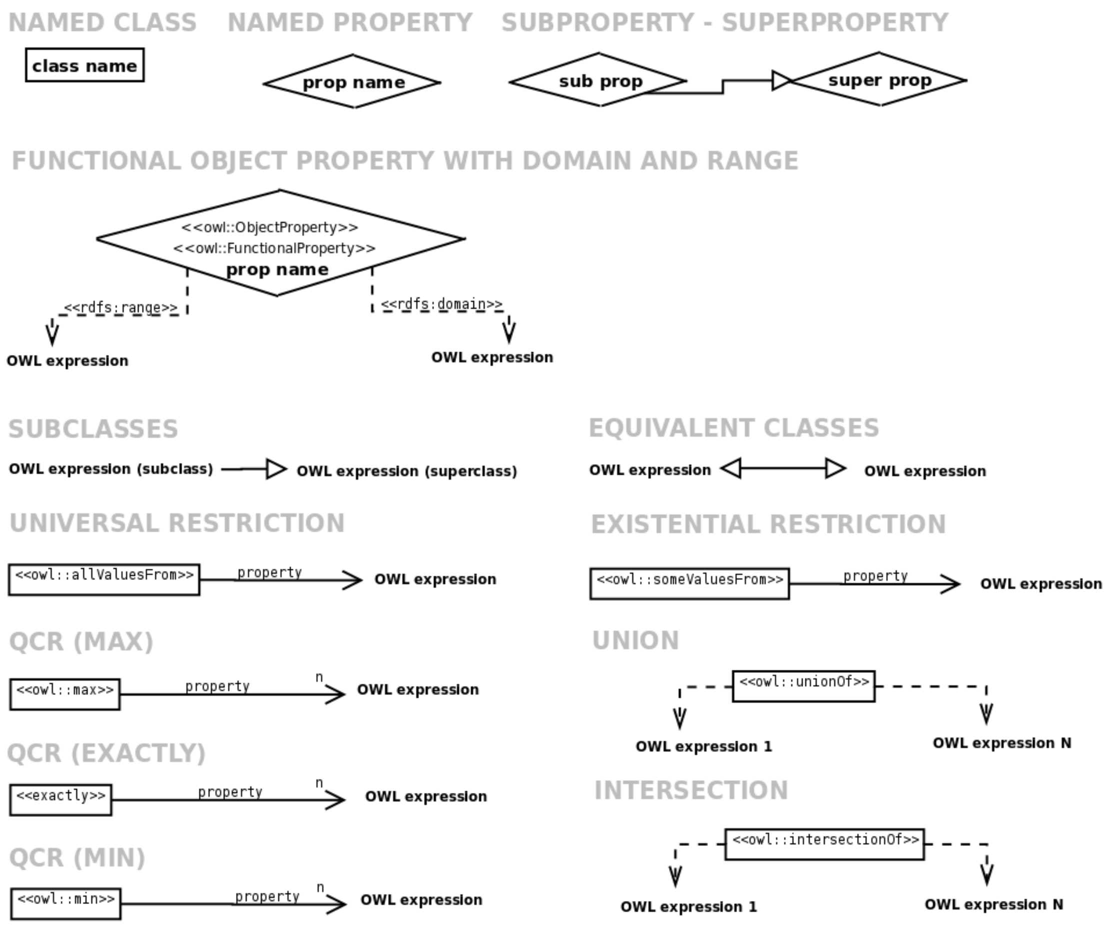


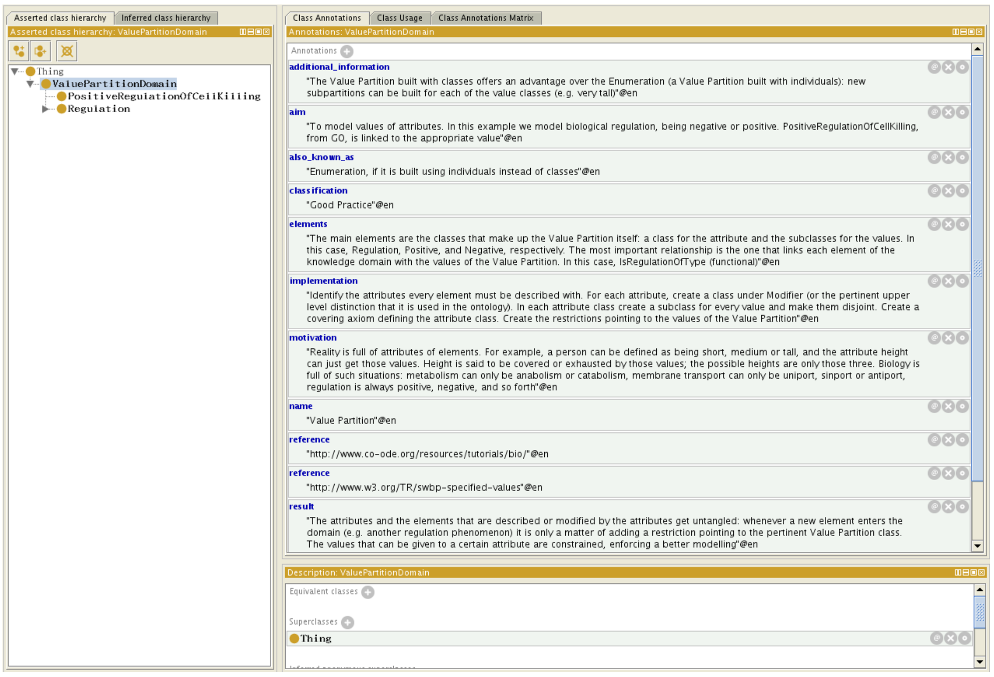
Figure 3.3: OWL to UML mapping, extended from [19]. Not all the OWL constructs are shown. An OWL expression can be a named class or an anonymous class (*e.g.* an anonymous class made by nested restrictions).

```
Class: ValuePartitionDomain
    SubClassOf: owl:Thing
    Annotations:
        sample "../img/ValuePartition_instance.png"@en,
        name "Value Partition"@en,
        structure "../img/ValuePartition_abstract.png"@en,
        additional_information "The Value Partition built with
            classes offers an advantage over the Enumeration (a Value Partition
            built with individuals): new subpartitions can be built for each of
            the value classes (e.g. very tall)"@en,
        classification "Good Practice"@en,
        also_known_as "Enumeration, if it is built using
            individuals instead of classes"@en,
        elements "The main elements are the classes that make up
            the Value Partition itself: a class for the attribute and the
            subclasses for the values. In this case, Regulation, Positive,
            and Negative, respectively. The most important relationship
            is the one that links each element of the knowledge domain with the
            values of the Value Partition. In this case, IsRegulationOfType
            (functional)"@en,
        motivation "Reality is full of attributes of elements.
            For example, a person can be defined as being short, medium or tall,
            and the attribute height can just get those values. Height is said
            to be covered or exhausted by those values; the possible heights are
            only those three. Biology is full of such situations: metabolism can
            only be anabolism or catabolism, membrane transport can only be
            uniport, sinport or antiport, regulation is always positive,
            negative, and so forth"@en,
        reference
                "http://www.w3.org/TR/swbp-specified-values"@en,
        reference
                "http://www.co-ode.org/resources/tutorials/bio/"@en,
        aim "To model values of attributes. In this example we
                model biological regulation, being negative or positive.
                PositiveRegulationOfCellKilling, from GO, is linked
                to the appropriate value"@en,
```

Figure 3.4: Fragment of the OWL file that describes the Value Partition ODP, in MOS. The class `ValuePartitionDomain` holds the documentation in the values of the annotation properties, *e.g.* `motivation`.

```
Class: ValuePartitionDomain
    SubClassOf: owl:Thing

Class: Regulation
    SubClassOf: ValuePartitionDomain
    EquivalentTo: Negative or Positive

Class: Positive
    SubClassOf: Regulation
    DisjointWith: Negative

Class: Negative
    SubClassOf: Regulation
    DisjointWith: Positive

Class: PositiveRegulationOfCellKilling
    SubClassOf: ValuePartitionDomain,
        is_regulation_of_type some Positive
```

Figure 3.5: Portion of the OWL file that describes the Value Partition ODP, in MOS. The class `ValuePartitionDomain` holds the structure of the ODP in its subclasses.



```
Thing
  ValuePartitionDomain
    PositiveRegulationOfCellKilling
    Regulation
```

additional_information
"The Value Partition built with classes offers an advantage over the Enumeration (a Value Partition built with individuals): new subpartitions can be built for each of the value classes (e.g. very tall)"@en

aim
"To model values of attributes. In this example we model biological regulation, being negative or positive. PositiveRegulationOfCellKilling, from GO, is linked to the appropriate value"@en

also_known_as
"Enumeration, if it is built using individuals instead of classes"@en

classification
"Good Practice"@en

elements
"The main elements are the classes that make up the Value Partition itself: a class for the attribute and the subclasses for the values. In this case, Regulation, Positive, and Negative, respectively. The most important relationship is the one that links each element of the knowledge domain with the values of the Value Partition. In this case, IsRegulationOfType (functional)"@en

implementation
"Identify the attributes every element must be described with. For each attribute, create a class under Modifier (or the pertinent upper level distinction that it is used in the ontology). In each attribute class create a subclass for every value and make them disjoint. Create a covering axiom defining the attribute class. Create the restrictions pointing to the values of the Value Partition"@en

motivation
"Reality is full of attributes of elements. For example, a person can be defined as being short, medium or tall, and the attribute height can just get those values. Height is said to be covered or exhausted by those values; the possible heights are only those three. Biology is full of such situations: metabolism can only be anabolism or catabolism, membrane transport can only be uniport, sinport or antiport, regulation is always positive, negative, and so forth"@en

name
"Value Partition"@en

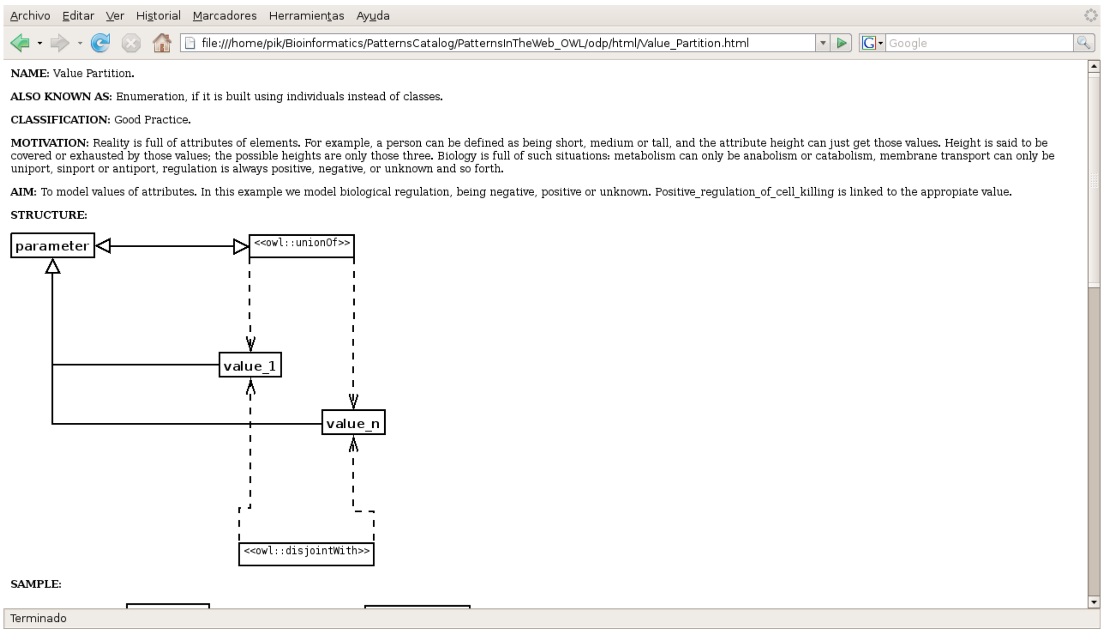
reference
"http://www.co-ode.org/resources/tutorials/bio/"@en

reference
"http://www.w3.org/TR/swbp-specified-values"@en

result
"The attributes and the elements that are described or modified by the attributes get untangled: whenever a new element enters the domain (e.g. another regulation phenomenon) it is only a matter of adding a restriction pointing to the pertinent Value Partition class. The values that can be given to a certain attribute are constrained, enforcing a better modelling"@en

Thing

Figure 3.6: OWL ontology representing the Value Partition ODP in Protégé. On the left pane, the whole structure of the ontology is shown. As the class `ValuePartitionDomain` is selected, the documentation is shown in the form of annotation values.



Archivo Editar Ver Historial Marcadores Herramientas Ayuda
file:///home/pik/Bioinformatics/PatternsCatalog/PatternsInTheWeb_OWL/odp/html/Value_Partition.html
Google

NAME: Value Partition.

ALSO KNOWN AS: Enumeration, if it is built using individuals instead of classes.

CLASSIFICATION: Good Practice.

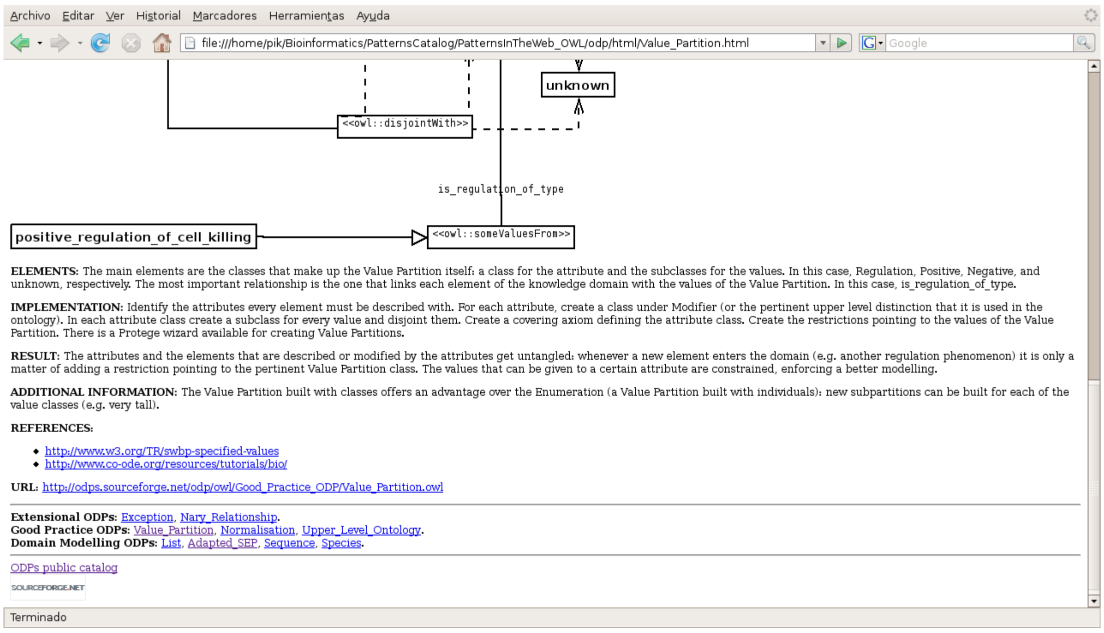
MOTIVATION: Reality is full of attributes of elements. For example, a person can be defined as being short, medium or tall, and the attribute height can just get those values. Height is said to be covered or exhausted by those values; the possible heights are only those three. Biology is full of such situations: metabolism can only be anabolism or catabolism, membrane transport can only be uniport, sinport or antiport, regulation is always positive, negative, or unknown and so forth.

AIM: To model values of attributes. In this example we model biological regulation, being negative, positive or unknown. Positive_regulation_of_cell_killing is linked to the appropiate value.

STRUCTURE:

SAMPLE:

Figure 3.7: A fragment of the HTML entry for the Value Partition ODP. The information shown on each section is generated by a script from the OWL file from Figures 3.4, 3.5, and 3.6. The following sections are shown: `name`, `also known as`, `classification`, `motivation`, `aim`, `structure`, `sample`.



Archivo Editar Ver Historial Marcadores Herramientas Ayuda
file:///home/pik/Bioinformatics/PatternsCatalog/PatternsInTheWeb_OWL/odp/html/Value_Partition.html
Google
unknown
<<owl::disjointWith>>
is_regulation_of_type
positive_regulation_of_cell_killing
<<owl::someValuesFrom>>

ELEMENTS: The main elements are the classes that make up the Value Partition itself: a class for the attribute and the subclasses for the values. In this case, Regulation, Positive, Negative, and unknown, respectively. The most important relationship is the one that links each element of the knowledge domain with the values of the Value Partition. In this case, is_regulation_of_type.

IMPLEMENTATION: Identify the attributes every element must be described with. For each attribute, create a class under Modifier (or the pertinent upper level distinction that it is used in the ontology). In each attribute class create a subclass for every value and disjoint them. Create a covering axiom defining the attribute class. Create the restrictions pointing to the values of the Value Partition. There is a Protege wizard available for creating Value Partitions.

RESULT: The attributes and the elements that are described or modified by the attributes get untangled: whenever a new element enters the domain (e.g. another regulation phenomenon) it is only a matter of adding a restriction pointing to the pertinent Value Partition class. The values that can be given to a certain attribute are constrained, enforcing a better modelling.

ADDITIONAL INFORMATION: The Value Partition built with classes offers an advantage over the Enumeration (a Value Partition built with individuals): new subpartitions can be built for each of the value classes (e.g. very tall).

REFERENCES:

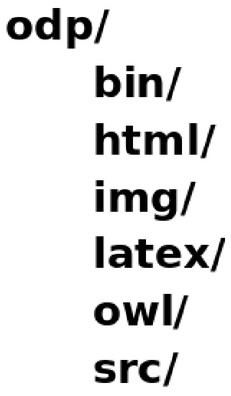
- http://www.w3.org/TR/swbp-specified-values
- http://www.co-ode.org/resources/tutorials/bio/

URL: http://odps.sourceforge.net/odp/owl/Good_Practice_ODP/Value_Partition.owl

Extensional ODPs: Exception, Nary_Relationship.
Good Practice ODPs: Value_Partition, Normalisation, Upper_Level_Ontology.
Domain Modelling ODPs: List, Adapted_SEP, Sequence, Species.

ODPs public catalog

Figure 3.8: Continuation from Figure 3.7. Shown sections: `elements`, `implementation`, `result`, `additional information`, `references`, and `url`.



```
odp/
    bin/
    html/
    img/
    latex/
    owl/
    src/
```

Figure 3.9: Directory structure of the catalogue bundle. The main directory, `odp`, contains the directories `bin`, `html`, `img`, `latex`, `owl`, and `src`.

3.4 Using the catalogue

The online catalogue should be explored online by the ontologists, and ODPs used as a guide in the ontology building process. The online catalogue can be downloaded as a bundle[6], consisting of a root directory, `odp`, that contains the following directories (Figure 3.9): `bin` (the software used to generate the HTML and the LaTeX 2ε versions of the online catalogue), `html` (the HTML version of the online catalogue), `img` (the images for the `structure` and `sample` sections of the online catalogue), `latex` (the LaTeX 2ε version of the online catalogue), `owl` (an OWL file for each ODP), `src` (the source code of the software available in `bin`).

The HTML and LaTeX 2ε versions of the online catalogue can be locally generated using the software available in the `bin` directory: the `OWL2HTML` script for the HTML version and the `OWL2LATEX` script for the LaTeX 2ε version[7]. Both scripts are Java software, make use of the OWL API, and are licensed under the GPL[8] so they can be adapted by anyone for concrete needs.

The online catalogue describes ODPs collected or adapted from the literature and the internet, and ODPs created by the author. Anyone can add an ODP to the online catalogue, if it is approved after proposing it in the mailing list or forums[9]. There are different criteria for judging the adequacy of an ODP to be included in the online catalogue:

[6]`http://sourceforge.net/project/showfiles.php?group_id=206639`
[7]Appendix A was generated using OWL2LATEX and only minor corrections were made.
[8]`http://www.gnu.org/copyleft/gpl.html`
[9]`http://sourceforge.net/projects/odps/`

Parsimonious: In terms of axiomisation, the candidate ODP should be as simple as possible, without unnecessary complexity. A complex ODP is not necessarily more useful, as demonstrated by ODPs like the Value Partition and the Closure. Simple patterns in software engineering have been widely used, like the Singleton pattern[10], one of the simplest conceivable patterns.

Clear origin: The ODP need not be new, or invented by the author, it can be added to the online catalogue as a new description of an already existing ODP, *e.g.* to describe it consistently and hence make it comparable to other ODPs. In that case due references should be available.

Clear benefits: It should be clear what are the ODP's contributions to the modelling. In other words, what is the difference between using the ODP and not using it? Why is the resulting new ontology axiomatically richer, more rigorous or more maintainable?

Clear costs: It should be clear what are the costs and side effects of using the ODP. For example there are ODPs that make a heavy use of automated reasoning (*e.g.* the List ODP[11]) or ODPs that complicate the subsumption hierarchy (*e.g.* the Exception ODP[12]).

Trivial ODPs: The ODP can be as simple as necessary, but trivial ODPs like language primitives should not be accepted. For example, a "subclass ODP" does not give any new information to the ontologists, and hence it should not be accepted in the online catalogue. However, an ODP describing how to use a subclass axiom in combination with an equivalent axiom can be considered a proper ODP, as is the case with the Defined Class Description ODP[13].

Required information: Clear, complete and concise information should be available for the required sections of the documentation schema, as described in Table 3.1, *e.g.* `motivation`, `aim`, *etc.* Information for the optional sections should also be available, but it is not mandatory.

Tested: The proposed ODP should have been used in different ontologies and thoroughly tested, especially with regards to automated reasoning.

[10]`http://en.wikipedia.org/wiki/Singleton_pattern`
[11]`http://www.gong.manchester.ac.uk/odp/html/List.html`
[12]`http://www.gong.manchester.ac.uk/odp/html/Exception.html`
[13]`http://www.gong.manchester.ac.uk/odp/html/DefinedClass_Description.html`

3.5 Related resources

There are other ontology best practices repositories available online. For example, some ODPs are collected on the web site of the W3C BPD[14], but they are not described following a set documentation schema and are mixed with best practices of other domains different from ontology engineering.

Another online catalogue of ODPs is maintained as part of the NeOn project[15]. The NeOn catalogue[16] differs from this one in various aspects. In the NeOn catalogue the classification of ODPs is different from this one (as mentioned in Section 2.6.2.1, there are more types of design patterns, related in a hierarchical model [34]), and the mechanism of distribution is not based on OWL files. Another difference is that, as explained in Section 2.6.2.1, the NeOn catalogue is focused around Content Ontology Design Patterns, which differ from the notion of ODPs used in this work, as they differentiate between the logical and content layers, a distinction lacking in the ODPs of this work, for the sake of usability. Finally, the target knowledge domain for this catalogue is biological knowledge, whereas in the NeOn catalogue it is any domain of knowledge.

3.6 Conclusions

The research question *How can we obtain ODPs?* has been answered by presenting an online catalogue of thoroughly described ODPs for the biological knowledge domain. ODPs in the online catalogue are described in a consistent manner, using a set documentation schema for all the ODPs, to ease exploration and comparison of different ODPs. The online catalogue is implemented through OWL files: each ODP is described in an OWL file, unifying the semantics and the documentation of the ODP in a single unit that can be shared between developers. The online catalogue is designed so as to allow added ODPs to conform to the documentation schema. Therefore the contribution of the chapter is a public online resource that can be used to explore and retrieve ODPs, offering a guide of the expressivity space of OWL.

[14]`http://www.w3.org/2001/sw/BestPractices/`
[15]`http://ontologydesignpatterns.org`
[16]At the time of this writing, the NeOn catalogue was empty.

Chapter 4

Ontology PreProcessor Language (OPPL)

Each ODP can be regarded as a modelling unit, thus a concrete set of axioms, documented and identified by a unique name, to be applied in a concrete ontology. Therefore there is a need for an automatic method for encapsulating and then applying such modelling units in ontologies, as expressed by the research question *How can we apply ODPs?*

This chapter provides an answer to that question by presenting the Ontology PreProcessor Language (OPPL). OPPL is a scripting language for programmatically codifying changes in the axioms of OWL ontologies: an OPPL script is a set of changes to be performed in an OWL ontology. Therefore any ODP can be defined in an OPPL script and applied in an OWL ontology using the OPPL interpreter.

The chapter is organised as follows. Section 4.1 provides the introduction to OPPL: its origin (Section 4.1.1), a definition and general properties (Section 4.1.2), a comparison between OPPL 1 and OPPL 2 (Section 4.1.3), and an overview of related work (Section 4.1.4). Section 4.2 describes the detailed usage of OPPL for applying ODPs in OWL ontologies.

4.1 Description of OPPL

4.1.1 Origin

The idea of OPPL arose as an improvement of the implementation of the Gene Ontology Next Generation (GONG) pipeline[1] [73].

In the GONG pipeline, the labels of the OWL classes from GO are syntactically dissected, and new axioms are added to those classes according to such dissection (Figure 4.1). The dissection is defined in a regular expression that has associated axioms. For example, the regular expression `(.+?) (development)` has the associated axiom `development and acts-on some <1>` (`<1>` refers to the first group matched with the regular expression, `(.+?)`). When such a regular expression is interpreted by the GONG pipeline software[2], *e.g.* if the GO class with the label `neuron development` is matched, the following axiom will be added to the class: `development and acts-on some neuron`. Therefore, when the pipeline is executed, `neuron development` will be redefined as a type of development that acts on neurons. Using the GONG procedure, bio-ontologies can be axiomatically enriched and efficient automated reasoning applied without much effort, as only some regular expressions and their associated axioms need to be defined.

The original implementation of the GONG pipeline needed a special OWL ontology, the GONG ontology, to be defined by the user, with one class per regular expression and associated axioms. Configuring the pipeline was cumbersome, as the user had to follow strict guidelines on how to populate the GONG ontology with regular expressions and their associated axioms. Also, the complexity of the axioms associated to each regular expression was limited: *e.g.* new entities could not be added, axioms could not be removed, and only equivalent restrictions with simple fillers (not nested restrictions) were processed by the GONG pipeline. OPPL was developed to overcome such limitations: OPPL can be used to define axiomatic changes of arbitrary complexity to be performed in classes retrieved by processing their annotation values with regular expressions (Figure 4.2).

Later it became obvious that such a language could be used for any kind of automated manipulation of OWL ontologies, so OPPL was extended by adding the capability of performing DL queries, full MOS support[3] and a new implementation presented

[1]`http://www.gong.manchester.ac.uk/`

[2]`http://www.gong.manchester.ac.uk/bin/bong_2007.tar.gz`

[3]A MOS syntax parser was already available in the OWL API (see Section 4.1.4), which made implementing OPPL a matter of adding a thin layer on top of an already existing tool.

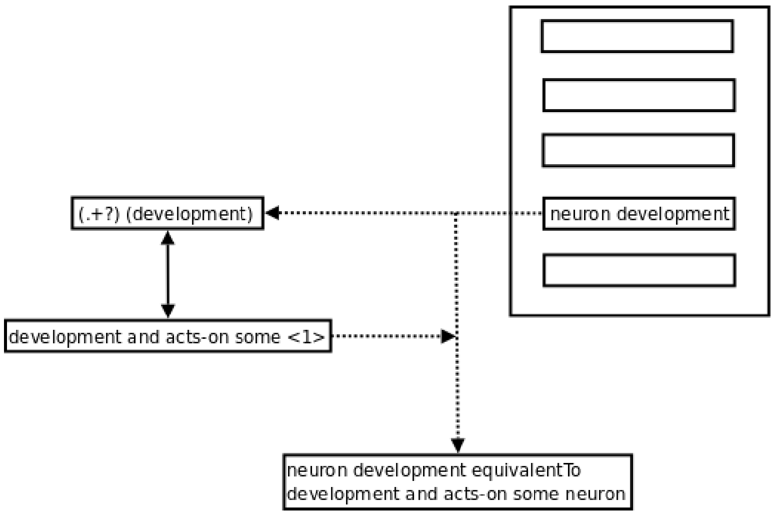

(.+?) (development)
neuron development
development and acts-on some <1>
neuron development equivalentTo
development and acts-on some neuron

Figure 4.1: GONG process example. An ontology (top right) is processed by the GONG pipeline. One of the regular expressions already defined (`(.+?) (development)`) matches one of the terms from the ontology (`neuron development`), and that term is enriched with the axioms associated with the regular expression (`development and acts-on some <1>`). The axioms are added to `neuron development`, which becomes equivalent to `development and acts-on some neuron`.

in [29].

As OPPL allows any kind of automatic manipulation of the axioms of an OWL ontology, it can be used to apply ODPs in OWL ontologies. Furthermore, OPPL fulfils the following requirements of ODPs' representation and application:

- Select entities, add and remove axioms.
- Shared and intuitive syntax for representing ODPs.
- Encapsulation of ODPs in modelling units.
- Automated application of modelling units.
- Modelling units can be shared for automatic application.
- Replicable application of modelling units.
- Explicit and documented application of modelling units.
- Flexible application of modelling units.

```
SELECT label "(.+?) (development)";ADD equivalentTo development and acts-on some
<1>;
```

Figure 4.2: Abstraction of the GONG process example with OPPL. This OPPL script has the same effect on an ontology as the item of the GONG pipeline described in Figure 4.1. However, executing this script needs no special setup apart from writing it and passing it to the OPPL interpreter. It is also much more flexible than the procedure of Figure 4.1, as any change can be codified in an OPPL script.

4.1.2 Definition and general properties

OPPL is a scripting language for interacting with OWL ontologies on an axiomatic level. Such interaction is based on querying for entities, and adding or removing axioms to or from the retrieved entities. Entities and axioms can also be added or removed without querying. OPPL is a Domain Specific Language (DSL), OWL being the domain, as it allows the expression of solutions in the same level of abstraction as the problem (OWL axioms) [117]. OPPL is also a declarative language, as it states the axioms that should be added or removed, without control flow structures.

The OPPL syntax is equivalent to MOS, with extra terms (`ADD`, `REMOVE`, `SELECT`) and tokens (semicolon,`<n>`). The semicolon delimits each OPPL statement. Two basic OPPL syntax examples are shown in Figures 4.3 and 4.4. For example, a query can be formulated in OPPL to retrieve an entity with the reasoner, as shown in Figure 4.3. Statements can be combined, so actions can be attached to retrieved entities, as shown in Figure 4.4.

OPPL is focused on entities. For example, if the user wants to add a named class as a subclass of another named class, *e.g.* `X subClassOf Y`, he needs to first add the class, then remove the axiom `subClassOf Thing`, and then add the desired subclass axiom (Figure 4.16). This is like that because OPPL, being entity-centric, automatically translates the instruction `ADD Class: X` into `X subClassOf Thing`.

OPPL can work in asserted or inferred mode. In asserted mode, only the explicitly stated (asserted) axioms are considered, since the reasoner is not used for exploiting OWL semantics. However, in asserted mode, the user can work with inconsistent OWL ontologies. For example asserted mode is useful to add an axiom to the direct subclasses of a given class, without adding the axiom to all the inferred subclasses or the descendants classes (the transitivity of the subsumption relationship is not taken into account). In inferred mode, as the reasoner is used, full semantics are available, thus the asserted and the inferred axioms are accessed by the OPPL interpreter, but the ontology must be consistent.

```
SELECT subClassOf has_part some (nucleus or mitochondrion);
```

Figure 4.3: Retrieving an entity with OPPL. The term `SELECT` is followed by a MOS expression and the whole statement is delimited by a semicolon. This statement will retrieve any entity that has the `has_part some nucleus or mitochondrion` anonymous class as a superclass (necessary condition).

```
SELECT subClassOf has_part some (nucleus or mitochondrion);ADD equivalentTo
part_of some tissue;
```

Figure 4.4: Retrieving an entity and adding axioms to it with OPPL. The `SELECT` statement is followed by an action statement that will add the anonymous class `part_of some tissue` as an equivalent class (necessary and sufficient condition) to the retrieved entities.

For example, the query `SELECT subClassOf part_of some cell` will deliver different results depending on the use of asserted or inferred mode. In asserted mode, only the direct parts of the cell will be delivered (*e.g.* `nucleus`), as the transitivity of `part_of` is not exploited. However, in inferred mode, the indirect parts of the cell will also be delivered. For example `nucleolus` is only asserted to be a part of the nucleus, but as the nucleus is part of the cell, and `part_of` is transitive, `nucleolus` is also considered by the reasoner to be a part of the cell and therefore delivered.

4.1.3 OPPL 1 and OPPL 2

There are currently two versions of OPPL: OPPL 1 and OPPL 2. OPPL 1 was developed by the author [30, 29, 10], and it is implemented as a standalone command line application[4]; OPPL scripts must be written in flat files[5] and passed to the interpreter via command line arguments (Figures 4.5, 4.6, and 4.7). OPPL 2 is being developed by Luigi Iannone, also from the Bio Health Informatics Group, in collaboration with the author, and it is implemented as a Protégé plugin[6] (Figures 4.8, 4.9) [62, 61]. The differences are summarised in Table 4.1.

The syntactic differences between OPPL 1 and OPPL 2 reflect the semantic differences between both versions (Figures 4.10 and 4.11). In OPPL 2, any entity of the query can be a variable, with the constraint that a variable can only be bound by a named entity and not by a complex expression (Figure 4.12). Also, OPPL 2 is strongly

[4]`http://oppl.sourceforge.net/`

[5]Comments, in OPPL 1 flat files, are any line that starts with the # symbol, and are skipped by the OPPL interpreter. Such convention allows for Perl style syntax highlighting in source editors to edit OPPL scripts.

[6]`http://www.cs.man.ac.uk/~iannonel/oppl/`

OPPL 1	**OPPL 2**
Single variable	Multiple variables
Centred on OWL entities	Centred on OWL axioms
No expression evaluation	Expression evaluation before execution
Command line	Protégé plugin
Annotation values processing (*e.g.* regular expressions)	No annotation values processing

Table 4.1: Comparison between OPPL 1 and OPPL 2. The main difference between OPPL 1 and OPPL 2 is that OPPL 1 cannot process expressions with multiple variables.

typed, that is, the types of the variables must be defined in each query: *e.g.* in the expression `?x:CLASS`, any entity that binds the variable `?x` must be an OWL class. The fact that in OPPL 2 only typed single entities can be bound to variables makes the querying computable, as it can be reduced to a list of DL queries and their combinations. The lack of variables makes OPPL 1 rather "local" to the ontology that is being modified, as the user needs some knowledge of the entities that are referenced in the axioms of the ontology before interacting with it. OPPL 2 can work with completely abstract structures, making it more flexible.

Another difference is that OPPL 2 more closely resembles OWL 2 semantics, as its syntax is centred on axioms rather than entities. Therefore, coming back to the example on Section 4.1.2, the axiom `X subClassOf Y` can be directly added, without having to add any further instructions, as is the case in OPPL 1.

OPPL 2 can also work in asserted mode or inferred mode. Both OPPL 1 and OPPL 2 can work with the Pellet and FaCT++ reasoners, and OPPL 1 can additionally work with any DIG[7] compliant reasoner.

Finally, OPPL 2 is further different to OPPL 1 in that it includes an evaluation step: whenever an OPPL script is introduced by the user, it is evaluated (Figure 4.8) and the axioms that will be affected are displayed (Figure 4.9), so the user can choose to execute the script or not.

On the other hand, OPPL 1 is able to process annotations, which is of special importance when dealing with bio-ontologies, as, *e.g.* GO presents structured annotation values [30, 73]. The annotation processing is implemented via regular expressions: a query is built using a regular expression, and entities whose annotation values match the regular expression are retrieved. Not only the entities are retrieved, the content of the matched string is also made available for further actions (Figure 4.13).

[7]`http://dig.sourceforge.net/`

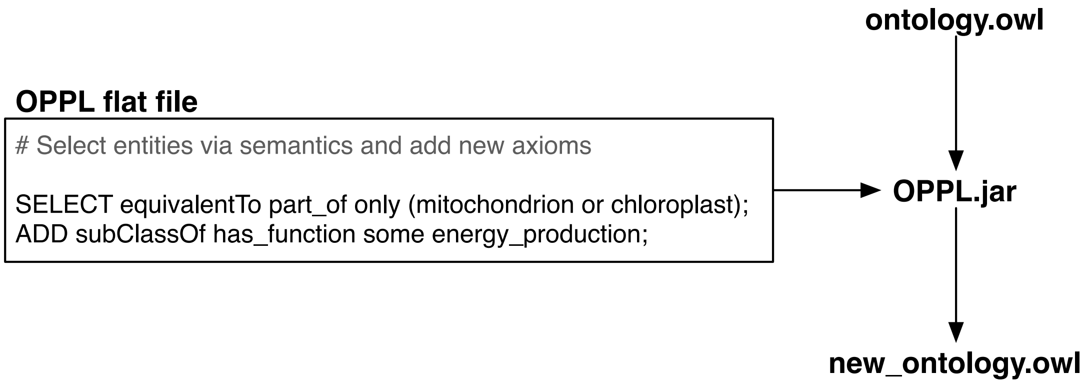

OPPL flat file
# Select entities via semantics and add new axioms
SELECT equivalentTo part_of only (mitochondrion or chloroplast);
ADD subClassOf has_function some energy_production;
ontology.owl
OPPL.jar
new_ontology.owl

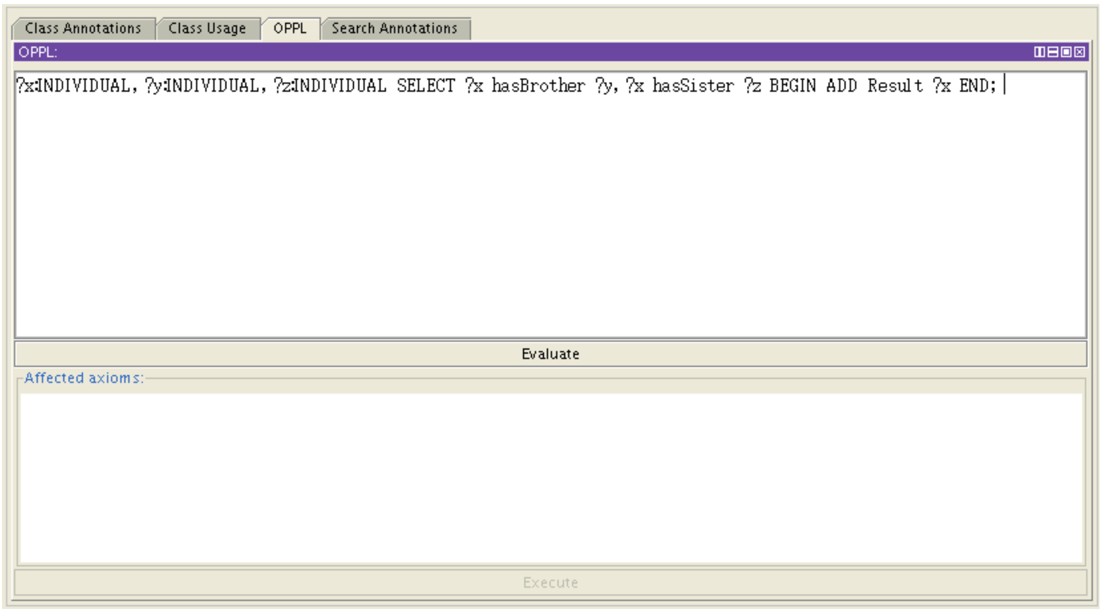
Figure 4.5: OPPL 1 interpreter. An OPPL flat file is passed to the interpreter (`OPPL.jar`), together with the target ontology (`ontology.owl`). After executing the instructions from the flat file and changing the target ontology, a new ontology is generated, with the changes in it (`new_ontology.owl`).

```
# Create the properties: immediately_precedes and immediately_preceded_by
# NOTE: reasoning didn't work in queries about the inverse's superproperty,
# we don't know why

ADD ObjectProperty: immediately_preceded_by;ADD functional;
ADD subPropertyOf preceded_by;

ADD ObjectProperty: immediately_precedes;ADD functional;
ADD subPropertyOf precedes;

# General cell cycle: G1 -> S -> G2 -> M

SELECT Class: CCO_P0000313;ADD subClassOf immediately_precedes some
CCO_P0000315;

SELECT Class: CCO_P0000315;ADD subClassOf immediately_preceded_by some
CCO_P0000313;ADD subClassOf immediately_precedes some CCO_P0000314;

SELECT Class: CCO_P0000314;ADD subClassOf immediately_preceded_by some
CCO_P0000315;ADD subClassOf immediately_precedes some CCO_P0000039;

SELECT Class: CCO_P0000039;ADD subClassOf immediately_preceded_by some
CCO_P0000314;
```

Figure 4.6: OPPL flat file used for applying the Sequence ODP in CCO. This flat file, when executed, adds some axioms related to the Sequence ODP[9]. `CCO_P0000313`: G1, `CCO_P0000315`: S, `CCO_P0000314`: G2, `CCO_P0000039`, M.

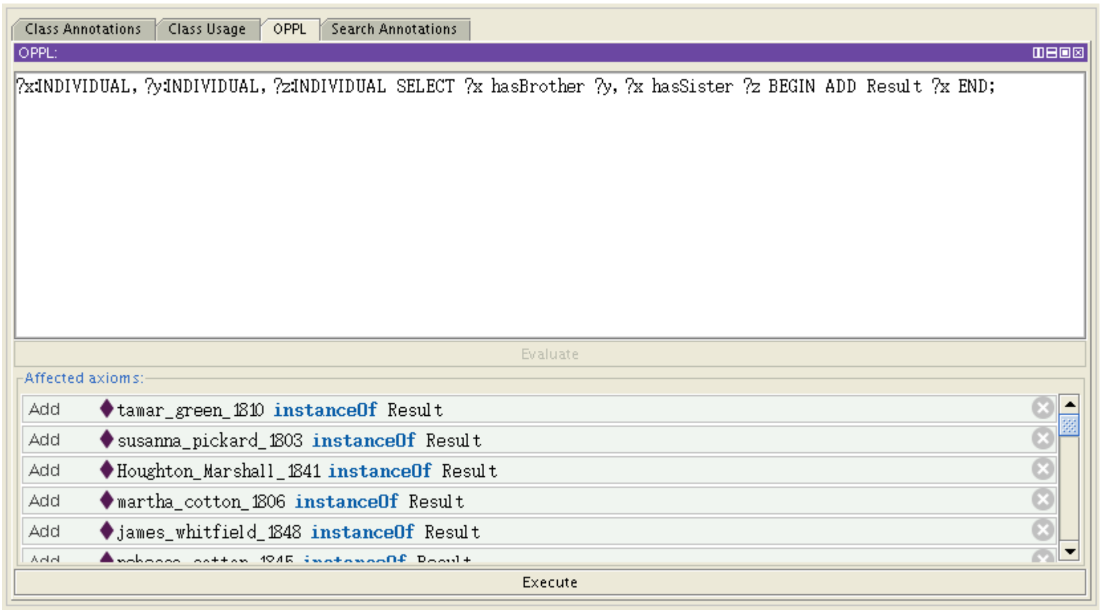
```
# Query 14: Proteins located in nucleus and cytoplasm: travelling?

ADD Class: query_14;ADD subClassOf query;REMOVE subClassOf Thing;
ADD comment "Proteins located in nucleus and cytoplasm: travelling?";
ADD oppl_query "CCO_U0000005 and (located_in some (part_of some CCO_C0000323
or CCO_C0000323)) and (located_in some (part_of some CCO_C0000252 or
CCO_C0000252))";

SELECT subClassOf CCO_U0000005 and (located_in some (part_of some CCO_C0000323
or CCO_C0000323)) and (located_in some (part_of some CCO_C0000252 or
CCO_C0000252));ADD subClassOf query_14;
```

Figure 4.7: OPPL flat file used for performing a DL query against CCO. This flat file, when executed, creates a class named `query_14`. The next step queries the reasoner for subclasses of the anonymous class `CCO_U0000005 and (located_in some (part_of some CCO_C0000323 or CCO_C0000323)) and (located_in some (part_of some CCO_C0000252 or CCO_C0000252))`, and adds them as a subclass of `query_14`, providing the answers to the query. `CCO_U0000005`: protein, `CCO_C0000323`: cytoplasm, `CCO_C0000252`: nucleus.



```
?x:INDIVIDUAL, ?y:INDIVIDUAL, ?z:INDIVIDUAL SELECT ?x hasBrother ?y, ?x hasSister ?z BEGIN ADD Result ?x END;
```

Figure 4.8: OPPL 2 Protégé plugin. The user enters an OPPL script in the top pane. In this case, the script selects any individual that has a `hasSister` and a `hasBrother` relationships and makes the individual an instance of the class `Result`.



```
?x:INDIVIDUAL, ?y:INDIVIDUAL, ?z:INDIVIDUAL SELECT ?x hasBrother ?y, ?x hasSister ?z BEGIN ADD Result ?x END;
```

```
Add   tamar_green_1810 instanceOf Result
Add   susanna_pickard_1803 instanceOf Result
Add   Houghton_Marshall_1841 instanceOf Result
Add   martha_cotton_1806 instanceOf Result
Add   james_whitfield_1848 instanceOf Result
Add   rebecca_cotton_1845 instanceOf Result
```

Figure 4.9: OPPL 2 Protégé plugin: evaluate and execute. The plugin evaluates the OPPL script entered by the user, and shows the axioms that will be affected. If the user is satisfied, the script can be executed.

```
SELECT subClassOf participates_in some (mitosis or meiosis);ADD equivalentTo
has_function some regulation_of_cell_cycle;
```

Figure 4.10: OPPL 1 syntax example. The script selects any class that is a subclass of the anonymous class `participates_in some (mitosis or meiosis)`, and adds the anonymous class `has_function some regulation_of_cell_cycle` to it as an equivalent class.

```
?x:CLASS SELECT ?x subClassOf participates_in some (mitosis or meiosis)
BEGIN ADD ?x equivalentTo has_function some regulation_of_cell_cycle END;
```

Figure 4.11: OPPL 2 syntax example. The result of executing this script is equivalent to the result of executing the script shown on Figure 4.10. The main difference with OPPL 1 is the use of a strongly typed variable, `?x:CLASS`. The OPPL 2 syntax also includes other tokens like `BEGIN` and `END`.

```
?x:CLASS, ?y:CLASS SELECT ?x subClassOf participates_in some (mitosis or ?y)
BEGIN ADD ?x equivalentTo has_function some ?y END;
```

Figure 4.12: OPPL 2 syntax example with multiple variables. This script will select any named class (`?x`) that is a subclass of the anonymous class `participates_in some (mitosis or ?y)`, being `?y` any named class. Then, it will add the condition `has_function some ?y` to the retrieved classes (`?x`), substituting `?y` with any named classes that match the query.

```
SELECT label "(.+$) (.+$) of (development)";ADD subClassOf regulation_type some
<1>;ADD subClassOf <2>;ADD subClassOf acts_on_process <3>;
```

Figure 4.13: Annotation processing with OPPL 1. The OPPL interpreter assumes that the tokens from the matched string also exist in the ontology as entity URI fragments or `rdfs:label` values; if they are not, the OPPL statement is simply ignored. For example, if the class with the label `positive regulation of development` is matched, OPPL will try to find classes in the ontology with the following strings as URI fragments or `rdfs:label` values: `positive`, `regulation`, `development` (corresponding to <1>, <2>, and <3> respectively).

4.1.4 Related work

OPPL is made of two parts: querying and actions (adding or removing axioms). They are both independent, *i.e.* OPPL can be used solely for querying or solely for actions. However, the combination of both (defining actions to be performed in entities retrieved by querying) is what makes OPPL useful, as such functionality allows the application of ODPs in different parts of an ontology.

In such combination, the closest to OPPL is the DL-safe fragment of SWRL (Semantic Web Rule Language) [76]. OPPL differs from it in that OPPL allows the removal of axioms[10], includes annotation querying and it can add axioms without a condition being fulfilled.

OPPL can also be compared in the querying and actions combination with the OWL Macros presented in [121], but they are different because OWL Macros cannot remove axioms (OPPL can remove and add axioms), and OPPL works by exploiting the semantics of OWL, not just the asserted axioms. OPPL and the OWL Macros also differ in the expertise required: XML/RDF in the case of the OWL Macros and MOS in the case of OPPL. MOS has been proposed as an OWL 2 working draft[11], and it is shared by many developers and ontology editors like Protégé and TopBraid composer. Therefore, it is likely that bio-ontologists are more used to MOS than RDF/XML, as performing OWL modelling using such editors already implies some knowledge of MOS, and MOS is one of the most used human-friendly OWL syntaxes.

On the querying side, SPARQL-DL [99] can also be used to query OWL ontologies but OPPL offers a greater expressivity, as expressions of any complexity can be used to query the reasoner.

On the actions side, OPPL can be regarded as a reduced and abstracted version of

[10]The `REMOVE` feature of OPPL makes it non-monotonic with regards to automated reasoning: the order of OPPL `REMOVE` statements should be taken into account when executing them [61].

[11]`http://www.w3.org/TR/owl2-manchester-syntax/`

```
OWLClass a = dataFactory.getOWLClass("A");
OWLClass b = dataFactory.getOWLClass("B");
OWLClass c = dataFactory.getOWLClass("C");
Set<OWLClass> set = new HashSet<OWLClass>(2);
set.add(b);
set.add(c);
OWLDescription intersection = dataFactory.createOWLObjectIntersection(set);
OWLAxiom axiom = dataFactory.getOWLSubClassAxiom(a,intersection);
ontologyManager.applyChange(new AddAxiom(ontology,axiom));
```

Figure 4.14: Java code for adding a subclass axiom using the OWL API. This code consists of 9 Java instructions, and some of the instructions present further complexity, requiring knowledge about classes like `ontologyManager` or `dataFactory`.

```
SELECT Class A;ADD subClassOf B and C;
```

Figure 4.15: OPPL script for adding a subclass axiom. This OPPL script is equivalent to the Java code shown in Figure 4.14, but only one statement is needed and the complexity of such a statement is directly proportional to the complexity of the axiom, apart from the `SELECT` and `ADD` instructions.

the OWL API that works with a further abstraction level, that is, axioms instead of programmatic procedures. This means that the ontologist can automate many actions, which is usually done with the OWL API, without knowing programming[12]. The complexity of such OPPL actions is directly proportional to the complexity of the axioms, regardless of the underlying programming complexity. For example, using the OWL API, adding the axiom `A subClassOf C and B` would be done using the code shown in Figure 4.14 (example taken from [61]). The same action can be performed in OPPL in a much shorter instruction (Figure 4.15), and without having to know about the OWL API structure (*e.g.* the *Command* design pattern [59]) or how to use concrete OWL API elements like `ontologyManager` or `dataFactory`.

4.2 Using OPPL

This section describes how OPPL can be used with concrete examples, and the advantages of such functionalities. The main functionality of OPPL is to apply ODPs, but it can be used to automatically apply any modelling. The advantages of applying ODPs with OPPL can be extrapolated to any other modelling applied with OPPL. OPPL can also be used to automate actions in pipelines that process or create OWL ontologies.

[12]The OWL API offers a wider range of functionalities, but the target users of this research, bio-ontologists, seldom use them.

The automated application of ODPs is possible because each ODP can be codified in an OPPL script and applied at will (Figure 4.17). Therefore, OPPL offers the possibility of codifying ODPs as modelling units that can be executed off-the-shelf. Such application is replicable because each developer need only take the script from another developer and apply it, making modelling consistent with regards to different developers or different stages of development.

The modelling process also becomes consistent in different parts of the same ontology. The procedure of applying ODPs in different parts of an ontology via queries is especially useful, as adding complex modelling manually in different parts of an ontology can be tedious and error prone. For example, OPPL was used to apply the Entity-Quality ODP[13] in GO[14], by querying GO using annotation values, as described in the use case of Section 6.2.3. When the script (Figure 4.16) was applied in GO, 24 classes out of *circa* 20,000 were captured and the ODP was correctly applied on them. Recreating such procedure manually would be at least a few hours of work, and the result would most probably be incomplete.

The above procedure for applying the Entity-Quality ODP in GO relies on the ontology having structured annotation values, which is common in bio-ontologies and especially in GO, but it could be that such an approach is not appropriate for other ontologies. However, even in the case in which the ODP is applied in too many entities (the regular expression matches too many annotation values), it is preferable to remove some axioms from those entities, which can be easily traced in OPPL logs, instead of applying the modelling manually. Having to select concrete parts of an ontology is a common problem to any automated modifying procedure, and OPPL solves it by providing the possibility of querying for annotation values, DL queries, asserted simple queries, or selecting entities by name (URI fragment).

The usage of OPPL and ODPs means that the modelling process becomes explicit, as the modelling steps are written in OPPL scripts, making modelling traceable, and documented, as comments can be added in the scripts about the rationale behind each modelling decision[15].

[13]`http://www.gong.manchester.ac.uk/odp/html/Entity_Quality.html`

[14]It was applied in CCO, but CCO includes GO.

[15]OWL 2 allows the annotation of axioms and therefore annotation of modelling as it is possible in an OPPL script, but OPPL comments are more compact as one comment refers to many axioms in the same script. In the same way that OPPL concentrates modelling that later, when applied, will be scattered in different parts of the ontology, it concentrates comments, and therefore offers an advantage over using only OWL annotations. In fact, OWL annotations can also be added using OPPL statements, so OPPL comments and OWL annotations can be used at the same time, exploiting the benefits of both.

However, OPPL expresses modelling by splitting it in procedures, and therefore different parts of the ODP are added in different OPPL statements. Thus OPPL does not naturally render the structure of the ODP, it does so implicitly in different OPPL statements that, when executed, will result in the whole ODP being recreated in the ontology. Languages like XSLT do offer the possibility of applying OWL Macros that more naturally (statically) render ODPs [121], however such renderisation poses the problem of having to know XSLT: as mentioned, MOS is more likely to be understood by a bio-ontologist working with OWL.

OPPL makes modelling flexible, as complex modelling can be tried out and dismissed or accepted by simply executing an OPPL script. Doing modelling manually in an ontology editor and then doing "undo"-s is inefficient and error prone, as plenty of independent steps must be cancelled. Modifying a script is more efficient than modifying an ontology, if complex modelling has been applied: *e.g.* the ontologist can delete, out of a script with 25 instructions executed sequentially, the instructions from 10 to 20 and reapply the script, which is difficult to do in an ontology editor where the 25 steps have been done manually. Also, different modelling alternatives (*e.g.* different ODPs for modelling modifiers [30]) can be easily tested, or prototypes built quickly to decide on their benefit for the ontology in early stages of the development. Different ontology versions can be aligned with OPPL scripts, making it possible to keep a history of changes or simply experiment with different development branches of the same ontology.

Applying ODPs in OWL ontologies with OPPL can be done using the following methods (Figure 4.18):

1. **Adding complete structures (OPPL 1 and OPPL 2):** A complete structure can be built from scratch, with named entities, by applying an already defined OPPL script (Figure 4.19). This means that the entities should be renamed by the ontologist in the OPPL script, if the script was provided by another developer. Such a renaming procedure is more efficient than renaming entities of an imported mini-ontology, as all the entities are concentrated in a compact file. Also, many bio-ontologies follow the practice of separating the class id and its name (provided in a `rdfs:label` value) which means that entities can be renamed by further OPPL mapping statements.

2. **Selecting concrete entities and adding axioms to them (OPPL 1 and OPPL 2):** Entities can be selected and complete ODPs or fragments of ODPs can be

applied on them. The selection can be done using different criteria to retrieve entities:

(a) **Retrieve by annotations (OPPL 1):** A regular expression is defined, like `(.+?) regulation of (.+?)`, to be matched against the annotation values of the entities of the ontology, along a given annotation property (usually `rdfs:label`) (Figure 4.20). When a match happens (*e.g.* `positive regulation of cell killing`), the entity is stored for further modification and the content of the annotation value is also stored, so it can be used for creating axioms, via the `<n>` keyword (`n` can be any number). The `<n>` keyword is a pointer to the groups of the annotation value, so `<1>` would point to the first group, namely `positive`. OPPL tries to resolve such a group against the ontology, in the reasonable expectation that an entity with the name (URI fragment) or label `positive` will be referenced; if such reference does not exist, the interpreter simply skips the OPPL statement.

(b) **Retrieve by semantics (OPPL 1 and OPPL 2):** An OWL expression of arbitrary complexity is defined and the reasoner is queried for named entities that are related to such an anonymous class (*e.g.* as a subclass or equivalent class) (Figure 4.21).

3. **Selecting abstract structures and adding axioms to them (OPPL 2):** The fact that OPPL 2 allows for variables in any position of the expression makes it possible to define structures in abstract terms, without named entities (Figure 4.22). This approach is more general than that of OPPL 1, and OPPL 2 can also be used for retrieving by semantics using only named entities (*i.e.* OPPL 2 includes OPPL 1, except the annotation processing functionality, absent in OPPL 2).

Besides applying ODPs, OPPL can also be used for axiomatically enriching or cleansing an ontology in an automatic way, *e.g.* if the ontology is periodically downloaded from an outside resource. For example, CCO is enriched using OPPL as a part of a bigger pipeline (Figure 4.6). OPPL is also useful for automating the manipulation (queries, removing or adding axioms) of ontologies that are too big to be manipulated with GUI based tools like Protégé. Again, in the case of CCO, OPPL is used for defining and executing queries (Figure 4.7): the OPPL script is executed in the background

```
######### Applying the Entity-Quality ODP in CCO #########

# Quality values

ADD Class: modifier;

ADD ObjectProperty: has_quality;

ADD Class: position;ADD subClassOf modifier;REMOVE subClassOf Thing;

ADD Class: apical;ADD subClassOf position;REMOVE subClassOf Thing;

ADD Class: basal;ADD subClassOf position;ADD disjointWith apical;
REMOVE subClassOf Thing;

# constrain the quality values to the entities (CCO_C0001882 = cell part)

SELECT Class: position;ADD equivalentTo apical or basal;ADD subClassOf
inv (has_quality) only CCO_C0001882;

# not having a position is legal

SELECT Class: CCO_C0001882;ADD subClassOf has_quality max 1 position;

# In order to apply the ODP in different places of the ontology, we need
# a general condition that will catch different target classes (doing it
# by hand would be tedious, inefficient and would betray the aim of ODPs).
# We will define a regular expression "(basal|apical) (.+?)": <1> refers
# to the first group from the string that matches the regular expression

SELECT label "(basal|apical) (.+?)";ADD subClassOf has_quality exactly 1 <1>;
```

Figure 4.16: An extract of an OPPL flat file for applying the Entity-Quality ODP in GO. Image taken from [30].

for a long time and the ontologist checks the result when finished, something inefficient with a GUI based tool.

4.3 Conclusions

The contribution of this chapter is OPPL, a scripting language that offers the possibility of programmatically changing the axioms of an OWL ontology, and hence apply ODPs on it. Using OPPL, ODPs are stored in OPPL scripts as modelling units, to be applied at the ontologist's will. OPPL fulfils the following requirements of ODPs' application:

Select entities, add and remove axioms: The most basic operation for applying ODPs is to be able to select entities, and add or remove axioms.

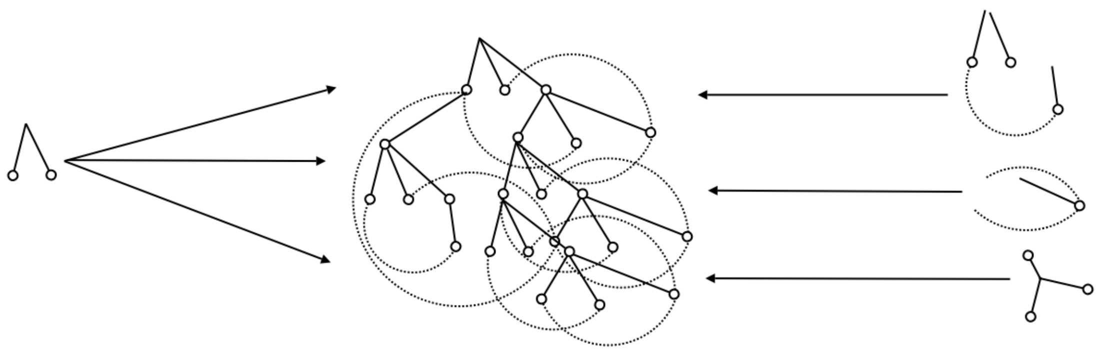


Figure 4.17: Using OPPL to apply ODPs in an OWL ontology. An OWL ontology is shown in the middle. On the left, the same ODP is applied in different parts of the ontology. On the right, different ODPs are applied in different parts of the ontology. Circles represent classes, solid lines `subClassOf` axioms, dotted lines other axioms (*e.g.* restrictions), and arrows represent the application of an ODP in the ontology.

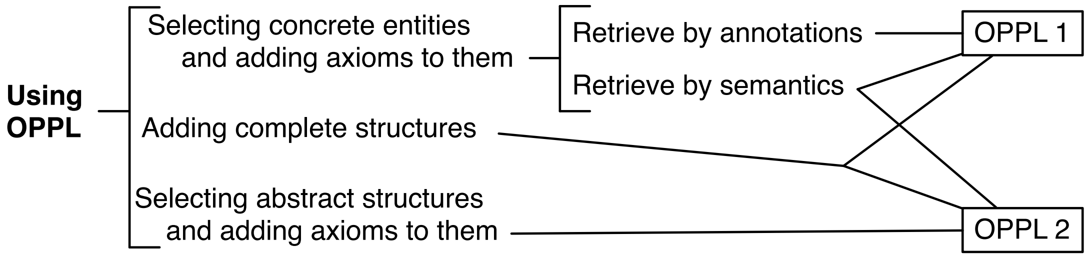

Using OPPL
Selecting concrete entities and adding axioms to them
Retrieve by annotations
Retrieve by semantics
Adding complete structures
Selecting abstract structures and adding axioms to them
OPPL 1
OPPL 2

Figure 4.18: Using OPPL: methods for applying ODPs. On the left, different ways of applying ODPs are described. On the right, the different versions of OPPL are shown in boxes. Some actions can be performed by both versions, and other actions only by OPPL 1 or OPPL 2.

```
ADD Class: Regulation;ADD Class: Positive;ADD subClassOf Regulation;
ADD Class: Negative;ADD subClassOf Regulation;ADD disjointWith Positive;
SELECT Class: Regulation;ADD equivalentTo Positive or Negative;
```

Figure 4.19: Adding the complete structure of the Value Partition ODP. The structure is added in a procedural manner. The entities can be later renamed by the ontologist, but the structure of the ODP remains on the ontology.

```
SELECT label "(.+?) regulation of (.+?)";ADD subClassOf regulationType some <1>;
```

Figure 4.20: Adding a fragment of the Value Partition ODP by retrieving entities via their annotations. Any entity with the `rdfs:label` value matching the regular expression `(.+?) regulation of (.+?)` will be retrieved, and the anonymous class `regulationType some <1>` added as a superclass to it, after resolving `<1>`. For example, if the class with the `rdfs:label` value `positive regulation of cell development` is retrieved, `regulationType some positive` will be added to it as a superclass.

```
SELECT subClassOf PositiveRegulation and acts_on only CellKilling;ADD subClassOf
regulationType some Positive;
```

Figure 4.21: Adding a fragment of the Value Partition ODP by retrieving entities via their axioms. The reasoner is queried for the subclasses of the anonymous class `PositiveRegulation and acts_on only CellKilling`, and the anonymous class `regulationType some Positive` will be added to all the retrieved classes as a superclass.

```
?x:CLASS, ?y:CLASS SELECT ?x subClassOf PositiveRegulation and acts_on only ?y;
BEGIN ADD ?x subClassOf (regulationType some Positive) and (acts_on some ?y)
END;
```

Figure 4.22: Adding a fragment of the Closure ODP by retrieving an abstract structure. The reasoner is queried for any subclass (`?x`) of the anonymous class `PositiveRegulation and acts_on only ?y`, being `?y` any named class that matches the expression. The anonymous class `(regulationType some Positive) and (acts_on some ?y)` is added as a superclass to the retrieved classes after resolving `?y`, effectively closing the relationship `acts_on`.

Shared and intuitive syntax for representing ODPs: MOS is a widely used and intuitive syntax for representing OWL, therefore providing an ideal syntax for representing ODPs for OWL ontologies.

Encapsulation of ODPs in modelling units: ODPs can be codified in OPPL scripts, creating self-contained modelling units.

Automated application of modelling units: Each ODP, being codified in an OPPL script, can be executed against an OWL ontology and the axiomisation described in the ODP transferred to the ontology, in an automatic manner.

Modelling units can be shared for automatic application: The scripts can be shared by the developers and executed at will.

Replicable application of modelling units The scripts can be consistently applied in different parts of the ontology, on different development stages, in different branches or versions of the ontology, or by different developers.

Explicit and documented application of modelling units: By using the scripts for applying ODPs, the modelling steps become explicit and traceable, and documentation about the modelling can be added as comments on such scripts.

Flexible application of modelling units: By using OPPL scripts, experimentation with new modelling units can be done more efficiently, as any step of the modelling (even intermediate steps) can be cancelled.

Therefore, the chapter has provided an answer to the research question *How can we apply ODPs?*

Chapter 5

Evaluation framework

The central problem that this thesis tackles is the lack of rigour and axiomatic richness of many current bio-ontologies, proposing the use of ODPs as a means for creating richer and more rigorous bio-ontologies. In order to test the benefit of the use of ODPs in bio-ontology engineering, ODPs need to be applied in real bio-ontologies and the results of such application evaluated. This chapter describes an evaluation framework for assessing the change in bio-ontology quality as a result of applying ODPs (ontology quality), the quality of the applied ODPs (ODP quality) and how each ODP affects the process of building bio-ontologies (ontology engineering). Therefore, this chapter answers the research questions *How can we assess ODP quality?*, *How can we assess the impact of ODPs in bio-ontology engineering?*, and *How can we assess the change of quality of bio-ontologies as a result of applying ODPs?*

The chapter is organised in two parts. Section 5.1 provides an introduction to the evaluation framework, comparing the ontology quality evaluation with related literature. The remaining of the chapter provides a detailed overview of the framework itself, in Section 5.2 (ODP quality), Section 5.3 (ontology engineering) and Section 5.4 (ontology quality).

5.1 Introduction

The evaluation of the results from this work lies in three areas: ODP quality, ontology engineering, and ontology quality. The whole framework is shown in Figure 5.1.

ODP quality is a combination of the features and possible deficiencies of a given ODP, in comparison to other ODPs. Ontology engineering analyses how a given ODP

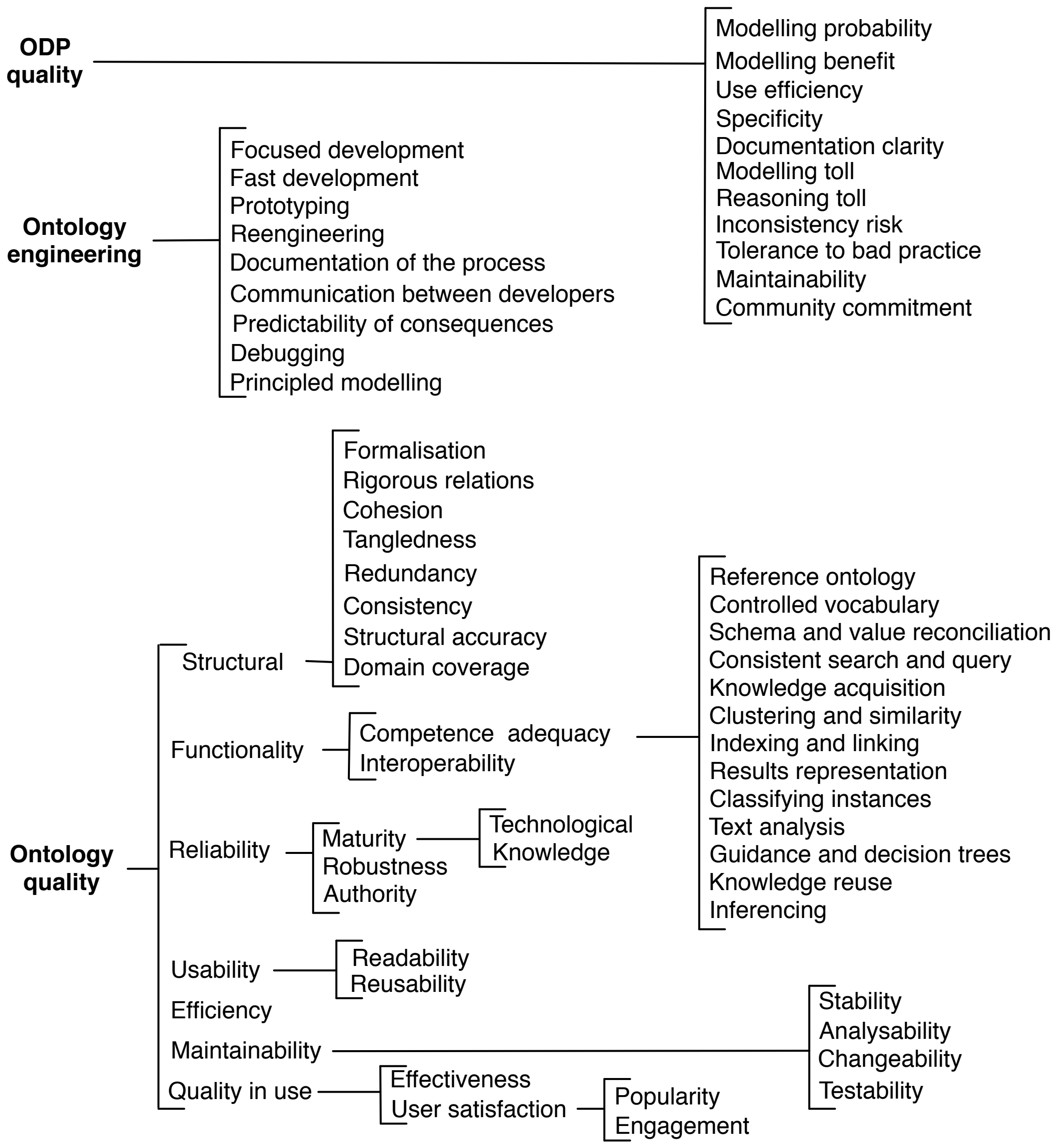


Figure 5.1: Evaluation framework. The evaluation framework is divided into three areas: ODP quality, ontology engineering and ontology quality. Each area is further divided into different criteria. Each criterion is assigned a value.

affects the process of building an ontology, also in comparison to other ODPs. Ontology quality evaluates the improvement of a bio-ontology's quality after ODPs have been applied, as defined by the ISO 9126 standard. Therefore ODP quality and ontology engineering focus on ODPs, whereas ontology quality focuses on ontologies.

Ontology evaluation is an open problem, and established solutions have not been agreed. There are different ontology evaluation methods, focused on different aspects of ontologies, but none is widely accepted, especially in the case of OWL DL [122]. For example, there are evaluation methods that focus on formal correctness, like Ontoclean [52, 120], which analyses the correctness of a taxonomy following some philosophical principles like "rigidity". Ontoclean is not appropriate for evaluating the impact of ODPs, as they rarely modify the modelling principles behind the taxonomic structure[1].

Other methods have been developed to choose adequate ontologies via ranking [4, 113, 114]. These methods are not adequate as they do not assess the impact of ODPs thoroughly, they simply point to which is the best ontology for a given task from a group of ontologies, especially in the context of the Semantic Web.

In terms of ontology quality, one of the most thorough evaluation methods is the one described in [36]. However, it includes the use of ODPs as a sign of good ontology quality. Thus, the very idea we are trying to test in this work, namely that the use of ODPs improves ontology quality, is already an assumption in such an evaluation method.

In general, most of the ontology quality evaluation methods, even though they include assessment criteria for structure, they lack assessment criteria for other aspects like functionality or maintainability. There are many aspects of ontologies that need to be evaluated and different evaluation strategies focus upon different aspects with greater or lesser success. It is unlikely that one strategy will fulfil the need to evaluate multiple aspects: fitness for purpose; rigour and axiomatic richness; domain correctness; ontological formality; *etc.*

The ISO 9126 standard for software quality evaluation includes aspects like functionality or maintainability, and therefore it was adapted to evaluate ontology quality [32]. We assume that an ontology can be considered to be equivalent to a software artefact, as the ISO 9126 standard defines software in a broad sense, *e.g.* including

[1]This does not mean that Ontoclean is not a valid modelling principle, as it can be used *e.g.* for choosing the primitive axes in the Normalisation ODP. Ontoclean can also guide the developers in the implementation of the Upper Level Ontology ODP.

documentation and executables. Therefore such definition can be extended to ontologies, as they can be regarded as software artefacts that can be queried for exploiting knowledge contained in them. Also, using practices from software engineering in ontology engineering has already been demonstrated to be useful [69, 21, 48, 39, 38]. We added one ontology-specific criterion, structure, to complete the framework, as detailed in Section 5.4.

The ISO 9126 is an internationally established standard that provides a systematic framework that has been used previously *e.g.* for evaluating e-Learning systems [22]. The ISO 9126 standard does not attempt to provide detailed metrics of every aspect of software quality; rather, it provides a comprehensive model that users and developers alike can use as a common language when assessing software quality, and agree on a concrete quality level. Therefore the output of the application of the ISO 9126 standard standard is not a numerical or absolute value, and it depends on the community that is applying the ISO 9126 standard. Hence, the ISO 9126 standard fits with the scenario of this thesis, where we are comparing an ontology after and before ODPs application: it is a relative evaluation, not an absolute evaluation that requires a concrete result.

In each of the three areas (ODP quality, ontology engineering and ontology quality), different criteria are defined, and one out of three values (1, 3 or 5) is given to each criterion (the criteria are described in detail in Sections 5.2, 5.3 and Section 5.4). A higher numerical value means a higher quality, thus a bigger benefit for the ontology or the ontology building process. In the case of ODP quality and ontology engineering areas, the score is given to the ODP, and in the case of ontology quality the score is given to an ontology. The numbers are not absolute values, as they are used to compare different ODPs (ODP quality and ontology engineering) or an ontology after and before ODP application (ontology quality). Using the score values, a radar graph is created for each ODP and the ontology on which the ODP has been applied. The usage of radar graphs provides an intuitive idea of which are the best ODPs for a given task, in which way they affect the ontology engineering process, and how ontology quality changes.

A completely rigorous and thorough evaluation is not feasible as the lack of evaluation methodologies in ontology building, due to the field's immaturity, permeates the three areas. There has been some work in ontology quality evaluation (reviewed above), but none in ODP quality or ontology engineering evaluation. Therefore, subjectivity is inevitable, especially when assigning the scores to the ODPs and ontologies in each evaluation criterion. Instead of aiming at a completely rigorous evaluation, we intend to provide the elements that the user can use to make an informed decision with regards to which ODPs to use. Also, we provide examples of how such elements can be used to assign values to ODPs and ontologies, and we provide a rationale (and hence orientation for potential users) for such an assignment.

5.2 ODP quality

Modelling probability: As mentioned in Chapter 1, ODPs can be seen as "beacons" on a expressivity space composed of an infinite number of models, each model being a unique combination of axioms. Thus, ODPs "attract" the modeller to a concrete model in the expressivity space. An usefulness measure is to ask with what probability would the modeller arrive at such a concrete model in the expressivity space without the aid of the ODP. The modelling probability also depends on the KR expertise of the modeller, thus, the same ODP will have different modelling probabilities for different modellers, but a general tendency of modelling probability can be established. For example the Closure ODP would have a high modelling probability (low score): it is highly probable for a modeller to realise that, in order to have a closure, a universal and existential restriction should be combined. On the other hand, the List ODP would have a low modelling probability (high score), as it is unlikely that a modeller would end up building a List ODP structure spontaneously, due to its complexity. Lowest modelling probability = 5.

Modelling benefit: To what degree does the ODP represent the source knowledge –the idea that needs to be represented in the ontology– in a concrete model? For example, the Closure ODP is highly fit for purpose (high modelling benefit): there is only one way, in OWL, to represent the idea of closure along a restriction without using a functional property. However, the Entity-Quality ODP is an example of the opposite situation: there are at least another two ODPs that can represent the same idea with different axioms, and, hence, different features and problems (low modelling benefit). An ideal ODP would have a low modelling probability and a high modelling benefit. Highest modelling benefit = 5.

Use efficiency: Once applied, how efficient is the exploitation of the ODP? For example, adding a new primitive class in a normalised ontology, normalised by the Normalisation ODP, is relatively easy (high use efficiency) compared to creating a new List structure using the List ODP (low use efficiency). That is because adding a new List structure is much more complex. Highest use efficiency = 5.

Specificity: There are ODPs that are generic (*e.g.* Closure ODP) and there are other ODPs that are more specific (*e.g.* Species ODP). The specificity reflects the applicability of the ODP in different domains: the more specific the ODP, the fewer domains in which it can be applied. Least specific = 5.

Documentation clarity: How thoroughly is the ODP documented, and how clear is such documentation. For example, the more optional sections of the online catalogue present content, the more useful the documentation will be. Also, the ODP should be documented using a language suitable for non-experts in KR. The documentation should be clear enough for the user to realise the benefits and problems related to the given ODP, and in which situation should the ODP be used. Clearest documentation = 5.

Modelling toll: What are the axiomatic consequences of using the ODP? There are ODPs that impose a structure that can have more serious consequences for the overall modelling, in terms of general structure of the ontology, querying, maintenance, *etc.* For example, the Upper Level Ontology ODP determines the whole structure of the ontology, whereas the Value Partition ODP is a self-contained modelling unit, affecting only a small portion of the target ontology. In another example, adding the List ODP adds a lot of axioms, adding the Closure ODP does not. Lowest modelling toll = 5.

Reasoning toll: How does automated reasoning performance decrease by adding the ODP to the target ontology? Especially in the case of dynamic ODPs, there are ODPs that demand a lot of automated reasoning resources, because of their complex axiomisation. For example in the case of the List ODP the reasoning toll is high, whereas in the case of the Value Partition ODP it is not. If such demanding ODPs are repeated in different parts of the ontology or they have a lot of entities (*e.g.* a long list), the automated reasoning performance can considerably decrease. Even though, strictly speaking, the axiomisation of an ontology always affects automated reasoning performance, there are situations in which there is no significant difference between one ontology and another. For example, in an ontology with no defined classes or disjoints and only existential restrictions with simple fillers (like the public OWL version of GO), it is likely that the addition of further classes (following the same structure) will not affect automated reasoning in a way that can be perceived by the user. However, even though the automated reasoning performance does not significantly decrease, it could be that the modelling adds additional complexity (*e.g.* if many classes are made subclasses of a given class), and hence the difference between modelling toll and reasoning toll. Lowest reasoning toll = 5.

Inconsistency risk: What is the probability of creating an inconsistency in the ontology by using the ODP? For example ODPs with richer axiomisation or a lot of disjoints are more likely to be misused and create an inconsistency. Lowest inconsistency risk = 5.

Tolerance to bad practice: There are ODPs that are more robust against bad modification or bad use. For example the Nary Relationship ODP is more robust than the Exception ODP, as adding a new relationship to a Nary Relationship is less dangerous than adding a new relationship to the application of the Exception ODP. However, it should be noted that, from the engineering point of view, it is preferable to have explicit incorrect axioms than implicit incorrect axioms, *e.g.* in labels, as they highlight problems in the knowledge domain: it is preferable to fail early with a lot of minor problems than to fail late with a few big problems. Highest tolerance = 5.

Maintainability: How difficult is it to maintain the model resulting from the application of the ODP? In general, the more complex the axiomisation, the more difficult the maintenance of the ODP in the ontology. However, as mentioned in Chapter 4, by applying ODPs with OPPL their maintenance is less laborious and more simple than it is manually, as it is not necessary to cancel modelling steps (simply comment out OPPL statements regardless of their position in the OPPL script). Highest maintainability = 5.

Community commitment: Different developers usually have different ideas about how to conceptually model the same portion of the knowledge domain. Therefore they must all reach a consensus, and a commitment to that consensus, regarding the structure of the ontology. However, there are ODPs that affect the ontology more deeply than others, therefore needing a bigger consensus. Other ODPs affect the ontology more “lightly”, hence not being important if the consensus is not complete. For example the Upper Level Ontology ODP needs a total commitment from the community, as it deeply affects the whole modelling, whereas the Value Partition ODP does not: an Upper Level Ontology is much more controversial than a Value Partition, and hence a Value Partition ODP can be applied with a smaller consensus compared to an Upper Level Ontology ODP. This criterion is related to the modelling toll, but they are not the same: the community commitment expresses the conceptual or philosophical agreement needed, whereas modelling toll refers to pure axiomatic costs. In some cases, *e.g.* the Upper Level Ontology ODP, both criteria will measure the same commitment. Lowest community commitment necessary = 5.

5.3 Ontology engineering

Focused development: ODPs, to be beneficial, should allow the ontologists to focus on the hard and important problems of the modelling process. Therefore ODPs should be able to be used to "mass produce" large portions of the target ontology, saving time. Most focused development = 5.

Fast development: ODPs should make it possible for the modeller to reach an optimum model in the expressivity space faster than without using ODPs, as ODPs save time that does not need to be invested in deciding how to build that optimum model. Fastest development = 5.

Prototyping: ODPs should enable a prototype ontology to be quickly built as a collection of ODPs. A prototype ontology can be used for testing different queries, or for highlighting potential maintenance issues. Quickest prototyping = 5.

Reengineering: ODPs should allow the modelling to be reused in different development stages. For example, the Value Partition ODP can be recycled for different values. Maximum reuse = 5.

Documentation of the process: By using ODPs the modelling decisions become explicit and traceable, as ODPs represent concrete models on the expressivity space with a unique name, and are codified in application units (OPPL scripts). Some ODPs abstract more complex modelling than other ODPs, and therefore they offer a more informative documentation of the modelling process. Also, the process should be further documented using OPPL comments (*e.g.* the observed consequences of using an ODP). Most informative documentation = 5.

Communication between developers: ODPs should help in creating a more fluid communication between developers, as they offer concrete models to discuss different modelling decisions [30]. They should also make understanding of an ontology a faster process, as the observer can recognise the different ODPs. There are ODPs that synthesise more complex modelling in easier to communicate notions, being more informative. Most informative communication = 5.

Predictability of consequences: By using ODPs the consequences of applying a concrete modelling should be clear, as ODPs have been previously tested and such consequences documented. Most predictable consequences = 5.

Debugging: By using ODPs it should be clearer what the errors of the ontology are, and how they can be traced, therefore making debugging an ontology a more efficient process. This is due to the fact that ODPs are concrete and documented fragments of modelling that are more predictable than modelling "on the wild". ODPs have well known properties, and therefore it is more likely to deduce what fragment of the ontology is the culprit. Most efficient debugging = 5.

Principled modelling: ODPs should constrain the ways in which axioms can be added to the ontology, resulting in more meaningful modelling. For example, when using the Sequence ODP it is clear where and how a new item should be added to the sequence, and therefore modelling is not done "on the wild". This practice should result in more maintainable ontologies and ontologies where collaborative work is more efficient. Most principled modelling = 5.

5.4 Ontology quality (ISO 9126 standard)

Structural: This is the only category that it is not included in the ISO 9126 standard. However, in ontology quality it is important to consider the structure of the artefact, as the structure considerably affects the rest of the criteria, especially the functionality.

Formalisation: A formal model with precise semantics allows for a precise interpretation of the statements made in the ontology. Therefore automated reasoning can be applied, and automated reasoning can be exploited, *e.g.* for expressive querying. Most formal = 5.

Rigorous relations: Rigorous relations have a clear ontological definition that can be codified using a formalism. The use of rigorous relations like RO improves the quality of ontologies, as the integration of such ontologies with other ontologies is made more efficiently, and more properties can be used for querying *via* automated reasoning. Highest number of rigorous relations = 5.

Cohesion: A strong cohesion (high connectivity between classes) is usually a sign of a high quality ontology, as, *e.g.* it indicates a rich axiomisation that can be exploited for automated reasoning. Highest connectivity = 5.

Tangledness: Asserted tangledness in an ontology is due to multiple inheritance, which is difficult to maintain [73]. Least tangled = 5.

Redundancy: A redundant ontology will have a high number of non-informative entities, *i.e.* entities that could be codified with fewer axioms, and therefore improve the maintenance of the ontology. Least number of redundant entities = 5.

Structural accuracy: Structural accuracy assesses the correctness of the terms used in the ontology, and it can be evaluated by looking for the terms in resources such as WordNet[2].

Domain coverage: If an ontology completely covers the defined domain, it can be used for many tasks within the domain. Therefore, highest coverage = 5.

Functionality: This category analyses how the ontology performs its intended roles.

Competence adequacy: Is the ontology appropriate for its intended use? The appropriateness is measured according to the different uses to which a bio-ontology can be put, as described in [109] and mentioned in Chapter 2. The uses are: reference ontology, controlled vocabulary, schema and value reconciliation, consistent search and query, knowledge acquisition, clustering and similarity, indexing and linking, results representation, classifying instances, text analysis, guidance and decision trees, knowledge reuse, inferencing. The last two, knowledge reuse and inferencing, were added and are not present in the list from [109]. Most appropriate ontology for each use = 5.

[2]`http://wordnet.princeton.edu/`

Interoperability: Interoperability measures to which extent the knowledge modelled in the ontology can be combined with that present in other ontologies to solve a concrete task. Most interoperable = 5.

Reliability: This category analyses the capability of an ontology to maintain its level of performance.

Maturity:

Technological: Technological maturity refers to the technologies available for editing and exploiting an ontology, like OBO or OWL. Most technologically mature = 5.

Knowledge: The frequency of modifications indicates the maturity of the knowledge represented in an (actively maintained) ontology: the less frequent the changes, the more mature the ontology. Most mature knowledge = 5.

Robustness: The robustness of an ontology assesses how the ontology is affected in the case that incorrect knowledge is identified *within* the ontology (it does not refer to knowledge not captured by the ontology). Most robust = 5.

Authority: The authority of an ontology comes from the authority of its curators, which can be measured with traditional measures like publications of the authors, H index[3], *etc.* Biggest authority = 5.

Usability: This category assesses the effort needed to use an ontology.

Readability: Readability is measured by the "human readable" part of the ontology, that is, the quantity and quality of annotation values. For example, the annotation values of `rdfs:label`, `rdfs:comment`, Dublin core annotation properties, and custom annotation properties of the ontology like `obsolete`. Greatest readability = 5.

Reusability: Capability of the ontology, or parts of the ontology, to be reused in other ontologies, *e.g.* via importing. A low reusability can be due to the modelling itself (*e.g.* an ontology that is not modular or does not make

[3]The H index is a measurement of the impact of the work performed by a scientist (`http://en.wikipedia.org/wiki/Hirsch_number`).

use of an Upper Level Ontology) or due to the underlying technology (*e.g.* OBO does not allow for importing). Most reusable = 5.

Efficiency: This category analyses the relation between the level of performance of the ontology and the amount of resources used, like time consumption, memory consumption, *etc.* Most efficient = 5.

Maintainability: This category measures the effort needed to make specific modifications.

Stability: The stability of an ontology is measured by assessing how the introduction of wrong axioms affects the ontology. Stability relates to errors in the axiomisation, and robustness relates to errors at a conceptual level. Most stable = 5.

Changeability: Effort needed for adding new axioms to the ontology. Highest changeability = 5.

Analysability: This criterion measures the difficulty in diagnosing problems in an ontology, in other words, how efficiently can the ontology be debugged. Highest analysability = 5.

Testability: Effort needed to validate the ontology, *e.g.* by automated reasoning, and how meaningful is such validation. Highest testability = 5.

Quality in use This category analyses the final product, when used in real conditions.

Effectiveness: Ability of the ontology to fulfil concrete user needs. Highest effectiveness = 5.

User satisfaction:

Popularity: The satisfaction of the users of the ontology. Highest popularity = 5.

Engagement: The degree to which users and projects make an intensive use of the ontology. Highest engagement = 5.

5.5 Conclusions

The contribution of this chapter is an evaluation framework for assessing ODPs and their impact on ontology development, divided into three areas: ODP quality, ontology engineering, and ontology quality. The framework can be used to assign quality values to ODPs and ontologies modified by ODPs, generating radar graphs that provide an intuitive idea of relative ODP and ontology quality. Therefore, the chapter has answered the following research questions:

How can we assess ODP quality? The following criteria have been defined to evaluate ODP quality: modelling probability, modelling benefit, use efficiency, specificity, documentation clarity, modelling toll, reasoning toll, inconsistency risk, tolerance to bad practice, maintainability, and community commitment.

How can we assess the impact of ODPs in bio-ontology engineering? The following criteria have been defined to evaluate how different ODPs affect ontology engineering: focused development, fast development, prototyping, reengineering, documentation of the process, communication between developers, predictability of consequences, debugging, and principled modelling.

How can we assess the change of quality of bio-ontologies as a result of applying ODPs? An adapted version of the ISO 9126 standard for software quality assessment has been defined to evaluate ontology quality.

Chapter 6

Evaluation results

This chapter describes the use cases for the application of ODPs in bio-ontologies, and the evaluation of the results, using the evaluation framework described in Chapter 5. Therefore, this chapter provides answers to the research questions *How can we assess ODP quality?*, *How can we assess the impact of ODPs in bio-ontology engineering?*, and *How does the use of ODPs change the quality of concrete bio-ontologies?*

The chapter is organised as follows. Section 6.1 describes the concrete use cases (Upper Level Ontology ODP in CCO, Sequence ODP in CCO, Entity-Quality ODP in GO, Selector ODP in GO, Normalisation ODP in CL) and the process of applying such ODPs in the concrete bio-ontologies. Section 6.2 analyses the results of evaluating the ODPs (ODP quality and ontology engineering) and their application (ontology quality).

6.1 Execution of use cases

The execution of the evaluation consisted of applying some ODPs in bio-ontologies and evaluating the process and the result using the framework described in Chapter 5. The following ODPs were selected from the online catalogue and applied in concrete bio-ontologies: the Upper Level Ontology ODP in CCO, the Sequence ODP in CCO, the Entity-Quality ODP in GO, the Selector ODP in GO, and the Normalisation ODP in CL.

GO and CL were selected for applying ODPs due to the fact that they are widely used bio-ontologies, especially GO [11]. CCO was selected because the author was involved in its development. The Upper Level Ontology ODP, the Sequence ODP, and the Normalisation ODP were selected because they fulfilled specific modelling requirements of CCO and CL. The Entity-Quality ODP was selected due to the fact that it is closely related to the Entity-Property-Quality ODP and the Entity-Feature-Value ODP, therefore making it an informative case for the ODP quality evaluation, as such evaluation is relative to other the ODPs. The Selector ODP was selected because it allowed to exploit the fact that GO `biological regulation` term names are syntactically structured, therefore making it possible to exploit OPPL's annotation processing capabilities.

The five ODPs were evaluated on ODP quality and ontology engineering, but only the Normalisation ODP was thoroughly evaluated on ontology quality. In the case of the other ODPs, only the most important issues are commented on this thesis with respect to ontology quality.

In the ontology quality area only one ODP was evaluated because, in order to perform a realistic evaluation, a real bio-ontology with real curators is needed, and performing such analysis on every ODP is beyond the scale of this work. This is mainly because it is unlikely to find bio-ontology curators ready to engage in a new ontology engineering method like the usage of ODPs in OWL. Also, such curators should be chosen from different bio-ontologies, as not all the ODPs can be applied in all bio-ontologies, making the experimental setting more difficult.

Given the resource limitations, the Normalisation ODP was chosen because it deeply changes the structure of the target ontology, whereas other ODPs make more superficial changes, and therefore it has a higher impact. Another justification for choosing the Normalisation ODP is its ready application in a collaborative setting, which allows assessment of the reaction of other curators, something less tangible with other ODPs. The Normalisation ODP allows collaborative work because it separates the structure of the target ontology into two main parts: a subsumption hierarchy and a collection of defined classes that will be used for automatic reasoning. The classes of the subsumption hierarchy can be divided between different curators in order for each curator to add axioms to those classes, and hence the different curators can efficiently collaborate.

There are two main scenarios where ODPs' application can be analysed: building bio-ontologies from scratch or applying ODPs to improve already existing bio-ontologies. We chose the second scenario, as assessing the process of bio-ontology building from scratch would need to capture many more variables. For example, the complexity of the domain of knowledge, the knowledge of the curators about such domain, and the OWL expertise of the curators should be taken into account, which is complex and not completely representative. Even if such variables were somehow taken into account, providing a collection of ODPs and some modelling problems, and expecting some modellers to choose the correct ODP in an experimental setting has been already shown not to yield conclusive results [25].

Finally, and most importantly, another justification for an ontology to ontology comparison is that already existing bio-ontologies reflect the tension between best practice and real practice more realistically than ontologies created from scratch in an experimental setting. In other words, sometimes bio-ontology developers have precise priorities and real implementation needs (*e.g.* updated annotations for database integration) that can clash with the use of ODPs, and that should be taken into account.

The following sections describe the details of the application of each ODP in each of the bio-ontologies. Section 6.2 analyses the results of evaluating that application.

6.1.1 Application of the Upper Level Ontology ODP in CCO

CCO integrates data from scattered resources and exploits implicit links between those resources to create a comprehensive representation of the cell cycle. CCO aims at providing a single resource to be queried by scientists interested in the cell cycle.

The information related to the cell cycle of the following resources is integrated in CCO: GO, UniProt, IntAct, GOA, and NCBI taxonomy[1]. CCO is created as a result of filtering and linking such information in an automatic Perl pipeline. The pipeline starts from the GO `cell cycle` subtree, and adds the pertinent UniProt entries using the GOA files as a filter. The added UniProt entries are used to retrieve the pertinent molecular interaction entries from IntAct, and any extra GO fragment (*e.g.* cellular components). CCO contains information about four model organisms, *H. sapiens*, *S. cerevisiae*, *S. pombe*, and *A. thaliana*, and the pertinent NCBI taxonomy subtree is added to each of them.

The Upper Level Ontology ODP[2] was applied in CCO to make the modelling process more principled and accommodate the other resources within CCO, as mentioned in [10]. A simple Upper Level Ontology was created for CCO, in order to avoid excessive commitment to a concrete view of the domain (*e.g.* as in BFO) but still ensure the possibility of adding new subontologies. A simple Upper Level Ontology can be

[1]`http://www.ncbi.nlm.nih.gov/Taxonomy/taxonomyhome.html/`

[2]`http://www.gong.manchester.ac.uk/odp/html/Upper_Level_Ontology.html`

extended with more fine grained content, but a rich Upper Level Ontology is more difficult to trim down, and therefore a lean Upper Level Ontology was chosen for CCO as it was a project in its early development stages. OPPL was not used to apply this ODP, differing from the other use cases, as CCO was produced by a Perl pipeline; the concrete Upper Level Ontology used was created manually and the rest of the ontology parts were attached to it in the pipeline.

6.1.2 Application of the Sequence ODP in CCO

The terms that describe the phases of the cell cycle in CCO (the subtree under `cell cycle phase`) come from GO. As GO only presents the properties `is_a`, `part_of`, `regulates`, `positively_regulates` and `negatively_regulates`, the sequentiality of those phases cannot be represented. Thus, the fact that, assuming that the cell cycle starts in the G1 phase, G1 is followed by S, G2 and M, in that order, cannot be represented in GO. The same applies for other subphases of the cell cycle, like the phases of mitosis.

The Sequence ODP[3] was applied in CCO to add such a sequential aspect to the cell cycle phases [10]. The OPPL script used can be seen in Figure 4.6. By adding sequentiality to the phases, queries regarding the timing of the phases could be made, *e.g. Does this protein participate in a process that happens at S and any other phase after that?*

The Sequence ODP presents a structure where the phases are related using the RO transitive property `preceded_by`[4]. A further property, `immediately_preceded_by`, is added as a functional subproperty of `preceded_by`, to query phases that happen before or after a given phase, without querying the rest of the phases (this feature is useful when the user is not aware of all the phases of a process, *e.g.* the phases of meiosis). As mentioned, the assumption behind the application of the Sequence ODP in CCO is to reduce a cycle, in which phases cannot be really regarded as happening before or after other phases (in a cycle everything happens before or after everything), to a sequence of events starting in G1.

[3]http://www.gong.manchester.ac.uk/odp/html/Sequence.html

[4]http://www.obofoundry.org/ro/#OBO_REL:preceded_by

6.1.3 Application of the Entity-Quality ODP in GO

The Entity-Quality ODP[5], together with the Entity-Property-Quality ODP[6] and the Entity-Feature-Value ODP[7], form a group of three alternatives for modelling modifiers and values [30]. Modifiers are dependent entities that refine an aspect of an independent entity, *e.g.* a car (independent entity) is refined by being white (white is a dependent entity). A modifier can take different values.

The three ODPs perform the following functions in different ways: representing which modifiers are applicable to which independent entities, representing whether an entity can have one or more modifiers, representing which values pertain to each modifier, and representing the constraints between the values. Therefore each ODP will fit different modelling requirements.

The Entity-Quality ODP consists of a single generic property that links modifiers to independent entities –`(inv) has_quality`–. The advantage of the Entity-Quality ODP is its simplicity, due to the fact that only one property is needed. However, a further axiom is needed to express the fact that a given entity presents a quality or not. A QCR is used for that ask, *e.g.* `cell_part subClassOf has_quality max 1 position` if the quality is optional, or `cell_part subClassOf has_quality exactly 1 position` if the quality is intrinsic to the entity.

The Entity-Property-Quality ODP represents each quality using an object property, which results in a proliferation of object properties, but offers the possibility of restricting which modifiers apply to which entities by simply using domain and range axioms. The Entity-Feature-Value ODP is the only one that allows the modelling of modifiers with multiple aspects (*e.g.* colour with a certain saturation and brightness), but it is the most complex one.

The Entity-Quality ODP was applied in GO with the OPPL script shown in Figure 4.16 [30], as a demonstration of how to use OPPL to apply ODPs.

6.1.4 Application of the Selector ODP in GO

As mentioned in chapter 2, the Selector ODP[8] is used to model entities through selectors like right and left. Using the this ODP, we avoid having entities like `right hand` or `left hand`, instead modelling an entity `hand` and then adding the selectors `right` or

[5]http://www.gong.manchester.ac.uk/odp/html/Entity_Quality.html
[6]http://www.gong.manchester.ac.uk/odp/html/Entity_Property_Quality.html
[7]http://www.gong.manchester.ac.uk/odp/html/Entity_Feature_Value.html
[8]http://www.gong.manchester.ac.uk/odp/html/Selector.html

```
Class: negative regulation of cytokine production
    equivalentTo:
        biological regulation,
        is_regulation_type some negative,
        regulates some cytokine production
```

Figure 6.1: Redefinition of the GO term `negative regulation of cytokine production` using the Selector ODP. After the redefinition, the term `negative regulation of cytokine production` is equivalent to any biological regulation of type negative that regulates cytokine production.

`left` as appropriate. For example, to refer to a finger of the right hand, the following expression would be used, without having to mention `right hand`: `part_of only hand and has_laterality some right`. Using this ODP the number of classes needed in the ontology decreases, and querying possibilities increase.

The Selector ODP can be used to model the regulation processes in GO. In GO, regulation is always positive or negative, and the properties `positively_regulates` and `negatively_regulates` are used to relate processes that regulate other processes, *e.g.* `negative regulation of cytokine production` is linked through the property `negatively_regulates` to `cytokine production`. Therefore, regulation can be defined by modelling `positive` and `negative` as selectors, like right and left in the hand example. Using such pattern, the class `negative regulation of cytokine production` is redefined as shown in Figure 6.1.

The Selector ODP was applied in GO (02:12:2008 11:31 version, CVS revision 5.896) using the OPPL script shown in Figure 6.2. Before applying the Selector ODP, the GO terms starting with `negative regulation of`, `positive regulation of` or `regulation of` were detached from the subsumption hierarchy, in order to test whether such a hierarchy could be recreated using inference after applying the ODP (the `part_of` relationships were maintained). ONTO-PERL was used for this task. When applying the ODP with OPPL, the GO properties `positively_regulates` and `negatively_regulates` were also removed.

For another example of the application of the Selector ODP, see the application in the Foundational Model of Anatomy (FMA) [91] described in [74].

```
ADD Class: Regulation;ADD label "regulation";

ADD Class: PositiveRegulation;ADD subClassOf Regulation;REMOVE subClassOf Thing;
ADD label "positive regulation";

ADD Class: NegativeRegulation;ADD subClassOf Regulation;REMOVE subClassOf Thing;
ADD label "negative regulation";

SELECT Class: PositiveRegulation;ADD disjointWith NegativeRegulation;

SELECT Class: Regulation;ADD equivalentTo PositiveRegulation or
NegativeRegulation;

ADD ObjectProperty: is_regulation_type;ADD functional;

REMOVE ObjectProperty: negatively_regulates;

REMOVE ObjectProperty: positively_regulates;

# Since we have removed negatively_regulates and positively_regulates, there is
# no axiom including regulates so we add it

ADD ObjectProperty: regulates;

SELECT label "(^negative regulation|positive regulation|regulation) of (.+$)";
ADD equivalentTo GO_0065007 and (is_regulation_type some <1>) and
(regulates some <2>);
```

Figure 6.2: OPPL script for applying the Selector ODP in GO. This is a reduced version of the actual script, for the sake of clarity. The first half of the script adds the selectors, `positive regulation` and `negative regulation`. The second half adds the actual Selector ODP: the property `is_regulation_type` is added, the GO properties `positively_regulates` and `negatively_regulates` are removed, and the annotation values are queried to redefine each class that matches the defined regular expressions(`(^negative regulation|positive regulation|regulation) of (.+$)`). Each class that matches the regular expression with its label annotation value will be redefined as `equivalentTo GO_0065007 and (is_regulation_type some <1>) and (regulates some <2>)`, being `<1>` and `<2>` the matched groups 1 and 2.

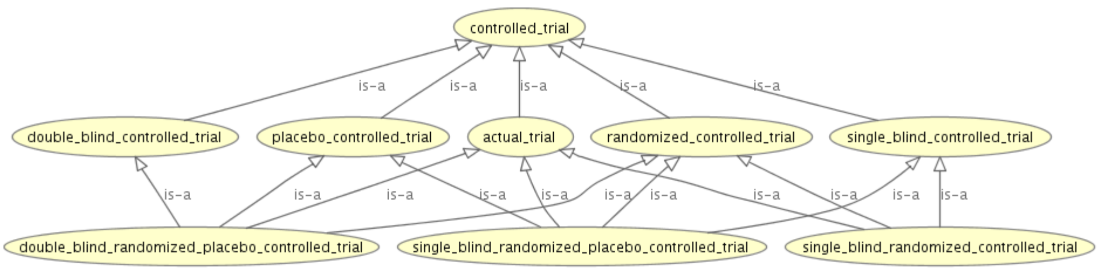

controlled_trial
is-a
double_blind_controlled_trial
placebo_controlled_trial
actual_trial
randomized_controlled_trial
single_blind_controlled_trial
double_blind_randomized_placebo_controlled_trial
single_blind_randomized_placebo_controlled_trial
single_blind_randomized_controlled_trial

Figure 6.3: A non-normalised ontology. All the subsumption relationships are manually asserted.

6.1.5 Application of the Normalisation ODP in CL

The Normalisation ODP[9] is used for creating and maintaining ontologies that are structured with multiple inheritance, that is, most of the classes have more than one superclass, forming a "polyhierarchy". The Normalisation ODP consists of building an ontology in a way that the reasoner is the one that maintains the polyhierarchy, instead of relying on curators manually maintaining it. In order to obtain such a structure, the ontology is divided in two parts: the primitive axis and the modules. The primitive axis is formed by classes with only one superclass, sibling-wise disjoint and primitive (classes with only necessary conditions). The modules are defined classes (classes with necessary and sufficient conditions) that are not disjoint, and are used by the reasoner to obtain the polyhierarchy: each of the modules has a restriction as a necessary and sufficient condition. Usually the classes from the primitive axis form the bulk of the ontology and they have more than one restriction that it is the same as any of the restrictions of the modules; therefore, the reasoner will infer that the different modules are superclasses of different primitive classes, resulting in each of the primitive classes having more than one module as a superclass. The multiple inheritance structure is obtained by running the reasoner. An example of how Normalisation works, inspired by a use case from OBI, can be seen in Figures 6.3, 6.4 and 6.5. A non-normalised ontology can be seen in Figure 6.3; a normalised ontology before and after automated reasoning can be seen in Figures 6.4 and 6.5, respectively.

CL[10] is a bio-ontology that describes canonical cell types that has been used in

[9]http://www.gong.manchester.ac.uk/odp/html/Normalisation.html
[10]http://www.obofoundry.org/cgi-bin/detail.cgi?id=cell

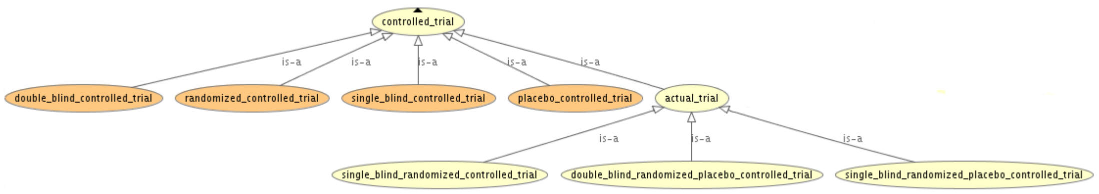

controlled_trial
is-a
double_blind_controlled_trial
randomized_controlled_trial
single_blind_controlled_trial
placebo_controlled_trial
actual_trial
single_blind_randomized_controlled_trial
double_blind_randomized_placebo_controlled_trial
single_blind_randomized_placebo_controlled_trial

Figure 6.4: A normalised ontology before automated reasoning. The modules are the dark circles, and the primitive axis is formed by the classes under `actual_trial`.

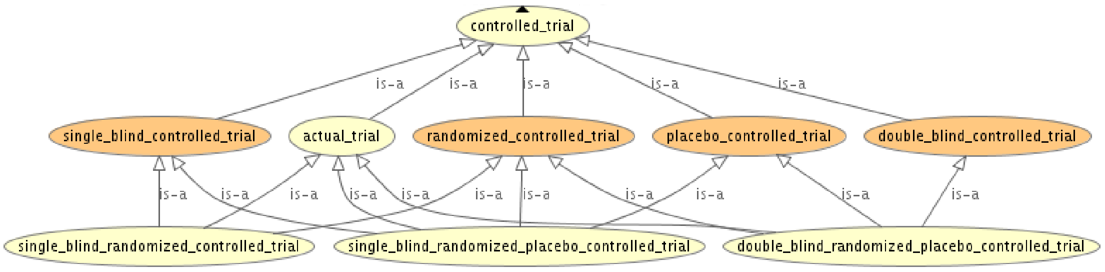

controlled_trial
is-a
single_blind_controlled_trial
actual_trial
randomized_controlled_trial
placebo_controlled_trial
double_blind_controlled_trial
single_blind_randomized_controlled_trial
single_blind_randomized_placebo_controlled_trial
double_blind_randomized_placebo_controlled_trial

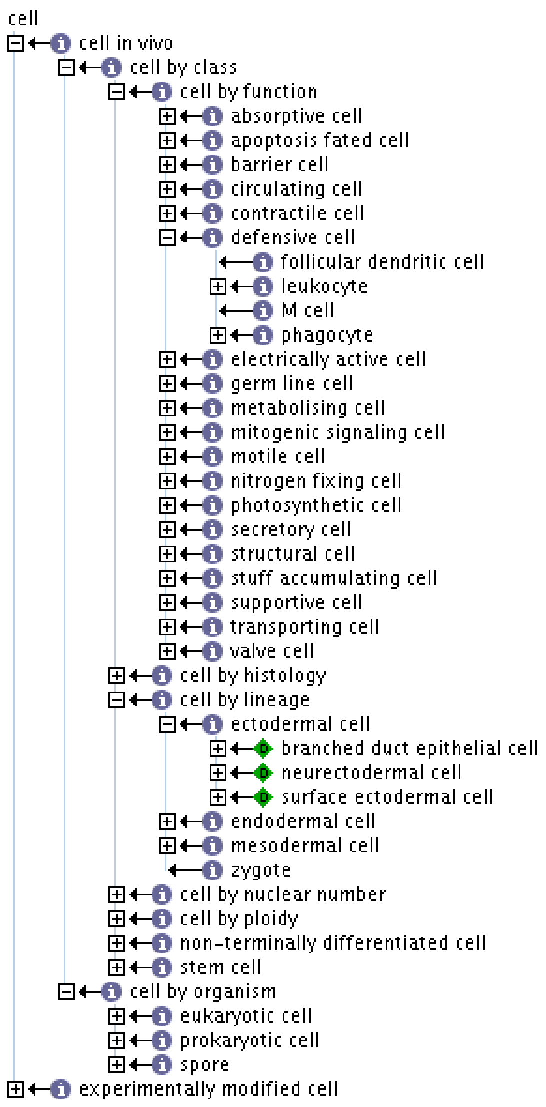
Figure 6.5: A normalised ontology after automated reasoning. Most of the subsumption relationships are inferred by the reasoner: each primitive class is a subclass of more than one defined class (module) inferred - and one primitive class (asserted).

projects like XSPAN[11]. The structure of CL is ideal for normalisation, since it consists of a classification of each cell type according to different criteria, resulting in asserted multiple inheritance (Figure 6.6). For example, a cell can be an eukaryotic cell (classification according to organism), a multinucleate cell (classification according to number of nuclei), a diploid cell (classification according to ploidy) and a defensive cell (classification according to function), and it will therefore have four superclasses. This results in a polyhierarchy, as each cell has more than one cell type superclass, and those superclasses are shared by other cells (*e.g.* there are various cells that have `defensive cell` as a superclass).

CL has two relationships: `is-a` and `develops-from`. The relationship `is-a` is used for expressing a subsumption relationship (*e.g.* `alveolar macrophage` is a subclass of `macrophage`) and `develops-from` is used for expressing that a cell is derived from another cell (*e.g.* `chondrocyte` is derived from `chondroblast`).

In order to apply the Normalisation ODP in CL, a two day meeting was organised with different scientists, to try to collaboratively recreate the structure of the original CL (herein called CL), in OBO format, in a normalised CL (herein called nCL), in OWL.

[11]`http://www.xspan.org/obo/index.html`



```
cell
  cell in vivo
    cell by class
      cell by function
        absorptive cell
        apoptosis fated cell
        barrier cell
        circulating cell
        contractile cell
        defensive cell
          follicular dendritic cell
          leukocyte
          M cell
          phagocyte
        electrically active cell
        germ line cell
        metabolising cell
        mitogenic signaling cell
        motile cell
        nitrogen fixing cell
        photosynthetic cell
        secretory cell
        structural cell
        stuff accumulating cell
        supportive cell
        transporting cell
        valve cell
      cell by histology
      cell by lineage
        ectodermal cell
          branched duct epithelial cell
          neurectodermal cell
          surface ectodermal cell
        endodermal cell
        mesodermal cell
        zygote
      cell by nuclear number
      cell by ploidy
      non-terminally differentiated cell
      stem cell
    cell by organism
      eukaryotic cell
      prokaryotic cell
      spore
  experimentally modified cell
```

Figure 6.6: Structure of CL. Circles with an “i” within are `is-a` relationships, and diamonds with a “D” within `develops-from` relationships. The ontology is divided into various axes (`cell by function`, `cell by lineage`, *etc.*) and each cell can have more than one superclass on each of those axes (*e.g.* the same cell can be defensive and mesodermal), creating a polyhierarchy.

During the two days, a normalisation schema was agreed, based on a set of properties to describe cells: `ploidy`, `morphology`, `cellular component`, `size`, `germ line`, `nucleation`, `process`, `process comment`, `lineage`, `organism`, `organism comment`, `potentiality`, `anatomy`, `cell surface protein`, `comments`. The leaf terms of the ontology were selected, thus cells with no child terms along `is-a` or `develops-from` relationships, and divided in groups by following the CL classification according to functions: `absorptive cell`, `defensive cell`, *etc.* Each of the participants took responsibility for a group of cells and described them using the agreed properties, filling a common excel file in google docs[12]. The Entity-Property-Quality ODP was also applied, albeit not explicitly, to add the fillers of the properties of each cell, as for each quality there was a property (*e.g.* `diploid`–`ploidy`)[13].

Some of the fillers for the properties were terms from other ontologies, like GO or Phenotypic Quality Ontology[14] (PATO), that were imported in nCL. PATO was used for the fillers of the following properties: `ploidy`, `morphology`, `size`, `nucleation`, and `potentiality`. GO was used for obtaining the filler classes for the properties `cellular component` and `process`. The original ontologies were downloaded in OBO format and then converted to OWL using the OBO2OWL script from the GONG project[15]. OPPL was used to remove the obsolete classes.

The general normalisation process mainly consisted of generating a whole new CL in OWL, nCL, from scratch, with the contributions from the participants. The process was not a single event, that is, the cells were sent in different groups and a new ontology was automatically generated each time, and therefore each version of the ontology was bigger than the prior version. The pipeline consisted of a collection of Perl and OPPL scripts, divided in the following steps:

1. The main ontology is created (nCL) and GO, RO and PATO are imported, without obsoletes.

2. A scaffold is created by executing the OPPL script shown in Figures 6.7 and 6.8. The scaffold consists of the classes `Cell` (for concrete cells) and `CellType` (for

[12]`http://docs.google.com/`

[13]The same phenomenom takes place with the Selector ODP and the Value Partition ODP, as the selector values are usually represented in a value partition. Therefore, the structure is similar, but they are *conceptually* different ODPs: the emphasis in the Selector ODP lies in the fact that selector values and properties can be used to reduce the number of classes of an ontology, and the emphasis on the Value Partition ODP lies on representing the selector values (or any other partition of values).

[14]`http://www.obofoundry.org/cgi-bin/detail.cgi?id=quality`

[15]`http://www.gong.manchester.ac.uk/bin/OBO2OWL.tar.gz`

defined modules), a few customized object properties (*e.g.* `has_ploidy`), object properties from RO, and defined modules under `CellType` (*e.g.* `HaploidCell`). The class `TerminallyDifferentiated`, non-existent in PATO, is also added to it.

3. Organisms ("buried" in term names with the keyword "sensu", *e.g.* `receptor cell (sensu Animalia)`) are extracted from CL using ONTO-PERL and added under `Organism` (Figure 6.9). Extra organisms, not present in nCL, are added by demand of the participants.

4. The google docs spreadsheet is transformed into an OPPL file (Table 6.1 and Figure 6.10). Any information added by the participant but not appropriate for adding to the ontology (*e.g.* non-existent terms) is captured in annotations, to retain all the information.

5. Intermediate levels (cells that are not "leaf terms" on CL) are selected with ONTO-PERL, and added to nCL. Another google docs spreadsheet is used for adding arbitrary axioms to such intermediate cells. This spreadsheet has two extra columns, for adding arbitrary MOS expressions as necessary and sufficient or necessary conditions, to try to recreate the structure of the intermediate cells *via* automated reasoning. The original CL can also be improved by the curators, not necessarily following original CL structure.

6. The `develops-from` relationship from CL is substituted by the RO relationship `derives_from` in nCL. This is done by automatically traversing CL with ONTO-PERL and adding all the corresponding `derives_from` restrictions to nCL, in order to recreate the original structure. Three classes are added for representing lineages: cells that derive from the endoderm (`EndodermalLineageCell`), cells that derive from the mesoderm (`MesodermalLineageCell`), and cells that derive from the ectoderm (`EctodermalLineageCell`) (Figure 6.11).

7. All the leaf cells are made disjoint and some new combined modules are created (Figure 6.12). A combined module is a defined class that combines different object properties: *e.g.* `MultinucleateCellParticipatesInImmuneResponse` is defined as any cell that is multinucleate and participates in immune response.

Once the final OWL ontology (nCL) is generated, automated reasoning is applied to check consistency, to obtain query results, and, most importantly, to check whether

```
# General structure and properties

ADD Class: Cell;
ADD Class: CellType;
ADD ObjectProperty: has_ploidy;ADD functional;ADD domain cto:Cell or
cto:CellType;ADD range pato:PATO_0001374;
ADD ObjectProperty: has_morphology;ADD functional;ADD domain cto:Cell
or cto:CellType;ADD range pato:PATO_0000001;
ADD ObjectProperty: has_nucleation;ADD functional;ADD domain cto:Cell
or cto:CellType;ADD range pato:PATO_0001404;
ADD Class: Size;
ADD Class: Small;REMOVE subClassOf owl:Thing;ADD subClassOf cto:Size;
ADD Class: Large;REMOVE subClassOf owl:Thing;ADD subClassOf cto:Size;
ADD Class: Medium;REMOVE subClassOf owl:Thing;ADD subClassOf cto:Size;
SELECT assertedSubClassOf cto:Size;ADD disjointWithSiblings;
SELECT Class: Size;ADD equivalentTo cto:Small or cto:Large or cto:Medium;
REMOVE ObjectProperty: ro:is_a;
SELECT ObjectProperty: ro:participates_in;ADD domain cto:Cell or
cto:CellType;ADD range go:GO_0008150;
SELECT ObjectProperty: ro:derives_from;ADD domain cto:Cell or cto:CellType;
ADD range cto:Cell or cto:CellType;ADD transitive;
SELECT ObjectProperty: ro:located_in;ADD domain cto:Cell or cto:CellType;
ADD ObjectProperty: has_size;ADD functional;ADD domain cto:Cell or
cto:CellType;ADD range cto:Size;
ADD ObjectProperty: potentiality;ADD domain cto:Cell or cto:CellType;
ADD Class: TerminallyDifferentiated;REMOVE subClassOf owl:Thing;
ADD subClassOf pato:PATO_0001397;
```

Figure 6.7: OPPL script for creating the scaffold structure of nCL. Manually created script. The classes `Cell` and `CellType` are artefacts for holding the disjoint leaf cells (under `Cell`) and the intermediate cells (under `CellType`).

the modules have multiple subclasses and hence the original polyhierarchy is recreated. However, the recreation of the CL hierarchy is attempted in the CL fragments that are correct: improvements were introduced by the curators in other fragments. Therefore, nCL is not a completely equivalent recreation, *via* Normalisation, of CL.

This process was repeated every time a contribution from any participant was added to the google spreadsheet. The programming effort needed for automating the process was due to the fact that contributions of many participants and original content (*e.g.* `derives_from` completeness from CL) needed to be synchronised in order to produce nCL. The participants had different levels of expertise ranging from bio-ontology to pure cell biology, but only one of the participants had a computer science background. Therefore, the most useful skill was biological knowledge, in order to properly describe the cells.

```
# Modules (defined classes): we define modules for each of the properties,
# and some combinations as well to provide examples
# Some properties not defined as modules as no one added a filler for
# them: inv (part_of), GERM LINE

ADD Class: SphericalCell;REMOVE subClassOf owl:Thing;ADD subClassOf
cto:CellType;ADD equivalentTo cto:has_morphology some pato:PATO_0001499;
ADD Class: HaploidCell;REMOVE subClassOf owl:Thing;ADD subClassOf
cto:CellType;ADD equivalentTo cto:has_ploidy some pato:PATO_0001375;
ADD Class: DiploidCell;REMOVE subClassOf owl:Thing;ADD subClassOf
cto:CellType;ADD equivalentTo cto:has_ploidy some pato:PATO_0001394;
ADD Class: SmallCell;REMOVE subClassOf owl:Thing;ADD subClassOf cto:CellType;
ADD equivalentTo cto:has_size some cto:Small;
ADD Class: LargeCell;REMOVE subClassOf owl:Thing;ADD subClassOf cto:CellType;
ADD equivalentTo cto:has_size some cto:Large;
ADD Class: MediumCell;REMOVE subClassOf owl:Thing;ADD subClassOf cto:CellType;
ADD equivalentTo cto:has_size some cto:Medium;
ADD Class: MononucleateCell;REMOVE subClassOf owl:Thing;ADD subClassOf
cto:CellType;ADD equivalentTo cto:has_nucleation some pato:PATO_0001407;
ADD Class: AnucleateCell;REMOVE subClassOf owl:Thing;ADD subClassOf
cto:CellType;
ADD equivalentTo cto:has_nucleation some pato:PATO_0001405;

# There is no cell with a potentiality different from terminally
# differentiated: poor

ADD Class: TerminallyDifferentiatedCell;REMOVE subClassOf owl:Thing;
ADD subClassOf cto:CellType;ADD equivalentTo cto:potentiality some
cto:TerminallyDifferentiated;

# Anatomy and cell surface protein are left

# Queries
ADD Class: Query_1;ADD equivalentTo ro:participates_in some go:GO_0032940 and
ro:participates_in some go:GO_0007589;
ADD Class: Query_2;ADD equivalentTo ro:participates_in some go:GO_0065008;
ADD comment
"Transitivity of is_a allows us to exploit GO’s structure for queries";
```

Figure 6.8: OPPL script. Continued from Figure 6.7.

```
ADD Class: Organism;
ADD Class: Nematoda_and_Protostomia;ADD subClassOf cto:Organism;
REMOVE subClassOf owl:Thing;
ADD Class: Arthopoda;ADD subClassOf cto:Organism;REMOVE subClassOf owl:Thing;
ADD Class: Vertebrata;ADD subClassOf cto:Organism;REMOVE subClassOf owl:Thing;
ADD Class: Mus;ADD subClassOf cto:Organism;REMOVE subClassOf owl:Thing;
ADD Class: Endoptyeygota;ADD subClassOf cto:Organism;
REMOVE subClassOf owl:Thing;
ADD Class: Mycetozoa;ADD subClassOf cto:Organism;REMOVE subClassOf owl:Thing;
ADD Class: Arthropoda;ADD subClassOf cto:Organism;
REMOVE subClassOf owl:Thing;
ADD Class: Diptera;ADD subClassOf cto:Organism;REMOVE subClassOf owl:Thing;
ADD Class: Nematoda;ADD subClassOf cto:Organism;REMOVE subClassOf owl:Thing;
ADD Class: Fungi;ADD subClassOf cto:Organism;REMOVE subClassOf owl:Thing;
ADD Class: Insecta;ADD subClassOf cto:Organism;REMOVE subClassOf owl:Thing;
ADD Class: Animalia;ADD subClassOf cto:Organism;REMOVE subClassOf owl:Thing;
ADD Class: Viridiplantae;ADD subClassOf cto:Organism;
REMOVE subClassOf owl:Thing;
ADD Class: Actinopterygii_and_Amphibia;ADD subClassOf
cto:Organism;REMOVE subClassOf owl:Thing;
ADD Class: Mammalia;ADD subClassOf cto:Organism;REMOVE subClassOf owl:Thing;
ADD Class: Protostomia;ADD subClassOf cto:Organism;
REMOVE subClassOf owl:Thing;
ADD Class: Aves;ADD subClassOf cto:Protostomia;REMOVE subClassOf owl:Thing;
```

Figure 6.9: OPPL script for adding the necessary organisms to nCL. The script is automatically generated using ONTO-PERL.

```
ADD Class: CL_0000811;REMOVE subClassOf owl:Thing;
ADD label ``CD8-positive, alpha-beta immature T cell'';
ADD subClassOf cto:Cell;ADD subClassOf cto:has_ploidy some pato:PATO_0001394;
ADD comment ``MORPHOLOGY: pleiomorphic'';
ADD comment ``CELULAR COMPONENT: '';
ADD subClassOf cto:has_size some cto:Small;
ADD comment ``GERM LINE: n/a'';
ADD subClassOf cto:has_nucleation some pato:PATO_0001407;
ADD subClassOf cto:participates_in some go:GO_2456;
ADD subClassOf cto:participates_in some go:GO_0021700;
ADD subClassOf cto:participates_in some go:GO_0032940;
ADD comment ``PROCESS: '';
ADD comment ``LINEAGE: mesoderm'';
ADD subClassOf cto:appears_in some cto:Animalia;
ADD comment ``ORGANISM COMMENT: '';
ADD subClassOf cto:potentiality some cto:TerminallyDifferentiated;
```

Figure 6.10: OPPL script for adding one of the contributions from one of the participants, thus, one leaf cell. The same participant contributed with around 40 cells. The script is generated automatically by parsing the google docs spreadsheet, in tabular format, with a custom Perl script. The values that are empty or are invalid are added as annotation values to keep track of all the errors.

Term Name	**CL id**	**ploidy**	**morphology**	**Cellular component**	**size**	**germ line**	**nucleation**	**process**	**process comment**	**lineage**	**organism**	**Organism Comment**	**potentiality**	**anatomy**	**Cell surface protein**
CD8-positive, alpha-beta immature T cell	CL:0000811	PATO:0001394	pleiomorphic	n/a	small	mesoderm	PATO:0001407	GO:0002456 GO:0021700 GO:0032940	n/a	mesoderm	Animalia	n/a	Terminally differentiated	circulatory system	CD marker CD-8

Table 6.1: Excerpt from the google docs spreadsheet. Attributes (first line, bold) and their values for a concrete leaf cell (second line) are shown.

```
SELECT Class: cto:CL_0000189;ADD subClassOf ro:derives_from some cto:CL_0000857;
SELECT Class: cto:CL_0000854;ADD subClassOf ro:derives_from some cto:CL_0000032;
SELECT Class: cto:CL_0000098;ADD subClassOf ro:derives_from some cto:CL_0000710;
SELECT Class: cto:CL_0000214;ADD subClassOf ro:derives_from some cto:CL_0000134;
```

Figure 6.11: OPPL script for adding all the necessary derives_from relations to nCL. Automatically generated script.

```
# Make all the cells (leaf nodes) disjoint

SELECT assertedSubClassOf cto:Cell;ADD disjointWithSiblings;

# Combination of modules. This exploits reasoning as it uses
# the GO hierarchy is_a for the functions (participates_in)
# We create a module for each of the groups from Simon's page

ADD Class: MultinucleateCellParticipatesInImmuneResponse;
REMOVE subClassOf owl:Thing;ADD subClassOf cto:CellType;
ADD equivalentTo cto:has_nucleation some pato:PATO_0001908 and
ro:participates_in some go:GO_0006955;
ADD Class: EctodermalCellParticipatesInImmuneResponse;
REMOVE subClassOf owl:Thing;ADD subClassOf cto:CellType;
ADD equivalentTo ro:derives_from some cto:CL_0000221 and ro:participates_in
some go:GO_0006955;

# Develops from modules are defined later by DevelopsFrom.pl

ADD Class: MammalianCell;REMOVE subClassOf owl:Thing;
ADD subClassOf cto:CellType;ADD equivalentTo ro:located_in some cto:Mammalia;
```

Figure 6.12: OPPL script for adding some corrections to nCL. Manually created script.

6.2 Results

6.2.1 Upper Level Ontology ODP in CCO

The ODP quality graph of the Upper Level Ontology ODP (Figure 6.13) reveals three major problems and three major benefits. The problems are the following: need for a high community commitment, low tolerance to bad practice and high risk of inconsistency. The benefits, on the other hand, are the following: high modelling benefit, high use efficiency and low specificity.

The need for a high community commitment is due to the fact that the Upper Level Ontology chosen will determine the whole modelling of the ontology, as it provides the basic distinctions of entities in the ontology. The distinctions can be basic, as in the case of CCO (*e.g.* distinctions between processes and entities), or fine grained. The community commitment is difficult to achieve (and hence the low score) because of the controversy inherent in those distinctions, as illustrated by the discussions on the BFO public mailing list[16]. The low tolerance to bad practice (low score) is due to the fact that it is likely that a developer will add a new entity in the wrong branch of the ULO. The high inconsistency risk is due to the fact that, as the different branches are disjoint to each other, it is likely to produce an inconsistency by adding several superclasses, from different branches, to a new class.

In terms of advantages, the high modelling benefit (high score) is due to the principled modelling that results from the usage of an ULO, as the different types of concepts of the ontology are ordered in different branches. The Upper Level Ontology ODP is general (low specificity), as every knowledge domain needs ULOs, and ULOs ideally are applicable in different domains. Finally, the use efficiency is high because in order to add a new entity complex axiomisation is not needed, only subclass and disjoint axioms.

In terms of ontology engineering the Upper Level Ontology ODP presents high scores in every area except in predictability of consequences (Figure 6.14). The predictability of consequences is medium due to the fact that an ULO, if fine grained, imposes a concrete structure in which it is not always evident what problems will arise in later stages of development, *e.g.* when adding a new branch.

[16] `http://groups.google.com/group/bfo-discuss`

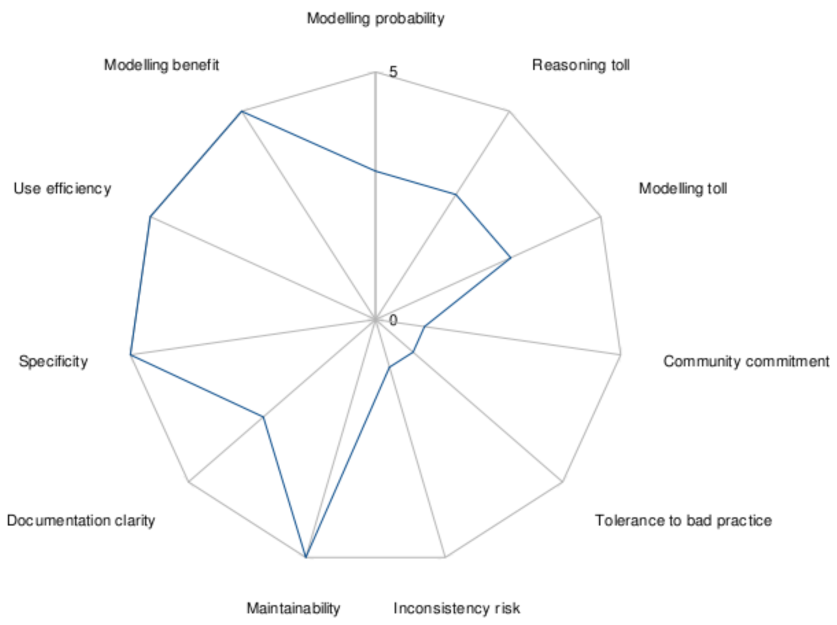


Figure 6.13: ODP quality radar graph of the Upper Level Ontology ODP. The highest qualities are obtained in modelling benefit, use efficiency, specificity and maintainability; the lowest in inconsistency risk, tolerance to bad practice, and community commitment.

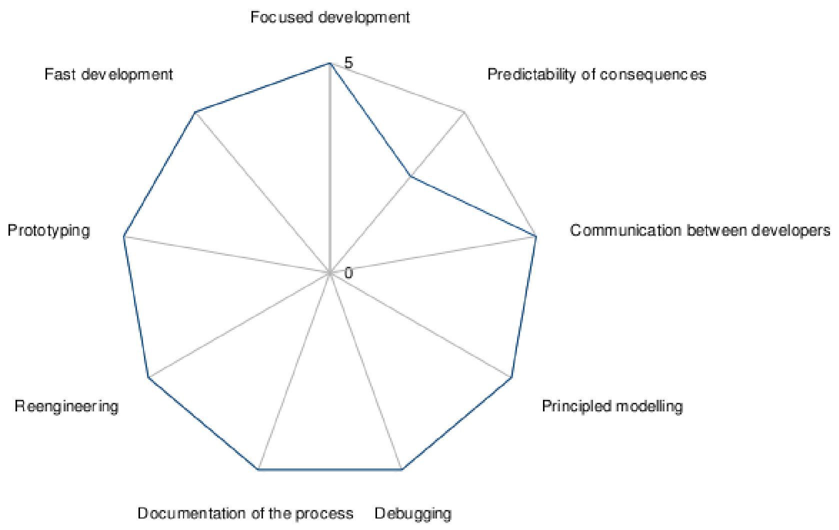


Figure 6.14: Ontology engineering radar graph of the Upper Level Ontology ODP. Every criterion gets a high score except predictability of consequences, which is medium.

6.2.2 Sequence ODP in CCO

The ODP quality radar graph, in Figure 6.15, shows that the Sequence ODP gets an average quality score in every area except in the areas of modelling benefit, community commitment and documentation clarity, where it gets a high score. The modelling benefit is high (high score) because the ODP fulfills a concrete modelling requirement, namely putting items in order, that is a common structure in biology. The needed community commitment is low (high score), as the modelling only affects a concrete portion of the ontology, and therefore a consensus can be promptly reached. The ODP is clearly documented (high score), as the problem that it solves is thoroughly explained.

In Ontology engineering, in Figure 6.16, it can be seen that the ODP gets average scores in every area except in prototyping, fast development and focused development, where it gets high scores. The high score in prototyping is due to the fact that, as sequential structures are a common requirement in biological knowledge, a prototype bio-ontology can be built quickly using this ODP in combination with other ODPs. The high scores in fast development and focused development are due to the same reason: the Sequence ODP offers a way of modelling a concrete and recurrent requirement in the biological domain, therefore allowing for a fast and focused development, as applying this ODP saves development time and developers can invest their effort in other areas of the bio-ontology.

6.2.3 Entity-Quality ODP in GO

As shown in the ODP quality radar graph of Figure 6.17, the Entity-Quality ODP gets high scores in reasoning toll, community commitment, inconsistency risk, and maintainability. The reasoning toll is low (high score) because this ODP does not rely on automated reasoning, and the axiomisation is not complex enough to represent a low automated reasoning performance. The community commitment needed is low (high score), as the Entity-Quality ODP is conceptually simple, and therefore it is likely to produce a consensus between the developers regarding the application of this ODP. The inconsistency risk is low (high score) because this ODP does not create a structure that is likely to produce inconsistencies, like a structure with a lot of disjoints or a structure in which adding a further quality creates an inconsistency (*e.g.* in the Entity-Property-Quality ODP the domain and range axioms can be misused). The ODP is highly maintainable (high score) due to its axiomatic simplicity, especially in front of the Entity-Feature-Value ODP. However, it should be noted that the QCR needed

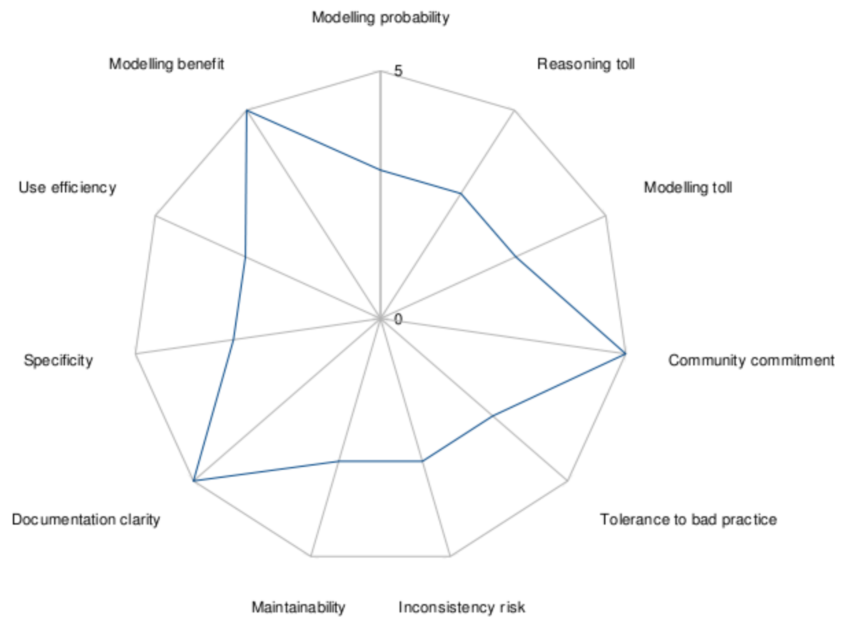


Figure 6.15: ODP quality radar graph of the Sequence ODP. There are three high scores, the rest being average: modelling benefit, documentation clarity, and community commitment.

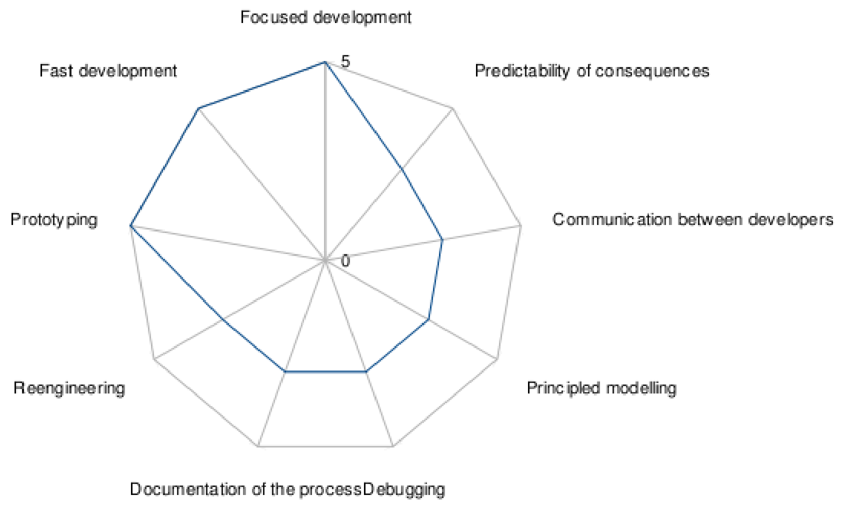


Figure 6.16: Ontology engineering radar graph of the Sequence ODP. There are three high scores (focused development, fast development, and prototyping) and the rest of the scores are average.

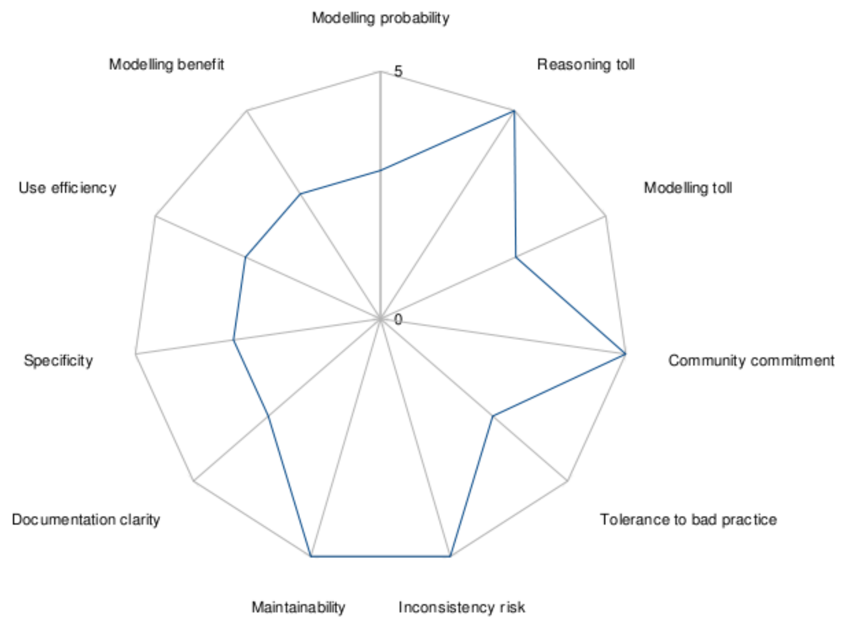


Figure 6.17: ODP quality radar graph of the Entity-Quality ODP. There are four high scores, the rest being average: reasoning toll, community commitment, inconsistency risk, and maintainability.

in the Entity-Quality ODP could be more difficult to maintain than the domain and range axioms of the Entity-Property-Quality ODP, which fulfil the same modelling requirement.

In Ontology engineering, as shown in Figure 6.18, reengineering stands out with a high score, as this ODP can be reused for different qualities (it uses one object property for every quality) and it is relatively easy to recycle it into the Entity-Property-Quality ODP. The focused development quality is low, as the modelling that this ODP offers does not address a complex modelling requirement.

6.2.4 Selector ODP in GO

The ODP quality radar graph of this ODP can be seen in Figure 6.19. Six areas get a high score: use efficiency, specificity, maintainability, inconsistency risk, community commitment and modelling toll. The use efficiency is high because it is relatively easy to add a new relation from an entity to a selector value. The specificity is low, as this ODP can be applied in any domain, and therefore the score is high. Maintainability and inconsistency risk get a high score because the Selector ODP structure is axiomatically simple, and therefore can be maintained without much effort and the likelihood of producing inconsistencies is low. The community commitment is not demanding, as it

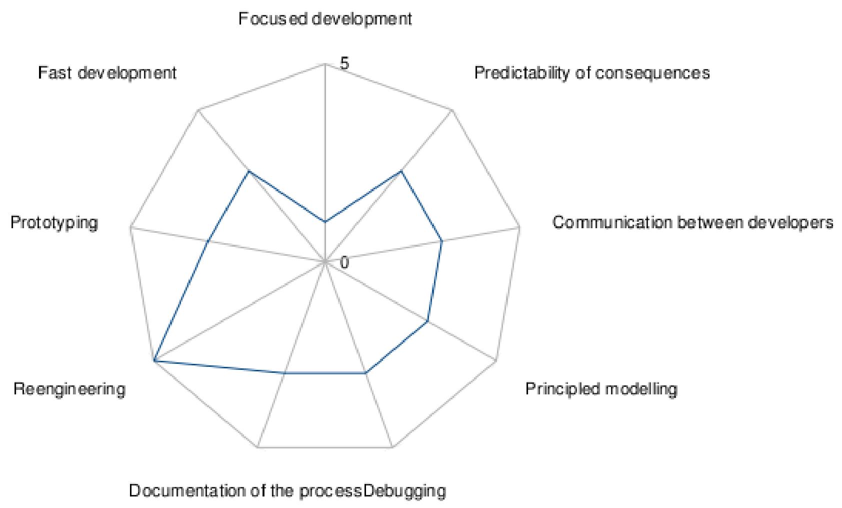


Figure 6.18: Ontology engineering radar graph of the Entity-Quality ODP. There is a high score, reengineering, and a low score, focused development, the rest being average.

is conceptually a simple ODP, and the modelling toll is also low as the axiomisation of the ODP is simple. The area that gets the worst score is the modelling probability, as the modelling that this ODP shows is more likely to be found by a developer without the aid of the ODP than other ODPs.

The Ontology engineering radar graph can be seen in Figure 6.20. The areas in which this ODP gets a high score are debugging and reengineering, the rest being average. Reengineering quality is high as due to its simplicity the ODP is likely to be reused without problems. Debugging quality is high because, as this ODP pulls apart entities and selector values, the modelling becomes cleaner and errors can be more promptly spotted.

In terms of ontology quality, it should be noted that the Selector ODP was not applied in a canonical way in GO, as the entities were maintained. Thus, coming back to the original example, `right hand` and `left hand` were maintained. For example, the class `negative regulation of cytokine production` was maintained instead of refering to it in expressions like `regulation and (is_regulation_type some negative) and (regulates some cytokine production)`. This was done in order to comply with the requirement of being able to annotate external entities with GO terms. As noted in [74], the requirement for reducing the number of entities of this ODP came from ontologies where such number could create a performance problem, like FMA. That is not the case for GO, and the requirement of maintaining the terms for annotation is more important.

The application of this ODP resulted in the axiomatic enrichment of GO, and hence improvent in querying and maintenance of the ontology. In terms of querying, more fine grained queries could be performed, *e.g.* queries of the form `is_regulation_type some negative` or `regulates some cytokine production`. The maintenance improved due to the possibility of recreating the `regulation` hierarchy using inference. Using inference makes the maintenance of such a hierarchy more efficient, as the complete hierarchy is produced by the reasoner, and, if it is not produced, errors are flagged in the axiomisation of such a hierarchy. The errors consisted mainly (but not exclusively) in missing processes that were mentioned in regulation terms. For example, `axonemal microtubule depolymerization` was mentioned in `positive regulation of axonemal microtubule depolymerization`, but did not exist as a standalone term in GO. Therefore, it did not get any extra axioms when the OPPL script was executed (there was no filler for the restriction `regulates some`) and it was left out of the regulation hierarchy when automated reasoning was performed, being flagged as problematic. Some of such problems were reported to the GO staff and new terms were added to the public version of GO, as shown in the GO curator tracker[17] (*e.g.* `cytosolic calcium ion transport`). The inferred regulation hierarchy, shown in Figure 6.22, considerably resembles the original GO hierarchy, shown in Figure 6.21. By applying the Selector ODP, the implicit axioms were made explicit and errors present in the actual modelling were shown.

The application of this ODP resembles the application of the Normalisation ODP in one aspect: the regulation hierarchy is recreated using defined classes, as happens with the intermediate nodes in nCL. However, the similarity is structural but not conceptual: the Selector ODP models selectors like positive and negative and the entities that are selected by such selectors, whereas Normalisation models the fact that the same entity can belong to many categories.

The Selector ODP was applied by querying the ontology with regular expressions, which means that fine grained regular expressions need to be defined if the whole ontology is to be processed. There comes a point, if too many regular expressions need to be defined, that it may not pay off. Also, it could be that the regular expression captures wrong terms. For example the term `regulation of neurotransmitter levels` was captured but there is not (and should not be) a `neurotransmitter levels` process. In any case, a few regular expressions led to a considerable axiomatic enrichment.

[17]`http://sourceforge.net/tracker/?func=detail&atid=440764&aid=2445858&group_id=36855`

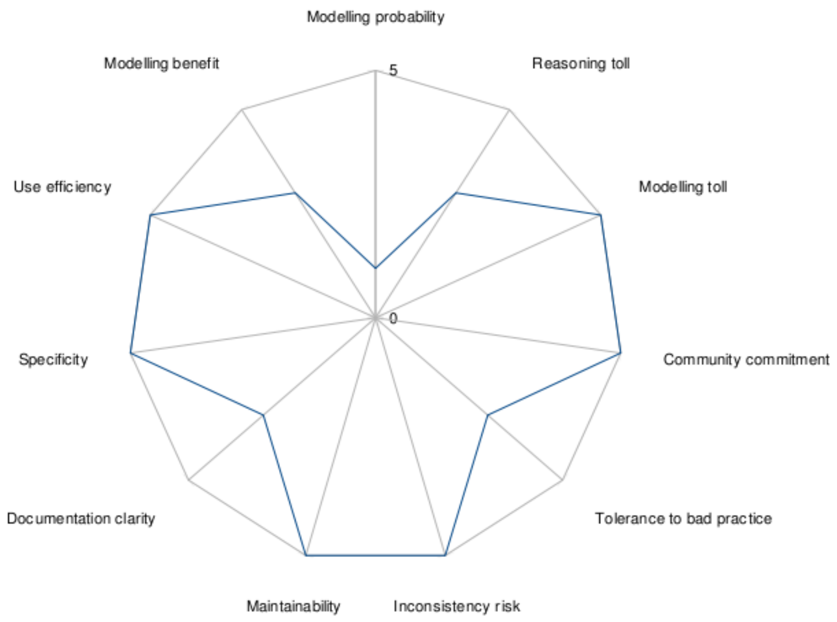


Figure 6.19: ODP quality radar graph of the Selector ODP. Six criteria score high: use efficiency, specificity, maintainability, inconsistency risk, community commitment and modelling toll. One criterion, modelling probability, scores low.

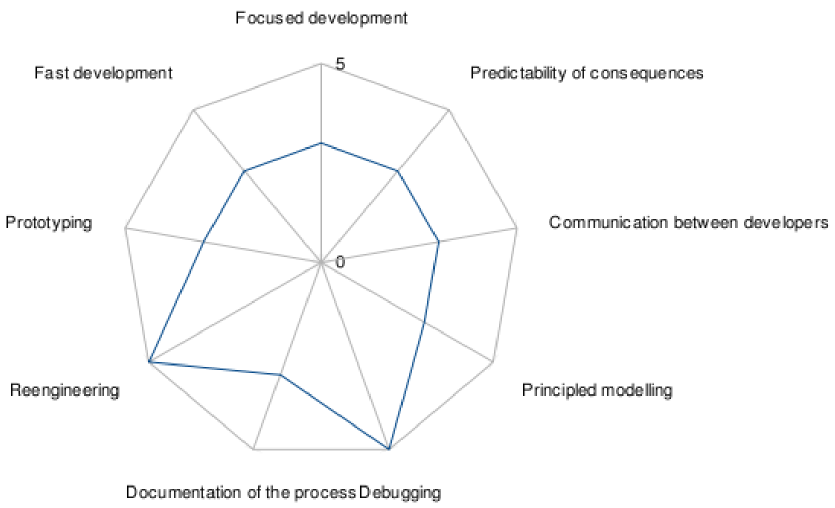


Figure 6.20: Ontology engineering radar graph of the Selector ODP. Only two criteria score high, the rest being average: reengineering and debugging.

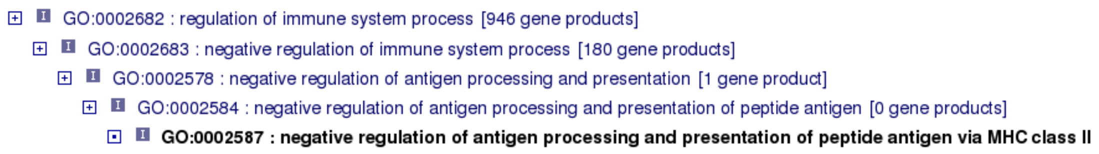

GO:0002682 : regulation of immune system process [946 gene products]
GO:0002683 : negative regulation of immune system process [180 gene products]
GO:0002578 : negative regulation of antigen processing and presentation [1 gene product]
GO:0002584 : negative regulation of antigen processing and presentation of peptide antigen [0 gene products]
GO:0002587 : negative regulation of antigen processing and presentation of peptide antigen via MHC class II

Figure 6.21: Fragment of the GO biological regulation hierarchy. This hierarchy is manually asserted by the GO curators.

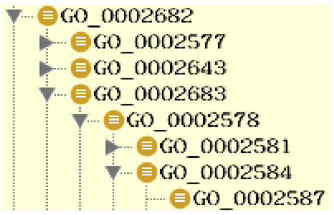


Figure 6.22: Fragment of the GO biological regulation hierarchy, recreated by automated reasoning. The new axiomisation resulting from the application of the Selector ODP allowed the reasoner to infer the hierarchy, which resembles the one shown in Figure 6.21.

6.2.5 Normalisation ODP in CL

Figure 6.23 shows the ODP quality radar graph, Figure 6.24 the Ontology engineering radar graph, and Figure 6.25 the Ontology quality radar graph, comparing CL before (CL) and after (nCL) applying the Normalisation ODP. The detailed values used to produce the ontology quality radar graph are shown in Appendix B. In ontology quality, another evaluation of the same set of ontologies was made by the MSc students of a Semantic Web course of the University of Murcia[18]. Both evaluations of ontology quality show that the nCL has a higher quality than CL.

In terms of ODP quality, it should be noted that both the reasoning toll quality and the modelling toll quality are low. The reasoning toll is high (low quality) because the ODP relies on the reasoner to maintain the *whole* structure of the ontology, which will represent a high load for the reasoner. The modelling toll is high because a lot of restrictions (at least one for each module or defined class) must be added, and in the case of modules added as equivalent conditions (at least an equivalent restriction per module). On the other hand, maintainability, modelling benefit and use efficiency score high, thus this ODP has a good quality in those areas. The maintainability is high because it is easier to maintain an exhaustive polyhierarchy using normalisation than manually. In an asserted polyhierarchy, the curator needs to explore the whole structure and add the appropriate subsumptions, which results in incomplete hierarchies [73, 89, 126]. In a normalised ontology, for any new added class the curator only needs to follow the object properties, as if filling a form, and add the appropriate fillers: the reasoner builds the polyhierarchy.

Modelling benefit is high due to the fact that the Normalisation affects the whole ontology, so if the Normalisation is the needed ODP, it can be highly beneficial to

[18]http://dis.um.es/~jfernand/icbo/

the target ontology. In the case of use efficiency, as mentioned, adding content to a normalised ontology is as simple as following the object properties and adding a filler when pertinent, so adding a new primitive or defined class is an efficient process.

In terms of ontology engineering, the scores are high except for debugging and predictability of consequences, which get average scores. Principled modelling quality is high because, as mentioned, Normalisation enforces a modelling dynamic that resembles filling a form: when adding a new leaf cell, a curator only needs to go through the object properties and add fillers. Also, Normalisation allows the extension of the ontology in a more principled way: a new axis of classification (*e.g.* cells classified by surface protein types [71]) can be added without affecting the rest of the axes. The fast development high score is due to the fact that the Normalisation ODP facilitates collaboration between different curators that contribute with their own content, something more difficult in an asserted polyhierarchy, making the development faster.

The score for debugging quality is average because, even though automated reasoning can be exploited for debugging, *e.g.* through justifications of entailments[19] (Figure 6.26), the use of automated reasoning for maintaining the structure can sometimes be difficult to debug. Predictability of consequences quality is also average because the ODP relies on automated reasoning for maintaining the structure: some axioms may yield unpredictable results, especially considering that, for generating the structure of the intermediate nodes, plenty of defined classes must be created.

In Ontology quality, the only dimension in which CL scores higher than nCL is in quality in use. This is due to the fact that CL is used by many more people and projects, as nCL is simply an experiment that has not yet been published.

The structure of nCL is of better quality mainly because of the lack of tangledness, although there are other criteria with high scores like the use of RO. Also, the axiomisation is more explicit and richer (high cohesion), as the subsumptions are inferred *via* restrictions: it is obvious *why* a class is a subclass of another class, as they both have at least a restriction that is identical. For example, the class `secretory cell` will have the restriction `participates_in some secretion` as a necessary and sufficient condition, and the class `Chromaffin cell` will have it as a necessary condition[20], and hence the subsumption relationship will be inferred.

The richer axiomisation of nCL also improves querying, as many more queries can be done using restrictions, and combining different object properties. In CL, there is

[19]`http://owl.cs.manchester.ac.uk/explanation/`

[20]Most probably it will have a subclass of `secretion` as a filler.

no room for queries as everything is hard-coded in subsumptions and object properties like `ploidy`, `participates_in` are not used. With a rich axiomisation automated reasoning can be used to exploit transitivity of `derives_from`. For example, we do not have to state which cell is mesodermal, as it must be done in CL: `fibroblast` was classified as a subclass of `mesodermal_lineage_cell`, even though only `derives_from some mesenchymal cell` was stated.

The transitivity of the OWL subsumption relationship was also exploited by automated reasoning: as `secondary spermatocyte` participates in `spermatogenesis`, which is a subclass of `sexual reproduction`, `secondary spermatocyte` was inferred to be a subclass of `sexual cell` (transitivity of `part_of` was also exploited for inference, *e.g.* when a cell participated in a process that was part of another process). Therefore, nCL, being a Normalised ontology, offers a wider range of functionalities than CL, *e.g.* when used as part of applications [84], due to its richer axiomisation.

Functionality in nCL is of higher quality because the functionalities of inference, decision support, reuse, instance classification and knowledge acquisition get a high score, due to a richer axiomisation and hence more meaningful automated reasoning

Efficiency in nCL is lower because it makes a heavy use of automated reasoning. Thus, it consumes a bigger amount of computational resources. However, it should be noted that as the functionality in nCL is of higher quality, nCL will consume less human resources.

Maintainability is of higher quality in nCL mainly because, as mentioned, automated reasoning can be exploited for maintenance, and therefore maintenance can be done automatically. Note that the maintainability in the case of ontology engineering compares the Normalisation ODP with other ODPs, and this maintainability compares nCL with CL.

The use of the Normalisation ODP brought benefits also to GO, as the process of performing the Normalisation in CL resulted in spotting missing terms in GO. The GO biological process hierarchy was not, in some cases, fine grained enough to describe function attributes of cells from CL, so new terms were needed and, when proposed to the GO curators, were accepted[21]: `Prolactin secretion`, `somatotropin secretion`, `thyroid stimulating hormone secretion` and `dentine secretion`.

[21]`http://sourceforge.net/tracker/index.php?func=detail&aid=2644865&group_id=36855&atid=440764`

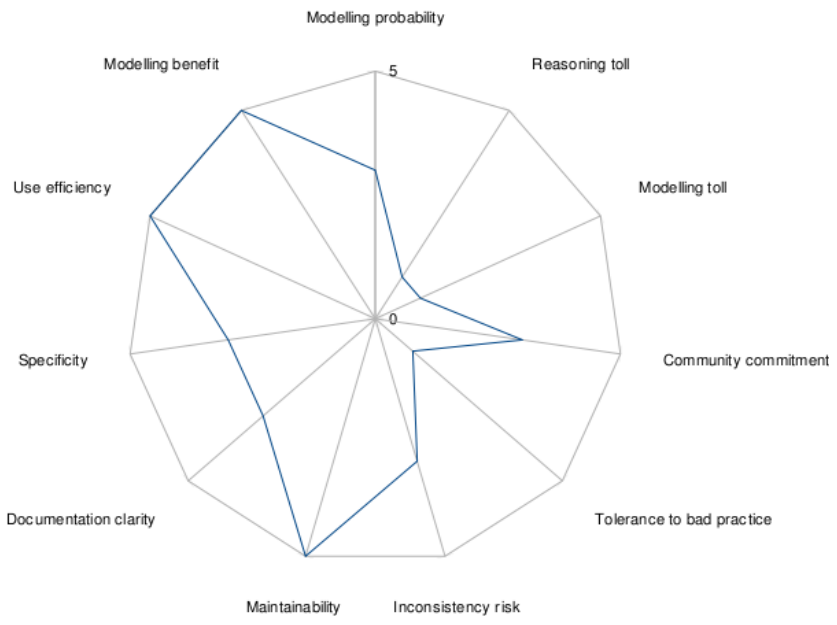


Figure 6.23: ODP quality radar graph of the Normalisation ODP. Three criteria get high scores: modelling benefit, use efficiency, and maintainability. Another three criteria get low scores: reasoning toll, modelling toll, and tolerance to bad practice.

This highlights the fact that, with a rich axiomisation and combining different bio-ontologies *via* imports, spotting problems is much more likely than using isolated bio-ontologies with a lean axiomisation, and the Normalisation ODP offers a procedure for obtaining such rich axiomisation in a principled manner.

6.3 Conclusions

This chapter has described the application of concrete ODPs in publicly available bio-ontologies: Upper Level Ontology ODP in CCO, Sequence ODP in CCO, Entity-Quality ODP in GO, Selector ODP in GO, and Normalisation ODP in CL. Such application has been evaluated using the framework described in Chapter 5, evaluating ODP quality and ontology engineering in the five use cases and also ontology quality in the Normalisation ODP in CL use case.

The contribution of the chapter is two fold: an explanation of how to use the evaluation framework, focusing on how to assign values to each criterion, and the results of applying the evaluation framework to the use cases, providing data for a conclusion regarding the usefulness of the usage of ODPs in bio-ontology engineering.

Therefore, the chapter has answered the following research questions:

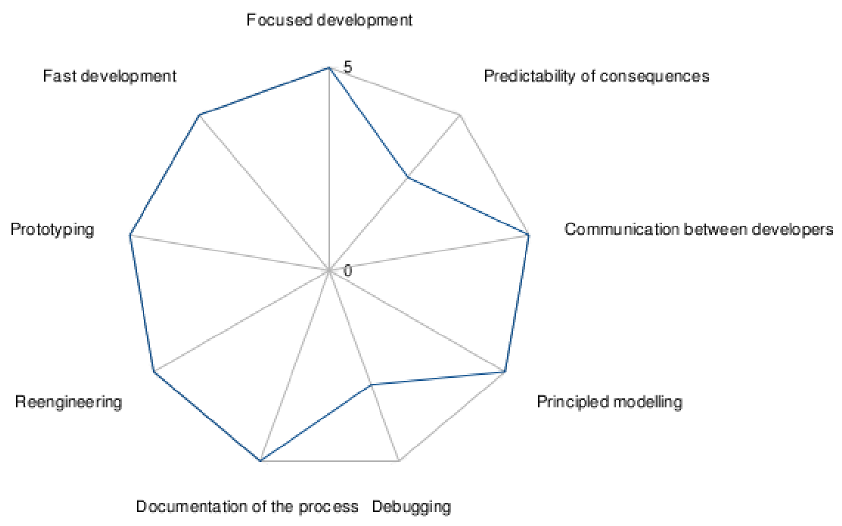


Figure 6.24: Ontology engineering radar graph of the Normalisation ODP. All the criteria get high scores except two criteria that get average scores: predictability of consequences and debugging.

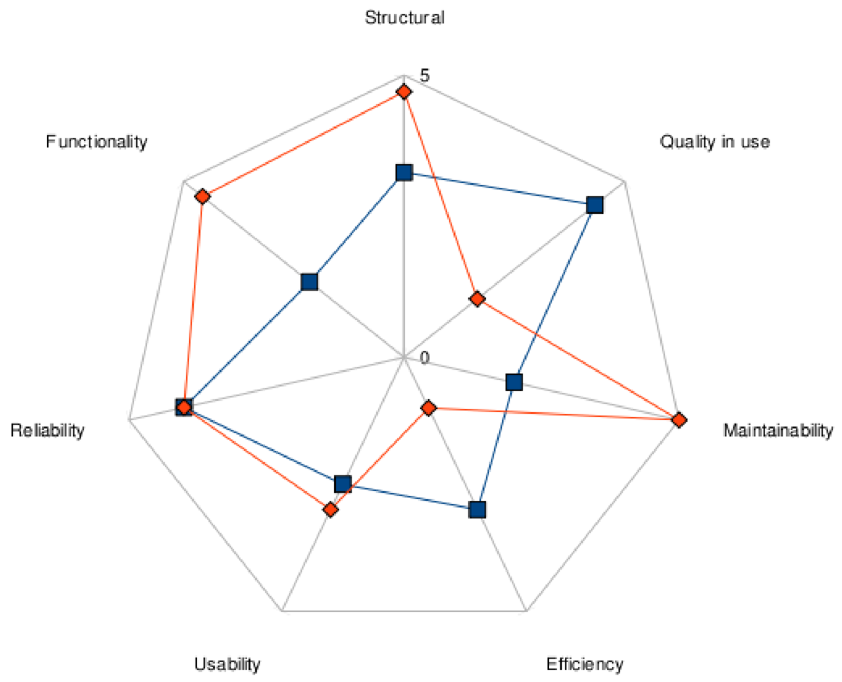
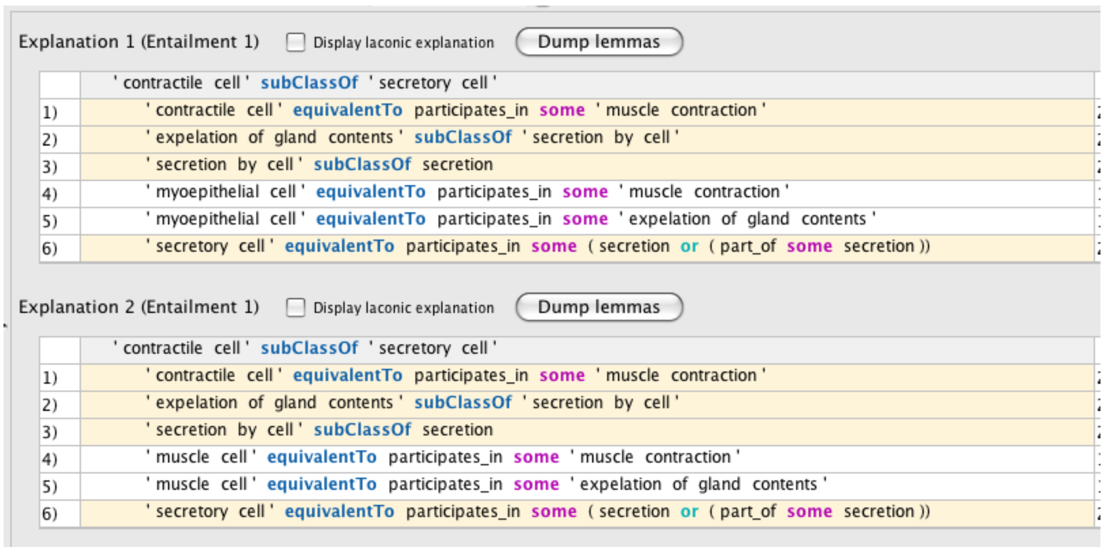


Figure 6.25: Ontology quality radar graph of the Normalisation ODP. The area formed by the diamonds represents the normalised ontology (nCL), and the area formed by the squares the original ontology (CL). CL has a higher quality in the criteria of quality in use and efficiency; in the rest of the areas nCL shows a higher quality, except in the case of reliability were they both get the same score.



```
'contractile cell' subClassOf 'secretory cell'
1) 'contractile cell' equivalentTo participates_in some 'muscle contraction'
2) 'expelation of gland contents' subClassOf 'secretion by cell'
3) 'secretion by cell' subClassOf secretion
4) 'myoepithelial cell' equivalentTo participates_in some 'muscle contraction'
5) 'myoepithelial cell' equivalentTo participates_in some 'expelation of gland contents'
6) 'secretory cell' equivalentTo participates_in some (secretion or (part_of some secretion))
```

```
'contractile cell' subClassOf 'secretory cell'
1) 'contractile cell' equivalentTo participates_in some 'muscle contraction'
2) 'expelation of gland contents' subClassOf 'secretion by cell'
3) 'secretion by cell' subClassOf secretion
4) 'muscle cell' equivalentTo participates_in some 'muscle contraction'
5) 'muscle cell' equivalentTo participates_in some 'expelation of gland contents'
6) 'secretory cell' equivalentTo participates_in some (secretion or (part_of some secretion))
```

Figure 6.26: Justifications of entailed axioms in nCL. When adding axioms to recreate the structure of the intermediate cells some unexpected results were observed. Justifications of the entailments were used as a guide to the axioms causing this unwanted inference.

How can we assess ODP quality? The criteria defined in Chapter 5 have been used to evaluate the quality of the Upper Level Ontology ODP, the Sequence ODP, Entity-Quality ODP, the Selector ODP, and the Normalisation ODP. The rationale used to assign the quality values has been explained.

How can we assess the impact of ODPs in bio-ontology engineering? The criteria defined in Chapter 5 have been used to evaluate how the Upper Level Ontology ODP, the Sequence ODP, Entity-Quality ODP, the Selector ODP, and the Normalisation ODP influence the bio-ontology development process. The rationale used to assign the quality values has been explained.

How does the use of ODPs change the quality of concrete bio-ontologies? The criteria defined in Chapter 5 have been used to evaluate the quality change of CL before and after applying the Normalisation ODP.

Chapter 7

Conclusions

ODPs are best practices of ontology engineering that tackle recurrent modelling issues that arise when building and maintaining ontologies. Therefore ODPs can be used to develop axiomatically rich and rigorous bio-ontologies.

Chapter 2 provided the rationale for the need for ODPs in bio-ontology engineering, and an overview of what ODPs are. In order to provide a technical infrastructure for applying ODPs in OWL bio-ontologies, a public catalogue of ODPs and a scripting language, OPPL, were created during this work, as described in Chapters 3 and 4, respectively. A framework for evaluating ODPs and the change of bio-ontology quality as a result of applying ODPs on them was presented in Chapter 5. The framework was used to evaluate the application of ODPs in bio-ontologies, as described in Chapter 6. This chapter provides the conclusions for the whole research, taking into account the prior chapters.

The chapter is organised as follows. Section 7.1 reviews the research hypothesis and the research questions introduced in Chapter 1, with respect to the overview of ODPs from Chapter 2 and the results of applying ODPs in bio-ontologies from Chapter 6. Section 7.2 expands the contributions described in Chapter 1, taking into account Chapters 2, 3 and 4. Section 7.3 reviews the outstanding issues, thus research items that were planned but not completed, and Section 7.4 suggests items for future research on ODPs for bio-ontologies. Section 7.5 provides an overall conclusion.

7.1 Research hypothesis and research questions revisited

The hypothesis of this thesis is that ODPs facilitate the creation of axiomatically rich and rigorous bio-ontologies. Such a hypothesis can be decomposed into several research questions. Each research question has been answered at a different point of the thesis, and they confirm the hypothesis, as shown by the summary that follows.

7.1.1 What are ODPs?

A working definition of the notion of ODPs was provided in Chapter 2. ODPs are well tested and documented modelling examples that solve concrete modelling problems found when building bio-ontologies. ODPs are discrete fragments of modelling that encapsulate concrete sets of axioms in modelling units with well known features and problems. Therefore, ODPs increase the efficiency of the exploitation of OWL for creating bio-ontologies, by guiding the bio-ontologist on how to use the expressivity and rigour of OWL. ODPs are presented as OWL instances of abstract best practices to ease their uptake and usage by bio-ontologists.

The advantages of the use of ODPs in bio-ontology engineering were also described in Chapter 2. Some of those advantages were empirically confirmed by the results from Chapter 6, revisited as follows:

Design

Rich and granular modelling: nCL presented a higher granularity than CL, as the description of each cell had many more attributes than only `is-a` and `develops-from` relationships, like `ploidy`, `function`, *etc.*, as shown by Table 6.1 and Figure 6.10. Also, after applying the Selector ODP, GO had a higher granularity, as axioms representing the type of regulation (positive or negative) were added, and GO's axiomisation became explicit instead of remaining implicit in the term names.

Robustness: The application of the Selector ODP in GO modified the ontology so as to make it possible to flag errors promptly, and some of the flagged errors were accepted and fixed in the public version of GO by its curators. In the case of nCL, the usage of automated reasoning allowed the exploitation of justifications of entailments for debugging the ontology.

Modularity: By applying the Normalisation ODP, nCL presented a more modular structure than CL. There was only one axis of asserted subsumptions, and the rest of the axes were inferred by the reasoner, resulting in a cleaner model.

Automated reasoning: Both CL and GO have an axiomisation that cannot be used for exploiting automated reasoning. Applying the Normalisation ODP and the Selector ODP allowed the exploitation of automated reasoning in both ontologies, for maintenance and querying.

Implementation

Focused development: The use of the Normalisation ODP allowed the bio-ontologists to focus on deciding the attributes of each cell, instead of deciding the overall structure of the ontology, as shown in Figure 6.24.

Collaborative development: By agreeing to use the Normalisation ODP, the different bio-ontologists could more efficiently collaborate. This is due to the fact that the ontology could be efficiently divided, as each of the ontologists need only pick up a group of cells and add attributes to them.

Prototyping: In the first meeting of the CL experiment a prototype of a normalised ontology was quickly setup by some participants, showing the other bio-ontologists the benefits of such a prototype (maintenance of the multiple inheritance by the reasoner, rich querying, *etc.*).

7.1.2 How can we obtain ODPs?

ODPs, being best practices that guide the bio-ontologists on how to use KR languages, need to be consistently described and centralised in an online resource, therefore allowing the bio-ontologists to efficiently explore them. A public online catalogue of ODPs was presented in Chapter 3, in order to provide such resource.

7.1.3 How can we apply ODPs?

There are different methods for applying ODPs, as described in Chapter 2: manually, OWL imports, Protégé wizards, macros, programmatic application, and Ontology Pre-Processor Language (OPPL).

OPPL has been used to apply ODPs in the use cases of Chapter 6, and has been demonstrated to have an appropriate balance between expressivity and ease of use, as it is based in a well known OWL syntax, MOS. OPPL implements the encapsulation of modelling that ODPs offer to the user by codifying ODPs in OPPL scripts. Therefore, the application of ODPs becomes automatic, traceable and flexible, and ODPs can be shared between developers and applied on different bio-ontologies.

7.1.4 How can we assess ODP quality?

The online catalogue is designed to ease the selection of ODPs. ODPs can be further assessed by using the ODP quality criteria described in Chapter 5: modelling probability, modelling benefit, use efficiency, specificity, documentation clarity, modelling toll, reasoning toll, inconsistency risk, tolerance to bad practice, maintainability, and community commitment. Using such criteria the bio-ontologist can make an informed decision regarding which is the more adequate ODP to apply to the concrete scenario.

The ODPs from the use cases of Chapter 6 (ULO ODP in CCO, Sequence ODP in CCO, Entity-Quality ODP in GO, Selector ODP in GO, Normalisation ODP in CL) were evaluated in terms of such ODP quality, providing orientation for the user on the possible problems associated with the use of those ODPs. From the four ODPs, there were two that got the lowest scores in any of the criteria: the Upper Level Ontology ODP and the Normalisation ODP. The Upper Level Ontology ODP scored low in the criteria of community commitment, tolerance to bad practice and inconsistency risk. The low score in community commitment should be noted, as it can be a serious problem when using this ODP: it requires a strong commitment from the curators on the general modelling of the ontology, something difficult to achieve. The Normalisation ODP scored low in the areas of reasoning toll, modelling toll, and tolerance to bad practice. The low score on reasoning toll should be especially taken into account, as depending on the available resources and the size of an ontology the automated reasoning overload can be considerable.

It can be concluded that, even though ODPs are useful modelling tools, the drawbacks of each ODP should be taken into account when assessing its application in a concrete scenario.

7.1.5 How can we assess the impact of ODPs in bio-ontology engineering?

The use of ODPs changes ontology engineering in different aspects and different ODPs change the same aspect in different ways. Therefore different criteria were presented in Chapter 5 to measure such changes and compare how different ODPs change the engineering process: focused development, fast development, prototyping, reengineering, documentation of the process, communication between developers, predictability of consequences, debugging, and principled modelling.

The ODP that got a lowest score in any of these criteria was the Entity-Quality ODP, as mentioned in Chapter 6. This was due to the fact that the Entity-Quality ODP does not contribute as much as the other ODPs to a more focused development, and therefore potential users of such ODP should consider that negative aspect. Taking into account the rest of the scores, it can be concluded that the use of ODPs makes ontology engineering a more efficient process, demanding less effort from the bio-ontologists.

7.1.6 How can we assess the change of quality of bio-ontologies as a result of applying ODPs?

A framework for evaluating ontology quality was presented in Chapter 5. The framework is an adaptation of the ISO 9126 standard for evaluating software quality. An area of evaluation was added to the framework (structural) to make it more suitable for ontology quality evaluation. The most important feature of the ISO 9126 standard is that it is not designed to provide absolute metrics, therefore allowing the performance of a relative evaluation of two ontologies (before and after applying an ODP).

7.1.7 How does the use of ODPs change the quality of concrete bio-ontologies?

The results of the evaluation presented in Chapter 6 confirm the idea that the use of ODPs contributes to the creation of axiomatically richer and more rigorous bio-ontologies. The quality of CL improved by applying the Normalisation ODP in the following areas: functionality, reliability, maintainability, and structural.

Apart from the improvement in the quality of the bio-ontologies, the application of ODPs brought other more direct results that were accepted by the bio-ontology curators, therefore providing a further demonstration of the usefulness of ODPs in bio-ontology engineering. For example, by applying the Selector ODP, errors were flagged in GO and some of these errors were accepted by the GO curators. As a result of applying the Normalisation ODP, some terms were requested in GO, as the `biological process` subtree from GO was not sufficiently fine grained: such requests were also approved by the GO curators.

7.2 Contributions

The general outcome of this thesis is an infrastructure that allows bio-ontologists to understand, assess, create, publish, share, and apply ODPs in OWL bio-ontologies. The more concrete contributions are reviewed as follows.

7.2.1 Explanation of the concept of ODPs

There is a lack of principled methodologies for bio-ontology engineering. A definition of ODPs was provided, and the advantages of using ODPs reviewed. The idea of ODPs has been presented to the bio-ontologists community, through publications [10, 30, 108] and a tutorial[1].

7.2.2 Catalogue of ODPs

A resource for efficient exploration and sharing of ODPs has been built, in the form of an online catalogue of ODPs[2]. The online catalogue has been used to collect already existing but scattered best practices and for systematically and more thoroughly describing them in the form of ODPs, facilitating their use and comparison. The bio-ontologists can use the online catalogue to explore and retrieve ODPs.

The main feature of the online catalogue is a schema for the description of ODPs, in the form of documentation sections that must be filled in order to consistently describe each ODP. Such schema facilitates the comprehension and exchange of ODPs between bio-ontologists, and it ensures that new ODPs are consistently documented.

The description of each ODP is implemented in annotation values on the OWL file that represents its structure, making it possible to exchange ODPs with their documentation in a single file.

[1]`http://www.co-ode.org/resources/tutorials/bio/`
[2]`http://odps.sf.net/`

7.2.3 OPPL

OPPL can be used for programmatically modifying OWL ontologies by adding or removing axioms. OPPL is also able to perform DL queries, OWL queries about the asserted axioms, and queries on the annotation values to retrieve entities on which axiomatic changes can be performed.

One of the main applications of OPPL is as a means to apply ODPs programmatically in OWL ontologies, and hence consistently and efficiently, by encapsulating ODPs in OPPL scripts. However, OPPL can also be used for modifying and querying OWL ontologies that are too big for manipulation through a graphical interface, or for modifying OWL ontologies as part of automatic pipelines.

7.2.4 Evaluation framework for ontology quality

Ontology quality evaluation is still an open issue. The framework presented in this thesis, based in adapting the ISO 9126 software quality standard to ontology requirements, is a first experimental step towards a more standard approach to the problem.

7.2.5 Improved ontological artefacts

CL, GO, and CCO were improved in the use cases, and the used OPPL scripts are public (CL[3], GO[4] [5], CCO[6]), so anyone can recreate the improvements or adapt the scripts.

7.3 Outstanding issues

7.3.1 Candidate ODPs

In order for a candidate ODP to be added to the online catalogue, it must be tested and thoroughly documented. Therefore, many candidate ODPs were left out of the online catalogue simply due to resources limitations.

Different candidate ODPs have been proposed in the literature for concrete areas of biological knowledge, like species taxonomy [96], chemical functional groups [119],

[3]http://www.gong.manchester.ac.uk/oppl.tar.gz
[4]http://www.gong.manchester.ac.uk/go_selector_odp_experiment.tar.gz
[5]http://www.gong.manchester.ac.uk/OPPL_EKAW2008.tar.gz
[6]http://www.cellcycleontology.org/download/supplementary-files

or different levels of granularity [81]. Other modelling practices, not directly focused on biological knowledge but adaptable to biological knowledge, have been presented, *e.g.* as OWL "Diamonds" [56] and OWL "Intervals" [55]. All those examples are promising best practices that should be suitable for inclusion in the online catalogue.

The addition of new ODPs to the online catalogue should also improve the evaluation of ODP quality, as such evaluation compares ODPs. Therefore, the more ODPs, the more informative the evaluation will be, as the evaluation of each ODP is relative to the rest of ODPs.

7.3.2 Catalogue improvements

In terms of interface, the online catalogue should be more collaborative and dynamic, as the only channel now for proposing ODPs is *via* the mailing list and the forum. An ODP retrieval mechanism should also be devised. Such a mechanism could be based in natural language processing, like the one proposed in [25], or in requirements questions (*e.g. Do you need to represent a parameter with different values?*).

In terms of content, the classification of the ODPs should be done in a more hierarchical, compositional and thorough manner, including different axes of classification, making it possible to explore the online catalogue along such axes. For example, the classification of ODPs according to their exploitation of automated reasoning (dynamic/static ODPs), mentioned in Chapter 3, should be added. Another possible classification to be added to the online catalogue is the classification according to expressivity of the ODP, by using OWL 2 profiles[7].

Also, the documentation schema should be extended with additional sections, such as an OPPL representation of the ODP, the version of the ODP, and the ODP quality and ontology engineering radar graphs. The `references` section should also be improved, by exploiting the fact that OWL 2 allows the annotation of axioms, to represent the references with a finer granularity, *e.g.* by annotating an "article" item with further Dublin Core values.

Finally, an evaluation of the online catalogue should have been defined and performed.

[7]`http://www.w3.org/TR/owl2-profiles/`

7.3.3 Tools

Tooling was planned but not implemented. A Protégé plugin for the creation and documentation of ODPs should be developed. The creation of ODPs should be done either by "recording" the modelling process (a functionality already present in the OPPL 2 Protégé plugin) or using a graphic language like UML or GrOWL. The plugin should be directly connected to the online catalogue, being able to save an ODP in the correct format, and generate forms for the documentation, based on the documentation schema.

Another Protégé plugin, for directly applying ODPs from the online catalogue, should be developed. Such an application should be done either in one step (applying the whole ODP once) or step by step (*e.g.* like in Protégé wizards).

7.3.4 Evaluation framework improvements

The presented evaluation framework can be improved in different aspects. The framework should be refined with more and more precise structural metrics. For example, the work on automatic explanations of entailments in OWL [60] can be adapted to assess the axiomatic complexity of an OWL ontology. Also, more ODPs should be evaluated, in order to refine ODP quality and ontology engineering assessments. Finally, the implementation of the framework should be improved, with questionnaires and guidelines.

7.4 Future work

7.4.1 More ODPs

More ODPs should be created in order to represent biological knowledge in bio-ontologies, *e.g.* ODPs for representing notions like phylogeny, development, temporal aspects, defaults, *etc.* More ODPs should also be devised in order for the community to "synchronise" with the new features and extensions of OWL 2[8].

7.4.2 Definition and representation of ODPs

A community-level agreement is needed for precisely defining the notion of ODPs. Such a definition would make the uptake of ODPs as an engineering technique a faster

[8]`http://www.w3.org/2007/OWL`

process.

A standard definition would also contribute to the graphical representation of ODPs, an issue not yet solved. As mentioned, UML is the most widely used graphical representation for OWL, but different authors use different UML profiles, even though there is an initiative for standardising UML for OWL and RDF[9], through the Ontology Definition Metamodel (ODM) initiative[10]. There are alternatives to UML that are OWL specific, like GrOWL or DLG^2 notation[11], but none of them is sufficiently mature.

7.4.3 ODPs mining

Bio-ontologies can be described in terms of the ODPs that are present in their structure. Therefore, bio-ontologies can be automatically mined to extract ODPs from them and assess their structure in order to find potential alignment areas.

7.5 Overall conclusion

The implantation and use of the Life Sciences Semantic Web (LSSW) is growing, but there are still crucial developments that need to occur for a completely functional LSSW to be in place [6]. Two of those developments are especially important: resolvable universal identifiers for biological entities (*e.g.* Shared Names[12]) and semantically rich bio-ontologies. Semantically rich bio-ontologies widen the range of the interactions of scientists with information, and make such interactions more efficient by relying on computational inferences about disparate information.

In terms of achieving semantically rich bio-ontologies, one of the main problems faced by the bio-ontologists' community is the difficulty of exploiting the power of KR languages like OWL. The use of ODPs helps in the development of bio-ontologies by encapsulating such power in documented modelling units that can be applied in bio-ontologies to solve concrete modelling problems. Therefore, ODPs help in codifying more and more biological information into semantically rich bio-ontologies.

[9] http://www.w3.org/2007/OWL/wiki/UML_Concrete_Syntax
[10] http://www.omg.org/spec/ODM/1.0/Beta3/
[11] http://www.charlestoncore.org/dlg2/
[12] http://neurocommons.org/page/Common_Naming_Project

Bibliography

[1] J.S. Aitken, B.L. Webber, and J.B.L. Bard. Part-of Relations in Anatomy Ontologies: a Proposal for RDFS and OWL Formalisations. In *Proc. PSB*, pages 166–177, 2004.

[2] Stuart Aitken. Formalizing concepts of species, sex and developmental stage in anatomical ontologies. *Bioinformatics*, 21(11):2773–2779, 2005.

[3] Harith Alani and Christopher Brewster. Metrics for ranking ontologies. In *4th Int. EON Workshop, 15th Int. World Wide Web Conf.*, 2006.

[4] Harith Alani, Christopher Brewster, and Nigel Shadbolt. Ranking Ontologies with AKTiveRank. In *The 5th International Semantic Web Conference (ISWC)*. LNCS, 2006.

[5] E Antezana, M Egaña, B De Baets, M Kuiper, and V Mironov. ONTO-PERL: An API supporting the development and analysis of bio-ontologies. *Bioinformatics*, 24(6):885–887, 2008.

[6] Erick Antezana, Ward Blondé, Mikel Egaña, Alistair Rutherford, Robert Stevens, Bernard De Baets, Vladimir Mironov, and Martin Kuiper. Biogateway: A Semantic Systems Biology Tool for the Life Sciences. *BMC bioinformatics*, accepted for publication, 2009.

[7] Erick Antezana, Mikel Egaña, Ward Blondé, 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.

[8] Erick Antezana, Elena Tsiporkova, Vladimir Mironov, and Martin Kuiper. A Cell-Cycle Knowledge Integration Framework (DILS 2006). In *LNBI 4075*, pages 19–34, 2006.

[9] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer, 2nd Edition (Cooperative Information Systems)*. The MIT Press, 2008.

[10] Mikel Egaña Aranguren, Erick Antezana, Martin Kuiper, and Robert Stevens. Ontology Design Patterns for bio-ontologies: a case study on the Cell Cycle Ontology. *BMC bioinformatics*, 9(Suppl 5):S1, 2008.

[11] M. Bada, R. Stevens, C. Goble, Y. Gil, M. Ashburner, J.A. Blake, J.M. Cherry, M. Harris, and S. Lewis. A Short Study on the Success of the Gene Ontology. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(2):235–240, 2004.

[12] Michael Bada and Lawrence Hunter. Identification of OBO nonalignments and its implications for OBO enrichment. *Bioinformatics*, 24(12):1448–1455, 2008.

[13] Christopher J. O. Baker, Arash Shaban-Nejad, Xiao Su, Volker Haarslev, and Greg Butler. Semantic Web infrastructure for fungal enzyme biotechnologists. *Web Semant.*, 4(3):168–180, 2006.

[14] Jonathan Bard, Seung Y Rhee, and Michael Ashburner. An Ontology for Cell Types. *Genome Biology*, 6:R:21, 2005.

[15] S. K. Bechhofer, R. D. Stevens, and P. W. Lord. GOHSE: Ontology driven linking of biology resources. *Web Semant.*, 4(3):155–163, 2006.

[16] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, MAY 2001.

[17] Eva Blomqvist and Kurt Sandkuhl. Patterns in ontology engineering: Classification of ontology patterns. In *ICEIS (3)*, pages 413–416, 2005.

[18] O Bodenreider and R Stevens. Bio-ontologies: current trends and future directions. *Brief. Bioinformatics*, 7(3):256–74, 2006.

[19] Sara Brockmans, Raphael Volz, Andreas Eberhart, and Peter Löffler. Visual Modelling of OWL DL Ontologies using UML. In *Proc. ISWC*, pages 198–213, 2004.

[20] Evelyn Camon, Michele Magrane, Daniel Barrell, Vivian Lee, Emily Dimmer, John Maslen, David Binns, Nicola Harte, Rodrigo Lopez, and Rolf Apweiler.

The Gene Ontology Annotation (GOA) Database: sharing knowledge in Uniprot with Gene Ontology. *Nucleic Acids Res*, 32:D262–D266, Jan 2004.

[21] E.J. Chikofsky and J.H. Cross II. Reverse engineering and design recovery: A taxonomy. *Software Magazine*, January:13–17, 1990.

[22] B.B. Chua and L.E. Dyson. Applying the ISO:9126 model to the evaluation of an e-learning system. In R. Atkinson, C. Mcbeath, D. Jonas-Dwyer, and R. Phillips, editors, *Proceedings of the 21st ASCILITE Conference*, 2004.

[23] Peter Clark, John Thompson, and Bruce Porter. *Knowledge Patterns*, pages 121–134. International Handbooks on Information Systems. Springer, 2003.

[24] John Day-Richter, Midori A A. Harris, Melissa Haendel, and Suzanna Lewis. OBO-Edit–an ontology editor for biologists. *Bioinformatics*, 23(16):2198–2200, 2007.

[25] Guadalupe Aguado de Cea, Asunción Gómez-Pérez, Elena Montiel-Ponsoda, and María del Carmen Suárez-Figueroa. Natural Language-Based Approach for Helping in the Reuse of Ontology Design Patterns. In A. Gangemi and J. Euzenat, editors, *EKAW*, pages 32–47. Springer-Verlag, 2008.

[26] K. Degtyarenko, P. de Matos, M. Ennis, J. Hastings, M. Zbinden, A. McNaught, R. Alcántara, M. Darsow, M. Guedj, and M. Ashburner. ChEBI: a database and ontology for chemical entities of biological interest. *Nucleic Acids Research*, 36:D344–D350, 2008.

[27] Jon Doyle and Ramesh S. Patil. Two theses of knowledge representation: language restrictions, taxonomic classification, and the utility of representation services. *Artif. Intell.*, 48(3):261–297, 1991.

[28] Nicholas Drummond, Alan Rector, Robert Stevens, Georgina Moulton, Matthew Horridge, Hai Wang, and Julian Sedenberg. Putting OWL in Order: Patterns for sequences in OWL. In *OWL Experiences and Directions (OWLEd)*, 2006.

[29] Mikel Egaña, Erick Antezana, and Robert Stevens. Transforming the Axiomisation of Ontologies: The Ontology Pre-Processor Language. In *OWLed DC*, 2008.

[30] Mikel Egaña, Alan Rector, Robert Stevens, and Erick Antezana. Applying Ontology Design Patterns in Bio-ontologies. In A. Gangemi and J. Euzenat, editors, *EKAW 2008, LNCS 5268*, pages 7–16. Springer-Verlag, 2008.

[31] James Farrugia. Model-theoretic semantics for the web. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 29–38. ACM, 2003.

[32] Jesualdo Tomás Fernández-Breis, Mikel Egaña Aranguren, and Robert Stevens. A Quality Evaluation Framework for Bio-Ontologies. In *ICBO*, 2009, accepted.

[33] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison-Wesley, 1995.

[34] A Gangemi, A Gomez-Perez, V Presutti, and Suarez-Figueroa. Towards a Catalog of OWL-based Ontology Design Patterns. In *CAEPIA 07*, 2007.

[35] Aldo Gangemi. Ontology Design Patterns for Semantic Web Content. In *LNCS 1729, ISWC*, pages 262–276, 2005.

[36] Aldo Gangemi, Carola Catenacci, Massimiliano Ciaramita, and Jos Lehmann. Modelling ontology evaluation and validation. In *ESWC*, pages 140–154, 2006.

[37] Yong Gao, June Kinoshita, Elizabeth Wu, Eric Miller, Ryan Lee, Andy Seaborne, Steve Cayzer, and Tim Clark. SWAN: A distributed knowledge infrastructure for Alzheimer disease research. *Web Semant.*, 4(3):222–228, 2006.

[38] D. Gasevic, N. Kaviani, and M. Milanovic. *Handbook on Ontologies*, chapter Ontologies and Software Engineering. Springer, 2008.

[39] Dragan Gasevic, Dragan Djuric, and Vladan Devedzic. *Model Driven Architecture and Ontology Development*. 2006.

[40] Gene Ontology Consortium. Gene Ontology: tool for the unification of biology. *Nature Genetics*, 23(May):25–29, 2000.

[41] Gene Ontology Consortium. The Gene Ontology project in 2008. *Nucleic acids research*, 36(Database issue), 2008.

[42] Andrew Gibson, Katy Wolstencroft, and Robert Stevens. Promotion of ontological comprehension: Exposing terms and metadata with web 2.0. In *WWW*, 2007.

[43] CA Goble and CJ Wroe. The Montagues and the Capulets. *Comparative and Functional Genomics*, 2:623–632, 2004.

[44] Carole Goble and Robert Stevens. State of the nation in data integration for bioinformatics. *Journal of Biomedical Informatics*, 41:687–693, 2008.

[45] Christine Golbreich, Matthew Horridge, Ian Horrocks, Boris Motik, and Rob Shearer. OBO and OWL: Leveraging Semantic Web Technologies for the Life Sciences. In *ISWC/ASWC*, pages 169–182, 2007.

[46] Christine Golbreich and Ian Horrocks. The OBO to OWL Mapping, GO to OWL 1.1! In *OWLED*, 2007.

[47] A. Gomez-Perez and N. Juristo. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. In *Engineering Workshop on Ontological Engineering (AAAI97)*, 1997.

[48] Asuncion Gomez-Perez and Maria Dolores Rojas-Amaya. *Lecture Notes in Computer Science 1621*, chapter Ontological Reengineering for Reuse, pages 139–156. 1999.

[49] Benjamin M. Good and Mark D. Wilkinson. The Life Sciences Semantic Web is Full of Creeps! *Brief Bioinform*, 7(3):275–286, September 2006.

[50] B. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler. OWL 2: The next step for OWL. *Web Semantics: Science, Services and Agents on the World Wide Web*, (6):309–322, 2008.

[51] Pierre Grenon, Barry Smith, and Louis Goldberg. Biodynamic Ontology: Applying BFO in the Biomedical Domain. In D. M. Pisanelli, editor, *Ontologies in Medicine*, pages 20–38. IOS Press, 2004.

[52] Nicola Guarino and Christopher A. Welty. *An Overview of OntoClean*, chapter 8, pages 151–172. Springer, 2004.

[53] Frank W. Hartel, Sherri de Coronado, Robert Dionne, Gilberto Fragoso, and Jeniffer Golbeck. Modeling a Description Logic Vocabulary for Cancer Research. *Journal of Biomedical Informatics*, (38):114–129, 2005.

[54] Robert Hoehndorf, Frank Loebe, Janet Kelso, and Heinrich Herre. Representing default knowledge in biomedical ontologies: Application to the integration of anatomy and phenotype ontologies. *BMC Bioinformatics*, 8(1), 2007.

[55] Rinke Hoekstra. Use of OWL in the Legal Domain. In *OWL: Experiences and Directions (OWLED)*, 2008.

[56] Rinke Hoekstra and Joost Breuker. Polishing Diamonds in OWL 2. In A. Gangemi and J. Euzenat, editors, *EKAW 2008, LNCS 5268*, pages 64–73. Springer-Verlag, 2008.

[57] M Horridge, N Drummond, J Goodwin, A Rector, R Stevens, and H Wang. The Manchester OWL syntax. In *OWLed*, 2006.

[58] Matthew Horridge, Johannes Bauer, Bijan Parsia, and Ulrike Sattler. Understanding Entailments in OWL. In *OWLED*, 2008.

[59] Matthew Horridge, Sean Bechhofer, and Olaf Noppens. Igniting the OWL 1.1 Touch Paper: The OWL API. In Christine Golbreich, Aditya Kalyanpur, and Bijan Parsia, editors, *OWLED*, volume 258 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[60] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and Precise Justifications in OWL. In *ISWC 2008, LNCS 5318*, pages 323–338. Springer-Verlag, 2008.

[61] Luigi Iannone, Mikel Egaña, Alan Rector, and Robert Stevens. Augmenting the Expressivity of the Ontology Pre-Processor Language. In *OWL: Experiences and Directions (OWLED Eu). Poster and short presentation*, 2008.

[62] Luigi Iannone, Alan Rector, and Robert Stevens. Embedding Knowledge Patterns into OWL. In *The Semantic Web: Research and Applications, LNCS 5554*, pages 218–232. Springer-Verlag, 2009.

[63] Eilbeck K., Lewis S.E., Mungall C.J., Yandell M., Stein L., Durbin R., and Ashburner M. The Sequence Ontology: A tool for the unification of genome annotations. *Genome Biology*, (6):R44, 2005.

[64] S. Kerrien, Y. Alam-Faruque, B. Aranda, I. Bancarz, A. Bridge, C. Derow, E. Dimmer, M. Feuermann, A. Friedrichsen, R. Huntley, C. Kohler, J. Khadake, C. Leroy, A. Liban, C. Lieftink, L. Montecchi-Palazzi, S. Orchard, J. Risse, K. Robbe, B. Roechert, D. Thorneycroft, Y. Zhang, R. Apweiler, and H. Hermjakob. IntAct - Open Source Resource for Molecular Interaction Data. *Nucleic Acids Research*, 35:D561–D565, 2007.

[65] P. Khatri and S. Drăghici. Ontological analysis of gene expression data: current tools, limitations, and open problems. *Bioinformatics*, 21(18):3587–3595, September 2005.

[66] Sergey Krivov, Richard Williams, and Ferdinando Villa. GrOWL: A tool for visualization and editing of OWL ontologies. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):54–57, 2007.

[67] Anand Kumar, Barry Smith, and Daniel D. Novotny. Biomedical Informatics and Granularity. *Comparative and Functional Genomics*, 5:501–508, 2004.

[68] Y. Lazebnik. Can a biologist fix a radio? – or, what I learned while studying apoptosis. *Biochemistry (Moscow)*, 69(12):1403–1406, 2004.

[69] Mariano Fernandez López, Asunción Gómez-Pérez, Juan Pazos Sierra, and Alejandro Pazos Sierra. Building a Chemical Ontology Using Methontology and the Ontology Design Environment. *IEEE Intelligent Systems*, 14(1):37–46, 1999.

[70] Joanne Luciano and Robert Stevens. OWL - PAX of mind or the AX that caused the split? In *OWL: Experiences and Directions (OWLED)*, 2008.

[71] Anna Masci, Cecilia Arighi, Alexander Diehl, Anne Lieberman, Chris Mungall, Richard Scheuermann, Barry Smith, and Lindsay Cowell. An improved ontological representation of dendritic cells as a paradigm for all cell types. *BMC Bioinformatics*, 10(1):70, 2009.

[72] Mikel Egaña Aranguren, Sean Bechhofer, Phillip Lord, Ulrike Sattler, and Robert Stevens. Understanding and using the meaning of statements in a bio-ontology: recasting the Gene Ontology in OWL. *BMC Bioinformatics*, 8:57, 2007.

[73] Mikel Egaña Aranguren, Chris Wroe, Carole Goble, and Robert Stevens. In situ migration of handcrafted ontologies to reason-able forms. *Data and Knowledge Engineering*, 66(1):147–162, 2008.

[74] Eleni Mikroyannidi, Alan Rector, and Robert Stevens. Abstracting and Generalising the Foundational Model Anatomy (FMA) Ontology. In *bio-ontologies SIG*, 2009.

[75] C. J. Miller and T. K. Attwood. Bioinformatics goes back to the future. *Nature Rev. Mol. Cell Biol.*, 4.

[76] Boris Motik, Ulrike Sattler, and Rudi Studer. Query Answering for OWL-DL with Rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, JUL 2005.

[77] C.J. Mungall. OBOL: integrating language and meaning in bio-ontologies. *Comparative and Functional Genomics*, 5(6):509–520, 2004.

[78] Eric K. Neumann, Eric Miller, and John Wilbanks. What the Semantic Web could do for Life Sciences. *Biosilico*, 2(6):228–236, 2004.

[79] C. Pasquier. Biological data integration using Semantic Web technologies. *Biochimie*, 2008.

[80] Lennart J. Post, Marco Roos, Scott M. Marshall, Roel Driel, and Timo M. Breit. A Semantic Web approach applied to integrative bioinformatics experimentation: a biological use case with genomics data. *Bioinformatics*, 23(22):3080–3087, 2007.

[81] A.L. Rector, J.E. Rogers, and T. Bittner. Granularity Scale and Collectivity: When Size Does and Doesn't Matter. *Journal of Biomedical informatics*, 539:333–349, 2006.

[82] Alan Rector. Analysis of propagation along transitive roles: Formalisation of the GALEN experience with medical ontologies. In *DL*, 2002.

[83] Alan L. Rector. Modularisation of Domain Ontologies Implemented in Description Logics and related formalisms including OWL. In *K-CAP*, pages 121–128, 2003.

[84] Alan L. Rector and Sebastian Brandt. Why do it the hard way? The Case for an Expressive Description Logic for SNOMED. *Journal of the American Medical Informatics Association : JAMIA*, August 2008.

[85] Alan L. Rector, Nick Drummond, Matthew Horridge, Jeremy Rogers, Holger Knublauch, Robert Stevens, Hai Wang, and Chris Wroe. OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors and Common Patterns. In Enrico Motta, Nigel Shadbolt, Arthur Stutt, and Nicholas Gibbins, editors, *EKAW*, volume 3257 of *Lecture Notes in Computer Science*, pages 63–81. Springer, 2004.

[86] Alan L Rector and Jeremy Rogers. Patterns, Properties and Minimizing Commitment: Reconstruction of the GALEN Upper Ontology in OWL. In *EKAW*, 2004.

[87] Alan L. Rector, Chris Wroe, Jeremy Rogers, and Angus Roberts. Untangling Taxonomies and Relationships: personal and Practical Problems in Loosely Coupled Development of Large Ontologies. In *K-CAP*, pages 139–146, 2001.

[88] Jacqueline Renée Reich. Ontological Design Patterns: Metadata of Molecular Biological Ontologies, Information and Knowledge. In M. Ibrahim, J. Kung, and N. Revell, editors, *DEXA*, pages 698–709, 2000.

[89] J. Rogers, C. Price, A. Rector, W. Solomon, and N. Smejko. Validating Clinical Terminology Structures: Integration and Cross-Validation of Read Thesaurus and GALEN Proc AMIA Symp. *Journal of the American Medical Informatics Association*, pages 845–849, 1998.

[90] Jeremy Rogers and Alan Rector. GALEN's Model of Parts and Wholes: Experience and Comparisons. In *Proc. AMIA symp*, pages 714–718, 2000.

[91] Cornelius Rosse and José L. V. Mejino. A reference ontology for biomedical informatics: the foundational model of anatomy. *J. of Biomedical Informatics*, 36(6):478–500, 2003.

[92] Alan Ruttenberg, Tim Clark, William Bug, Matthias Samwald, Olivier Bodenreider, Helen Chen, Donald Doherty, Kerstin Forsberg, Yong Gao, Vipul Kashyap, June Kinoshita, Joanne Luciano, Scott M. Marshall, Chimezie Ogbuji,

Jonathan Rees, Susie Stephens, Gwendolyn Wong, Elizabeth Wu, Davide Zaccagnini, Tonya Hongsermeier, Eric Neumann, Ivan Herman, and Kei H. Cheung. Advancing translational research with the Semantic Web. *BMC Bioinformatics*, 8(Suppl 3), 2007.

[93] S. Schulz, K. Marko, and U. Hahn. Spatial location and its relevance for terminological inferences in bio-ontologies. *BMC Bioinformatics*, 8(1):134, 2007.

[94] Stefan Schulz and Udo Hahn. Part-whole representation and reasoning in formal biomedical ontologies. *Artif Intell Med*, 34(3):179–200, 2005.

[95] Stefan Schulz, Martin Romacker, and Udo Hahn. Part-Whole Reasoning in Medical Ontologies Revisited - Introducing SEP triplets into Classification-based Description Logics. In *Proceedings of the 1998 AMIA Annual Fall Symposium. A Paradigm Shift in Health Care Information Systems: Clinical Infrastructures for the 21st Century*, pages 830–834. Hanley and Belfus, 1998.

[96] Stefan Schulz, Holger Stenzhorn, and Martin Boeker. The ontology of biological taxa. *Bioinformatics*, 24(13):i313–321, July 2008.

[97] Julian Seidenberg and Alan Rector. Representing Transitive Propagation in OWL. In *LNCS 4215, ER*, 2006.

[98] Stefan Shultz and Udo Hahn. Part-whole representation and reasoning in formal biomedical ontologies. *Artificial Intelligence in Medicine*, 34:179–200, 2005.

[99] Evren Sirin and Bijan Parsia. SPARQL-DL: SPARQL Query for OWL-DL. In *OWLED*, 2007.

[100] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Web Semant.*, 5(2):51–53, 2007.

[101] B Smith, W Ceusters, B Klagges, J Kohler, A Kumar, J Lomax, CJ Mungall, F Neuhaus, A Rector, and C Rosse. Relations in Biomedical Ontologies. *Genome Biology*, 6:R46, 2005.

[102] Barry Smith, Michael Ashburner, Cornelius Rosse, Jonathan Bard, William Bug, Werner Ceusters, Louis J. Goldberg, Karen Eilbeck, Amelia Ireland,

Christopher J. Mungall, Neocles Leontis, Philippe Rocca-Serra, Alan Ruttenberg, Susanna-Assunta Sansone, Richard H. Scheuermann, Nigam Shah, Patricia L. Whetzel, and Suzanna Lewis. The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nat Biotech*, 25(11):1251–1255, November 2007.

[103] Barry Smith, Jacob Köhler, and Anand Kumar. On the Application of Formal Principles to Life Science Data: A Case Study in the Gene Ontology. In *International Workshop on Data Integration in the Life Sciences (DILS 2004)*, Lecture Notes in Computer Science. Springer, March 2004.

[104] Barry Smith and Anand Kumar. Controlled vocabularies in bioinformatics: a case study in the Gene Ontology. *Biosilico*, 2(6):246–252, 2004.

[105] Barry Smith, Jennifer Williams, and Steffen Schulze-Kremer. The Ontology of the Gene Ontology. In *Annual symposium of American Medical Informatics Association (AMIA)*, 2003.

[106] Larisa N. Soldatova and Ross D. King. Are the Current Ontologies in Biology Good Ontologies? *Nature Biotechnology*, 23(9):1095–1098, 2005.

[107] Steffen Staab, Michael Erdmann, and Alexander Maedche. Engineering Ontologies Using Semantic Patterns. In Preece A, and O'Leary, D, editor, *IJCAI*, pages 198–213, 2001.

[108] Robert Stevens, Mikel Egaña Aranguren, Katy Wolstencroft, Ulrike Sattler, Nick Drummond, Matthew Horridge, and Alan Rector. Using OWL to model biological knowledge. *Int. J. Hum.-Comput. Stud.*, 65(7):583–594, 2007.

[109] Robert Stevens and Phillip Lord. *Handbook on Ontologies in Information Systems*, chapter Application of ontologies in bioinformatics. Springer.

[110] Robert Stevens, Chris Wroe, Phillip Lord, and Carole Goble. *Handbook Ontologies in Information Systems*, chapter Ontologies in bioinformatics, pages 635–657. Springer, 2003.

[111] Rudi Studer, Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *DKE*, 25(1-2):161–198, 1998.

[112] Svatek V. Design Patterns for Semantic Web Ontologies: Motivation and Discussion. In *7th Conference on Business Information Systems, Poznan*, 2004.

[113] Samir Tartir and I. Budak Arpinar. Ontology Evaluation and Ranking using OntoQA. In *International Conference on Semantic Computing*, pages 185–192, 2007.

[114] Adolfo Lozano Tello and Asunción Gómez-Pérez. ONTOMETRIC: A Method to Choose the Appropriate Ontology. *J. Database Manag.*, 15(2):1–18, 2004.

[115] Dmitry Tsarkov and Ian Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.

[116] Uniprot Consortium. The Universal Protein Resource (UniProt). *Nucleic Acids Research*, 35(Jan):D193–D197, 2007.

[117] A VanDeursen, P Klint, and Visser J. Domain-specific languages: an annotated bibliography. *ACM SIGPLAN Notices*, 35(6).

[118] N. Villanueva-Rosales and M. Dumontier. yOWL: An ontology-driven knowledge base for yeast biologists. *Journal of biomedical informatics*, 2008.

[119] Natalia Villanueva-Rosales and Michel Dumontier. Describing Chemical Functional Groups in OWL-DL for the Classification of Chemical Compounds. In Christine Golbreich, Aditya Kalyanpur, and Bijan Parsia, editors, *OWLED*, volume 258 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[120] Johanna Völker, Denny Vrandecic, and York Sure. Automatic Evaluation of Ontologies (AEON). In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors, *Proceedings of the 4th International Semantic Web Conference (ISWC2005)*, volume 3729 of *LNCS*, pages 716–731. Springer Verlag Berlin-Heidelberg, 2005.

[121] Denny Vrandečić. Explicit Knowledge Engineering Patterns with Macros. In Chris Welty and Aldo Gangemi, editors, *Ontology Patterns for the Semantic Web Workshop (ISWC)*, 2005.

[122] Denny Vrandecic and York Sure. How to design better ontology metrics. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, *The Semantic Web: Research and Applications — Proceedings of the 4th European Semantic Web Conference (ESWC 2007)*, number 4519 in Lecture Notes in Computer Science, pages 311–325. Springer, 2007.

[123] C. Welty, M. Gruninger, F. Lehmann, D. McGuinness, and M. Uschold. Ontologies: Expert systems all over again? In *AAAI-1999 Invited Panel.*, 1999.

[124] John Wilbanks and William Neal. Introduction to science commons. *www.sciencecommons.org*, 2006.

[125] K. Wolstencroft, R. Mcentire, R. Stevens, L. Tabernero, and A. Brass. Constructing ontology-driven protein family databases. *Bioinformatics*, 21(8):1685–1692, 2005.

[126] Chris J. Wroe, James J. Cimino, and Alan L. Rector. Integrating Existing Drug Formulation Terminologies Into an HL7 Standard Classification using OpenGALEN. In *Annual Fall Symposium of American Medical Informatics Association*, Washington DC., November 2001.

[127] Iwei Yeh, Peter D. Karp, Natasha F. Noy, and Russ B. Altman. Knowledge acquisition, consistency checking and concurrency control for Gene Ontology (GO). *Bioinformatics*, 19(2):241–248, 2003.

[128] A. Young, N. Whitehouse, J. Cho, and C. Shaw. OntologyTraverser: an R package for GO analysis. *Bioinformatics*, 21(2):275–276, 2005.

[129] A. C. Yu. Methods in biomedical ontology. *Journal of Biomedical Informatics*, 39(3):252–266, June 2006.

Appendix A

The Catalogue of Ontology Design Patterns

A.1 Adapted SEP ODP

ALSO KNOWN AS: Transitive propagator.

CLASSIFICATION: Domain Modelling.

MOTIVATION: In the biomedical domain the propagation of properties along the partonomy relation is very important. For example, there are cases where the fault of the part should be assumed to be a fault of the whole (an appendix perforation is an intestine perforation) and other cases where it should not be considered like that (appendicitis is not enteritis). The problem of propagating properties along partonomy relates directly to the problem of (for example) overloading PartOf in the Gene Ontology: for example Location, a property that should propagate along (or not) with PartOf , is always implicitly present anywhere there is a PartOf relation. For example Polarisome is PartOf CellCortex and PartOf SiteOfPolarizedGrowth, inheriting both locations, creating a conflict: polarisome is not located in the whole of the cell cortex, is only located in the cell cortex in the site of polarised growth.

AIM: To model selective transitive propagation.

STRUCTURE: See Figure A.1.

SAMPLE: See Figure A.2.

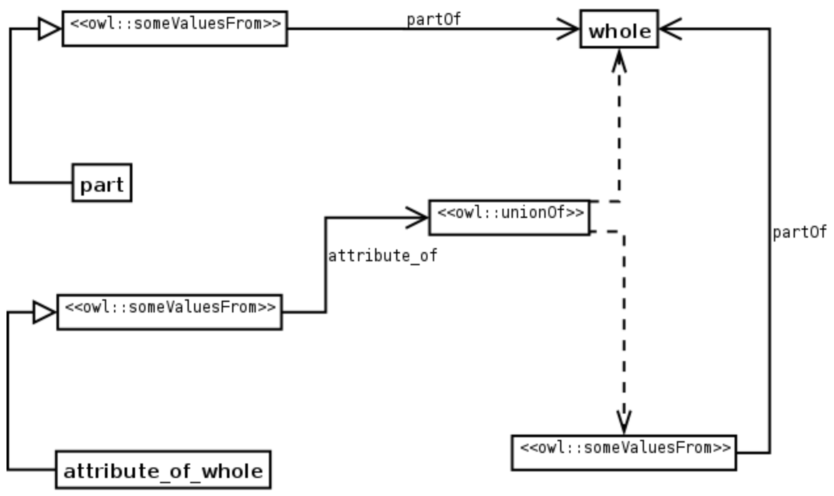


Figure A.1: Abstract structure of the Adapted SEP ODP.

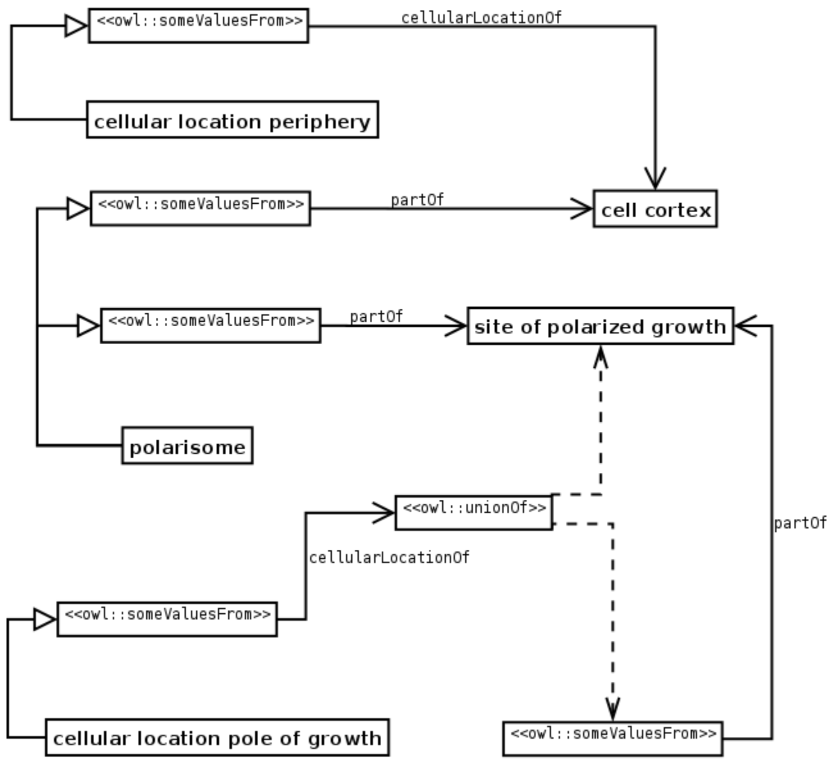


Figure A.2: Sample structure of the Adapted SEP ODP.

ELEMENTS: The elements of the partonomy hierarchy are maintained and in this case two new elements are added to represent concrete locations in the cell (CellularLocationPole and CellularLocationPeriphery). The PartOf relationship is maintained (defined as transitive) and in this case a new object property is added to link locations with cellular components, CellularLocationOf.

IMPLEMENTATION: The most important step is to define the class CellularLocationPoleOfGrowth as the location of SiteOfPolarizedGrowth or any of its parts, so the location is propagated to the parts (but it is not propagated in the case of CellCortex).

RESULT: The location property CellularLocationOf is propagated along PartOf in a selective way, allowing for a precise and unambiguous definition of the polarisome location. To check the result two classes can be created: PolarisomeLocation [partial CellularLocationOf some Polarisome] and SiteOfPolarisedGrowthLocation [complete cellularLocationOf some (SiteOfPolarisedGrowth and PartOf some SiteOfPolarisedGrowth)]. After reasoning PolarisomeLocation should be a subclass of SiteOfPolarisedGrowthLocation.

ADDITIONAL INFORMATION: There have been different proposals in the literature for modelling transitive propagation in the biomedical domain. The approach chosen for this ODP relies on the possibility of creating transitive object properties given by OWL DL. Another approach is the one described by Stefan Shultz and Udo Hahn (see references), which relies in simulating the transitivity by creating SEP triples (Structure - Entity - Part) for each class of the partonomy hierarchy, allowing for selective inheritance of properties.

REFERENCES:

- Alan L Rector and Sebastian Brandt. Why do it the hard way? The Case for an Expressive Description Logic for SNOMED. JAMIA (28 August 2008).
- `http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/`
- Julian Seidenberg, Alan Rector. ”Representing transitive propagation in OWL”. ER2006.
- Stefan Shultz and Udo Hahn. Part-whole representation and reasoning in formal biomedical ontologies. Artificial Intelligence in Medicine, 34: 179-200, 2005.

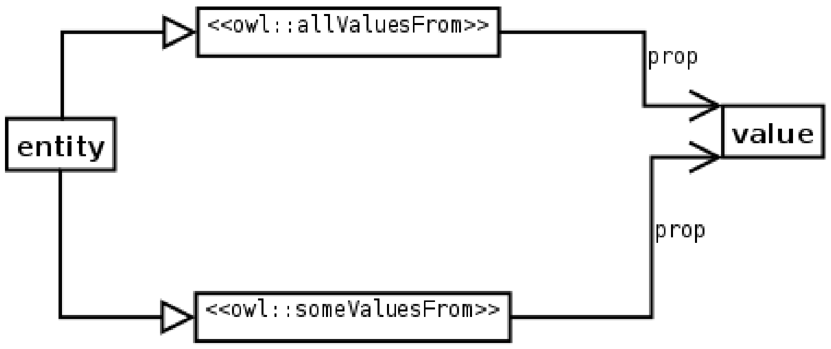


Figure A.3: Abstract structure of the Closure ODP.

URL: Adapted_SEP.owl

A.2 Closure ODP

CLASSIFICATION: Good Practice.

MOTIVATION: OWL sometimes is anti-intuitive due to the Open World Assumption. One of the examples of such problem is the fact that plenty of users think that asserting an existential restriction is enough to close a relationship, when in fact a universal restriction is also needed: it is not enough to say that carnivore eats some meat, as that is equivalent to saying that it can eat another things apart of meat.

AIM: Simulate the closed world assumption in a concrete class.

STRUCTURE: See Figure A.3.

SAMPLE: See Figure A.4.

ELEMENTS: The only element to take into account is the object property that will be used to produce the closure.

IMPLEMENTATION: The only necessary step is to add an existential restriction and an universal restriction with the same filler.

RESULT: The closure axiom allows to close the world and express that something has got a property and only that property. For example, following the example, without the closure (without the universal restriction) carnivore and herbivore

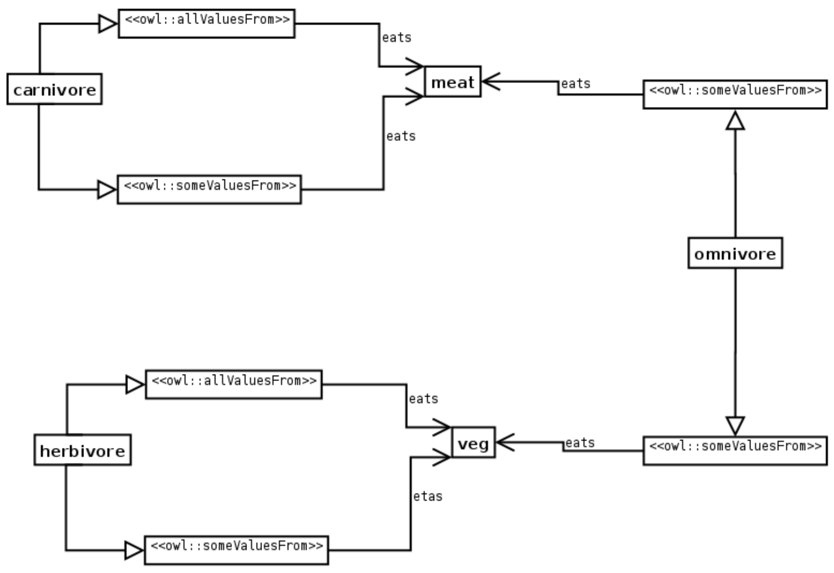


Figure A.4: Sample structure of the Closure ODP.

would appear as subclasses of omnivore. However, with the closure axiom, they do not.

REFERENCES:

- Explicit Knowledge Engineering Patterns with Macros. Denny Vrandecic. In Proceedings of the Ontology Patterns for the Semantic Web Workshop (ISWC 2005).
- Alan Rector, Nick Drummond, Matthew Horridge, Jeremy Rogers, Holger Knublauch, Robert Stevens, Hai Wang, Chris Wroe. OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors and Common Patterns. In Proceedings of the European Conference on Knowledge Acquistion, 2004. LNCS- LNAI 3257, Springer-Verlag.pp 63-81.

URL: Closure.owl

A.3 Composite Property Chain ODP

CLASSIFICATION: Domain Modelling.

MOTIVATION: A composite chain can be appreciated by the following example: the son of the brother of my father is my cousin. The same structure can be

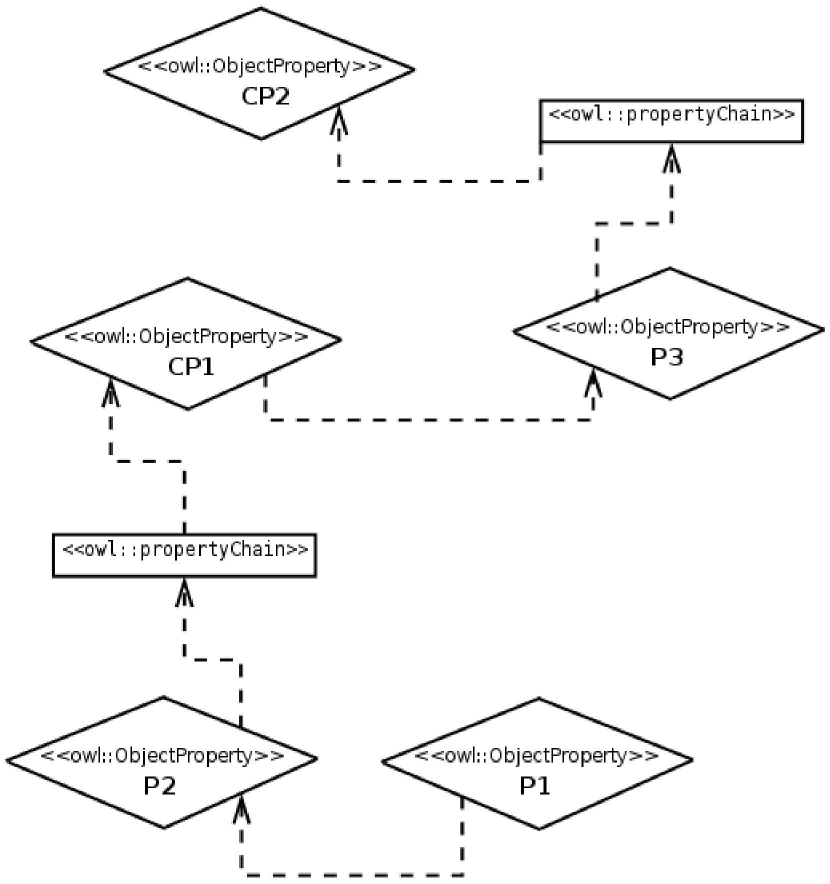


Figure A.5: Abstract structure of the Composite Property Chain ODP.

applied for modelling, for example, the sucessive modifications that a protein goes through. The key on the composite chain is that there are two chains, but one of them is composed by a relationship that will be inferred by the reasoner: the reasoner will first infer that the brother of my father is my uncle (first chain: father + brother = uncle), and then that the son of my uncle is my cousin (second chain: uncle + son = cousin). The property uncle is common to both chains.

AIM: To model a double chain of properties, i.e. two chains that link four individuals.

STRUCTURE: See Figure A.5.

SAMPLE: See Figure A.6.

ELEMENTS: This ODP is made by five object properties, grouped in two chains. Both chains have one object property in common: in one of them it is the head and in the other it is one of the precedents.

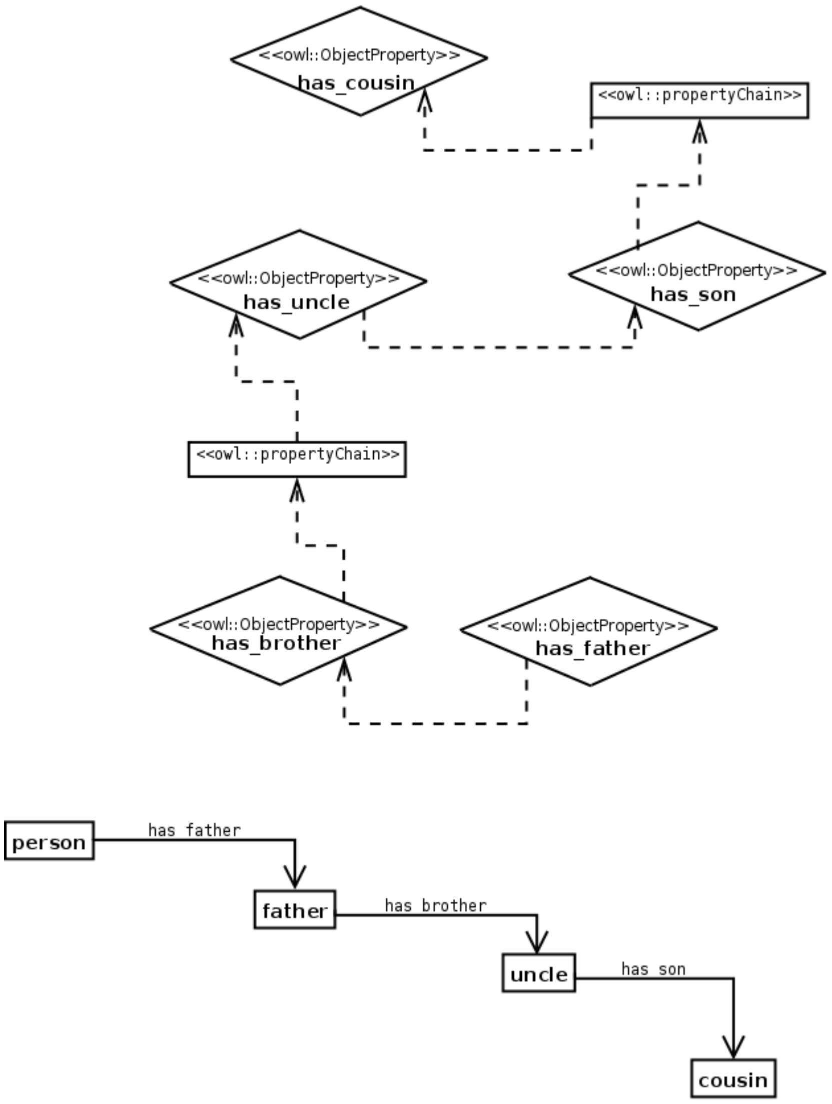


Figure A.6: Sample structure of the Composite Property Chain ODP.

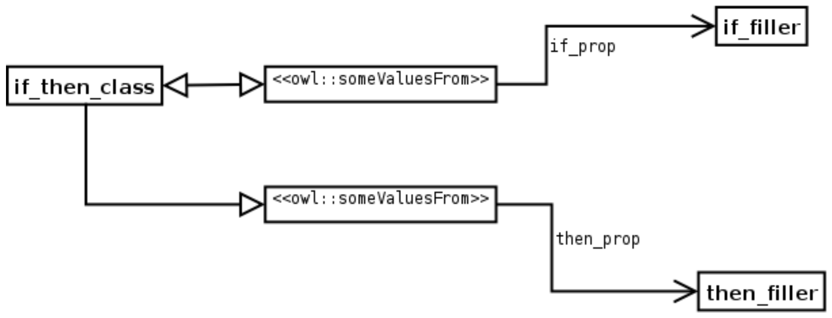


Figure A.7: Abstract structure of the Defined Class Description ODP.

IMPLEMENTATION: The only main step of this ODP is to create both chains, and to link the appropriate individuals.

RESULT: The double chain is modelled. This allows for queries with the composite properties (e.g. has_uncle and has_cousin).

REFERENCES:

- `http://odps.sourceforge.net`

URL: Composite_Property_Chain.owl

A.4 Defined Class Description ODP

CLASSIFICATION: Good Practice.

MOTIVATION: If-Then structures are very common and intuitive and this ODP offers the possibility of representing them within OWL DL expressivity.

AIM: To simulate an If-Then of the type: if something fullfills certain conditions, it should have a further given attribute.

STRUCTURE: See Figure A.7.

SAMPLE: See Figure A.8.

ELEMENTS: the important elements are the class that is being used to simulate the rule and the properties that are used in the condition (the equivalent restrictions) and the conclusion (the neccesary restrictions).

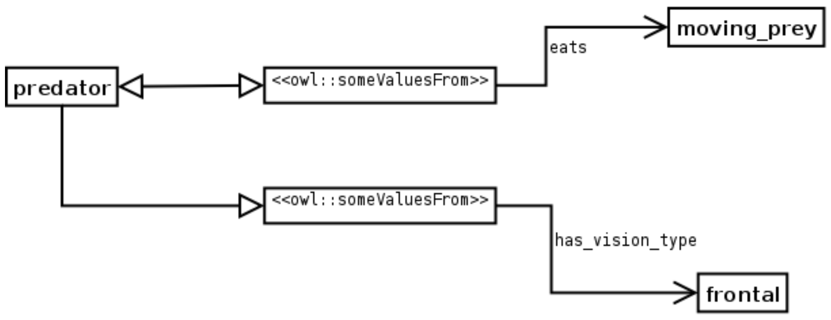


Figure A.8: Sample structure of the Defined Class Description ODP.

RESULT: The If-Then rule is represented in the ontology and can be used, for example, when adding new classes and performing reasoning: if a class fulfill the If condition, it will have also the Then attribute.

REFERENCES:

- Alan Rector, Nick Drummond, Matthew Horridge, Jeremy Rogers, Holger Knublauch, Robert Stevens, Hai Wang, Chris Wroe. OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors and Common Patterns. In Proceedings of the European Conference on Knowledge Acquistion, 2004. LNCS- LNAI 3257, Springer-Verlag.pp 63-81.

URL: DefinedClass_Description.owl

A.5 Entity-Feature-Value ODP

CLASSIFICATION: Good Practice.

MOTIVATION: This ODP is used to represent modifiers with multiple aspects, thus features (e.g. colour with a certain brightness and saturation).

AIM: To model features with the simplest structure possible.

STRUCTURE: See Figure A.9.

SAMPLE: See Figure A.10.

ELEMENTS: The most important elements are the object properties (one for each aspect), the feature, and the values of the aspects.

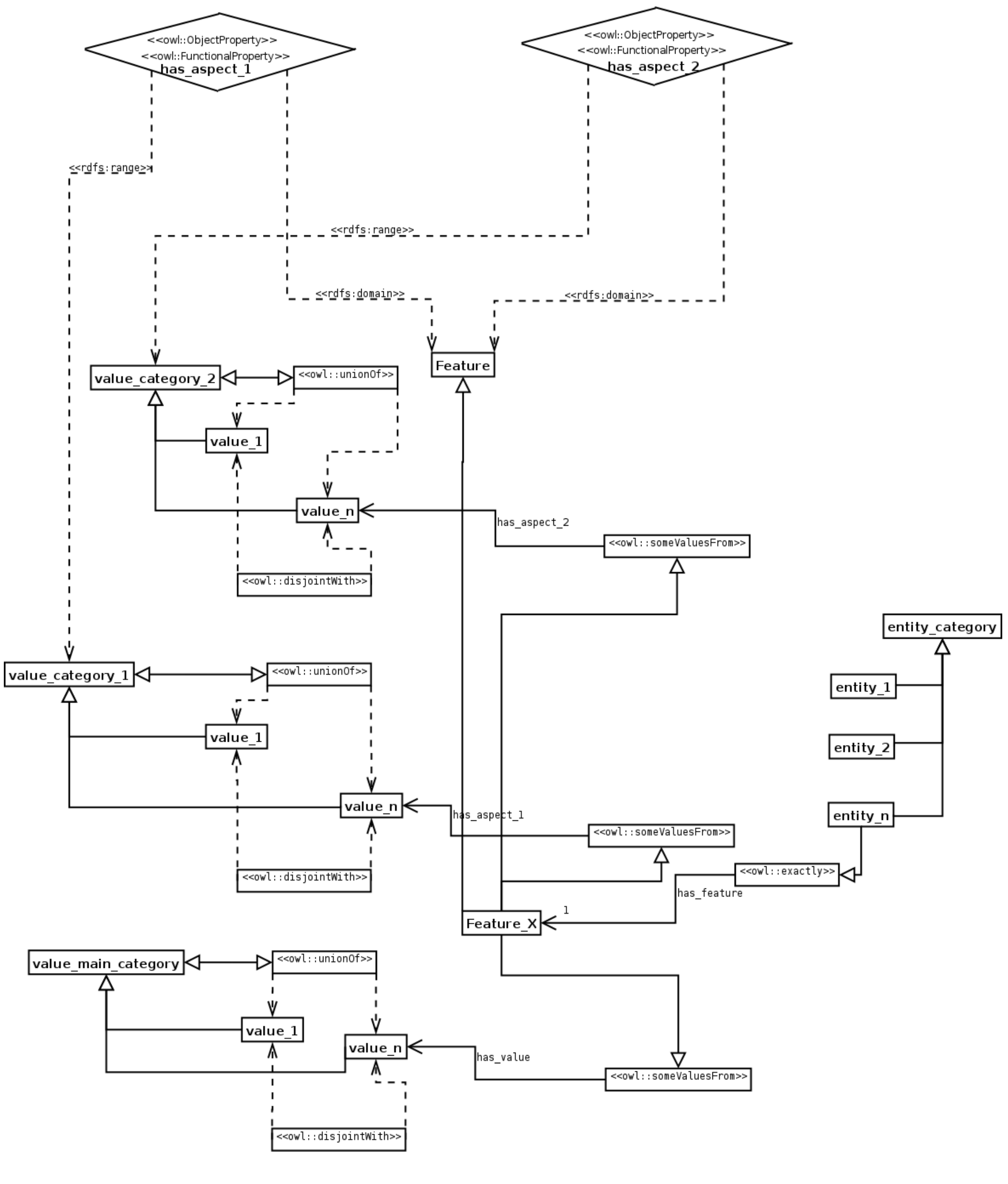


Figure A.9: Abstract structure of the Entity-Feature-Value ODP.

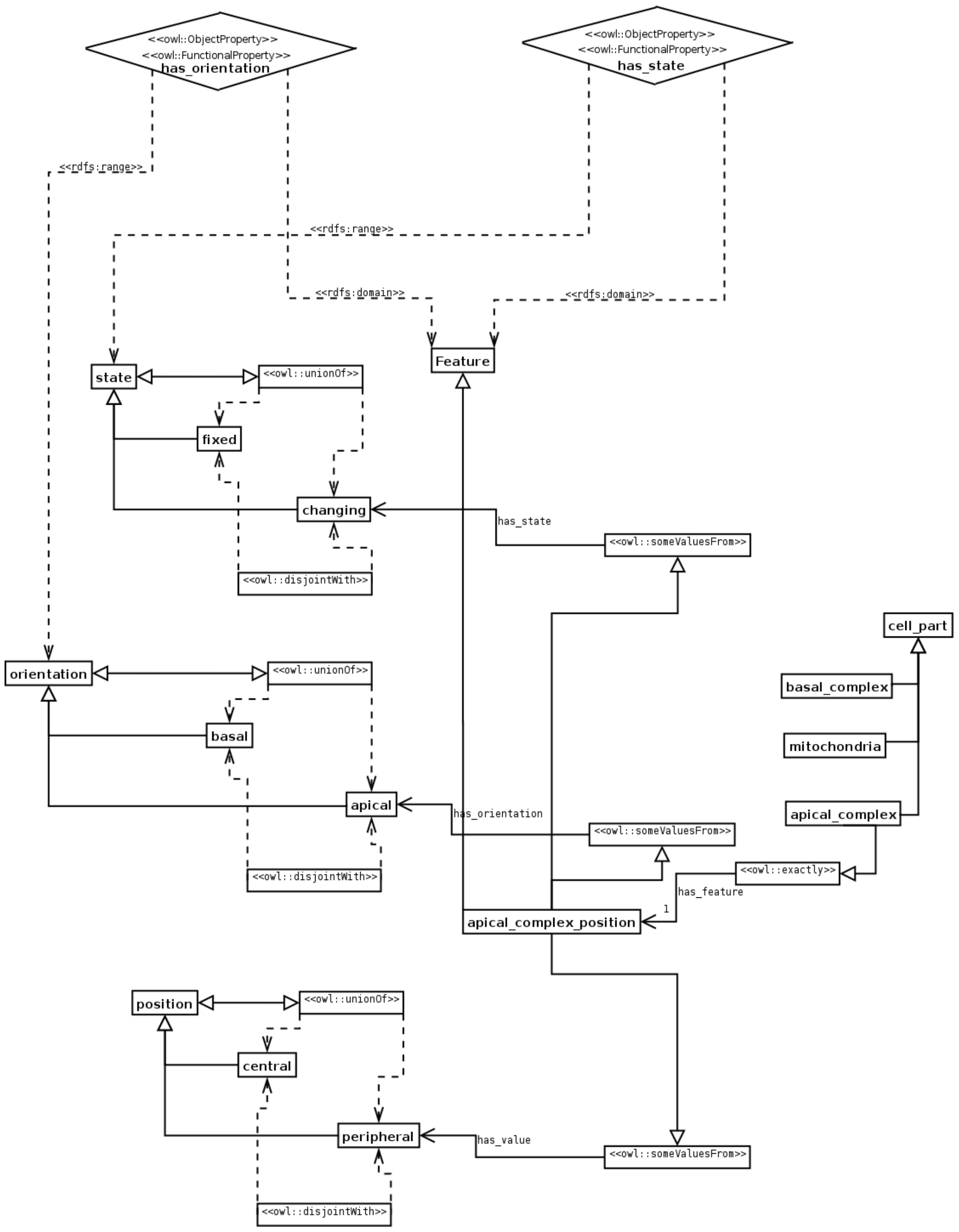


Figure A.10: Sample structure of the Entity-Feature-Value ODP.

IMPLEMENTATION: For each aspect of the feature, an object property and a value partition should be created. For each of them, the domain should be the class Feature and the range the aspect value. The feature is really an Nary relationship ODP, and it is linked to each aspect by existential restrictions. The entity is linked to the feature with a Qualified Cardinality Restriction (QCR) of exactly one.

RESULT: The entities, que features of those entities and the aspects of the features are properly separated.

SIDE EFFECTS: Although this ODP can obviously handle multi aspect qualities, it is difficult to author because of the amount of entities that need to be added.

ADDITIONAL INFORMATION: See also the Entity-Quality ODP and the Entity-Property-Value ODP.

REFERENCES:

- Alan Rector (Personal Communication).
- A. Gangemi, N. Guarino, C. Masolo, A. Oltramari and L. Schneider. Sweetening ontologies with dolce. In: LNCS, EKAW. (2002) 166-182.
- Mikel Egana, Alan Rector, Robert Stevens and Erick Antezana. Applying Ontology Design Patterns in bio-ontologies. EKAW 2008. LNCS 5268, pp. 7-16, 2008.

URL: Entity_Feature_Value.owl

A.6 Entity-Property-Quality ODP

CLASSIFICATION: Good Practice.

MOTIVATION: Qualities (modifiers) are refining entities, thus refine or modify the description of another (independent) entity. They are very important in many domains. They should not be confused with selectors (e.g. left hand), although both modifiers and selectors are refining entities.

AIM: To model qualities of independent entities (e.g. position, colour, ...).

STRUCTURE: See Figure A.11.

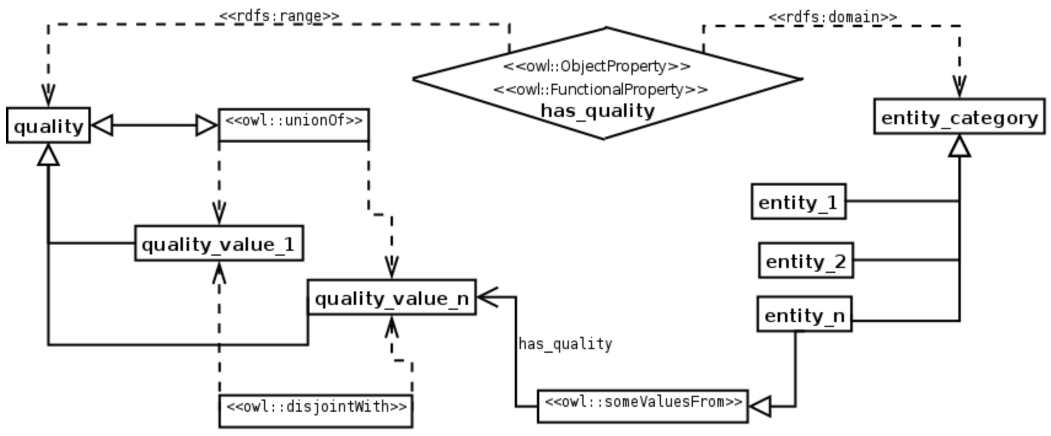


Figure A.11: Abstract structure of the Entity-Property-Quality ODP.

SAMPLE: See Figure A.12.

ELEMENTS: The core of this ODP is formed by the qualities, placed in a single hierarchy (the qualities are disjoint and the superclass is covered by them, like in the Value Partition ODP). Entities are linked to qualities by a functional object property whose domain and range are the entities and the qualities, respectively.

IMPLEMENTATION: The first step is to create the qualities hierarchy, in the same way as the Value Partition ODP. Create the functional object property to link entities to qualities, adding the entities as domain and quality as range. Link entities to qualities by existential restrictions.

RESULT: The qualities that modify independent entities are modelled, and which qualities apply to which entities is defined.

SIDE EFFECTS: Proliferation of object properties (one for each quality). This ODP Cannot handle multi-aspect qualities (features).

ADDITIONAL INFORMATION: See also Entity-Feature-Value ODP and Entity-Quality ODP.

REFERENCES:

- Alan Rector (Personal Communication).

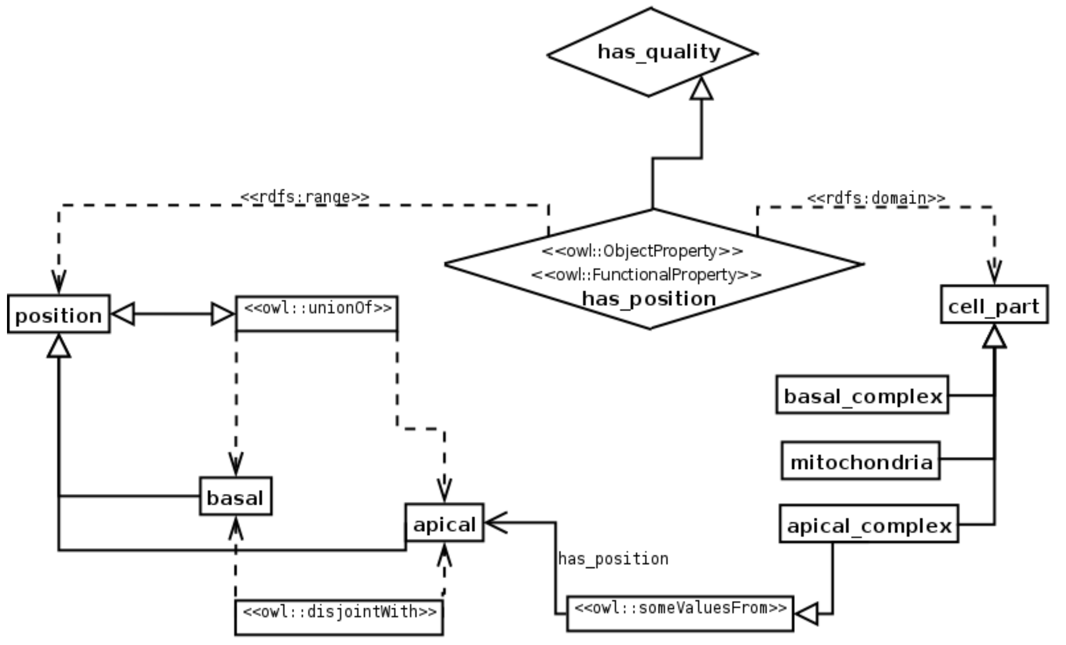


Figure A.12: Sample structure of the Entity-Property-Quality ODP.

- Mikel Egana, Alan Rector, Robert Stevens and Erick Antezana. Applying Ontology Design Patterns in bio-ontologies. EKAW 2008. LNCS 5268, pp. 7-16, 2008.

URL: Entity_Property_Quality.owl

A.7 Entity-Quality ODP

CLASSIFICATION: Good Practice.

MOTIVATION: Qualities modify independent entities (e.g. position, colour, etc.) and thus they are dependent entities.

AIM: To model qualities without relying in a proliferation of object properties, as in the Entity-Property-Quality ODP.

STRUCTURE: See Figure A.13.

SAMPLE: See Figure A.14.

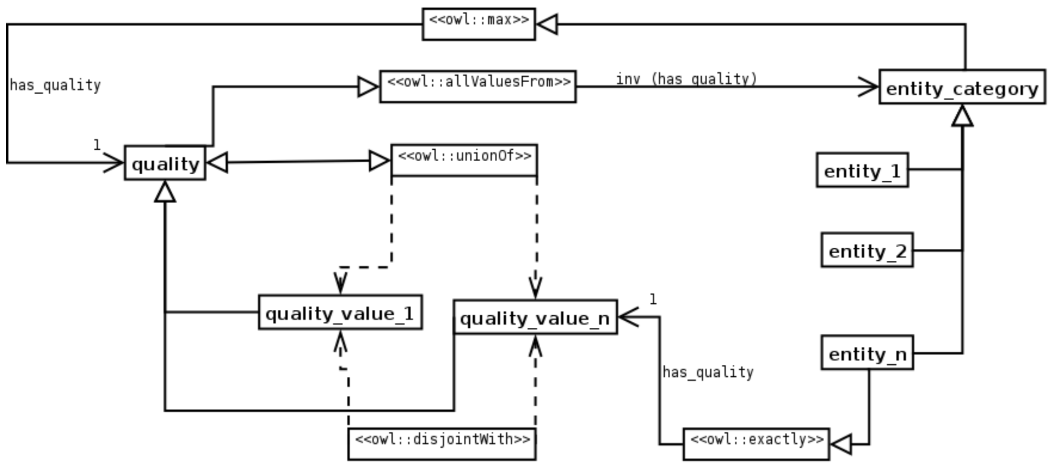


Figure A.13: Abstract structure of the Entity-Quality ODP.

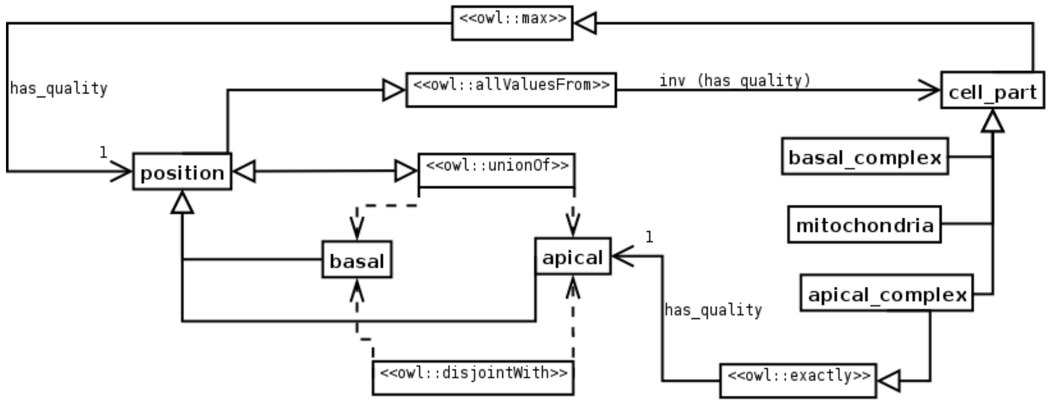


Figure A.14: Sample structure of the Entity-Quality ODP.

ELEMENTS: The core of this ODP is formed by the qualities, placed in a single hierarchy (the qualities are disjoint and the superclass is covered by them, like in the Value Partition ODP). Entities are linked to qualities by an object property, and a Qualifed Cardinality Restriction is used to express whether the quality is intrisic (exactly 1) or accidental (max 1). Also, qualities are limited to the entities to which they apply by an universal restriction (e.g. mitochondria do not regulate mitosis).

IMPLEMENTATION: The first step is to create the qualities hierarchy, in the same way as the Value Partition ODP. Create the object property to link entities to qualities. Add the restriction [QualityCategory inv (HasQuality) only Entity] (this restricts the qualities to the entity). Add the restriction [EntityCategory HasQuality max 1 QualityCategory] or [EntityCategory HasQuality exactly 1 QualityCategory] (this restricts the entities to the qualities, max 1 in the case of accidental qualities and exactly 1 in the case of intrinsic qualities).

RESULT: The entities and the qualities of those entities are properly separated, and which qualities apply to which entities is also expressed.

SIDE EFFECTS: it is very difficult to add sub-qualities. Cannot handle multi-aspect qualities.

ADDITIONAL INFORMATION: See also Entity-Feature-Value and Entity-Property-Value.

REFERENCES:

- P. Grenon, B. Smith, L. Goldberg. Biodynamic ontology: Applying BFO in the biomedical domain. In Pisanelli, D.M., ed.: Ontologies in Medicine, IOS Press (2004) 20-38.
- Alan Rector (Personal Communication).
- Mikel Egana, Alan Rector, Robert Stevens and Erick Antezana. Applying Ontology Design Patterns in bio-ontologies. EKAW 2008. LNCS 5268, pp. 7-16, 2008.

URL: Entity_Quality.owl

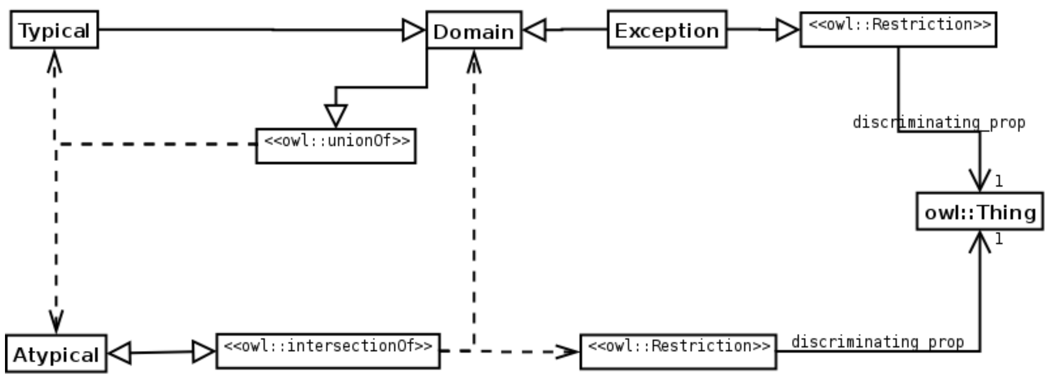


Figure A.15: Abstract structure of the Exception ODP.

A.8 Exception ODP

CLASSIFICATION: Extension.

MOTIVATION: Plenty of areas of knowledge work with defaults or canonical knowledge: biological classifications, for example, state what is the canonical norm and then the exceptions are classified under the norm, even if the classification is inconsistent from the point of view of logic. A clear example can be found in the classification of cells: in canonical biology eukaryotic cells are considered to be cells with a nucleus. Mammalian red blood cells are considered by any biologist as eukaryotic cells, but they lack a nucleus. Thus they are a subclass of eukaryotic cells, but they break the condition for belonging to that class (having a nucleus).

AIM: to model exceptions without breaking the strict class-subclass hierarchy: for example the class MammalianRedBloodCell (with the restriction HasNucleus exactly 0) would be a subclass of EukaryoticCell (with the restriction HasNucleus exactly 1), resulting in an inconsistent ontology. There can be exceptions to the exception in the next level: avian red blood cells do posses a nucleus, thus, they are considered normal eukaryotic cells (they are an exception to the norm that all red blood cells lack a nucleus). So the problem that this ODP solves can rise in different levels.

STRUCTURE: See Figure A.15.

SAMPLE: See Figure A.16.

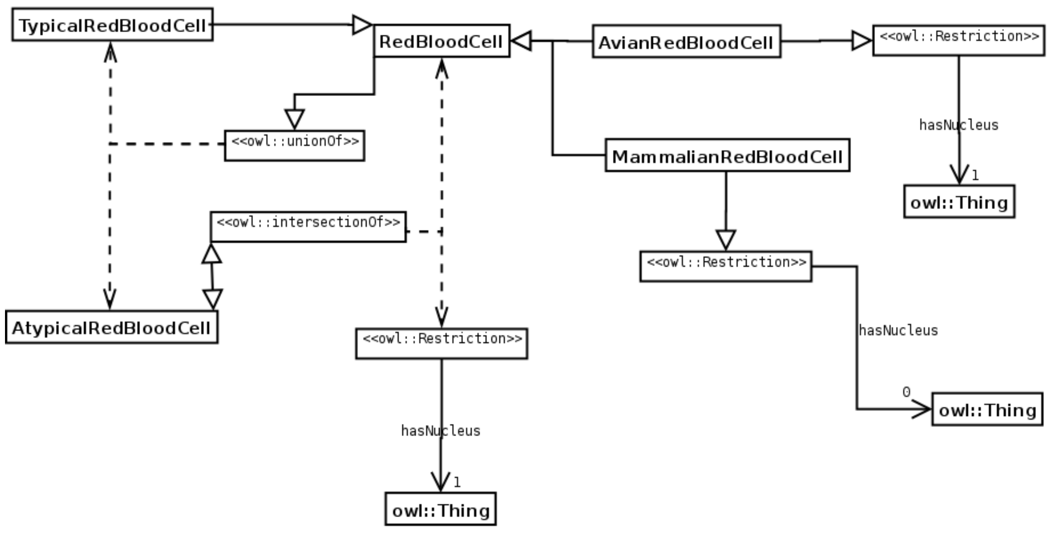


Figure A.16: Sample structure of the Exception ODP.

ELEMENTS: The most important elements are the newly created TypicalEukaryoticCell, TypicalRedBloodCell, AtypicalEukaryoticCell, AtypicalRedBloodCell classes. The rest of the classes are maintained. The most important object property is the discriminating property, in this case, HasNucleus.

IMPLEMENTATION: Starting from the example ontology described in the Aim section, two disjoint classes are created for typical and atypical elements. The discriminating condition HasNucleus is only stated in the typical subclass. A covering axiom is added to the main class (i.e. EukaryoticCell) to state that all instances must belong to one or the other subclass (TypicalEukaryoticCell or AtypicalEukaryoticCell). A covering axiom is done by creating a equivalent class (a neccesary and sufficient condition) that is the union of the subclasses (In this case TypicalEukaryoticCell and AtypicalEukaryoticCell). The reasoner will infer the whole structure.

RESULT: After reasoning the correct hierarchy is obtained, with the typical/atypical distinction at every level.

SIDE EFFECTS: If the ODP is used in plenty of different levels of the ontology it can produce too complex and unmanageable ontologies. This type of structure can be very counter-intuitive for biologists.

REFERENCES:

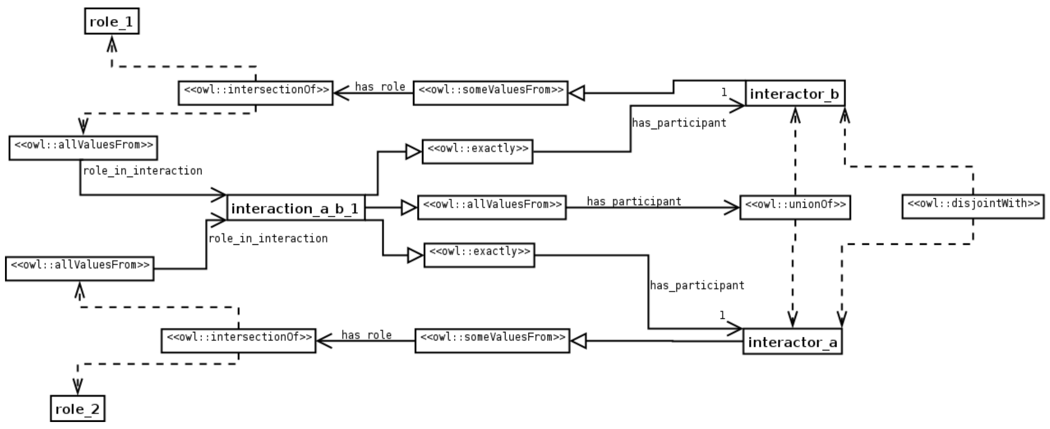


Figure A.17: Abstract structure of the Interactor Role Interaction ODP.

- `http://www.co-ode.org/resources/tutorials/bio/`
- Robert Stevens, Mikel Egana Aranguren, Katy Wolnstencroft, Ulrike Sattler, Nick Drummond and Mathew Horridge. Using OWL to Model Biological Knowledge. International Journal of Human Computer Studies 2006, 65:7, 583-594.

URL: Exception.owl

A.9 Interactor Role Interaction ODP

CLASSIFICATION: Domain Modelling.

MOTIVATION: Protein Protein Interactions (PPI) are the base for most of the biological processes at a molecular level. For example (http://www.ebi.ac.uk/intact/binary-search/faces/search.xhtml?query=BRCA2).

AIM: To model different interactions where the interactors can have different roles.

STRUCTURE: See Figure A.17.

SAMPLE: See Figure A.18.

ELEMENTS: Three object properties are needed: HasRole, RoleInInteraction, and HasParticipant. This ODP has got two aims: close the interactors an interaction can have, and decouple roles from interactions. An interaction is a unique combination of interactors and roles, whereas an interactor can have at the same

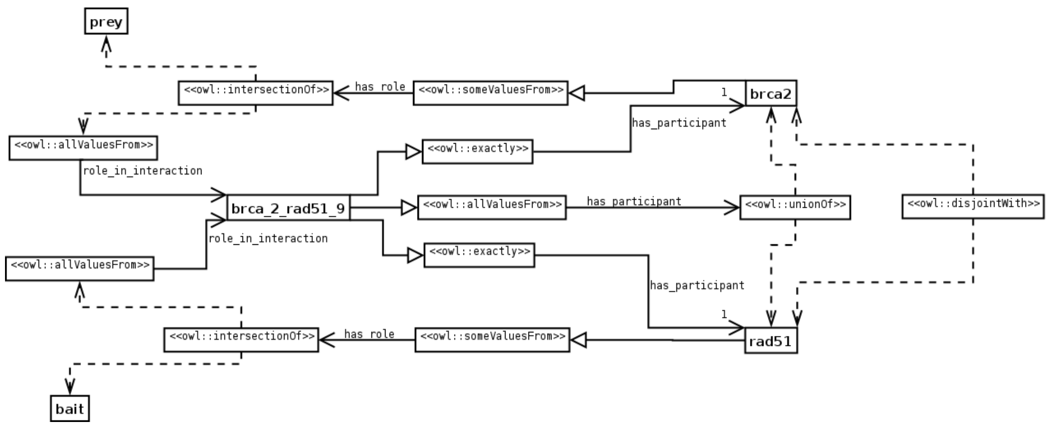


Figure A.18: Sample structure of the Interactor Role Interaction ODP.

time many roles and participate in many different interactions (but never with different roles in the same interaction).

IMPLEMENTATION: The three object properties must be created first (HasRole, RoleInInteraction and HasParticipant). Each interaction bears a closure, so apart of an axiom [HasParticipant exactly 1 InteractorX], an axiom of the type [HasParticipant only (Interactor1 or InteractorN)] should be used. For each interactor, add an axiom of the type [HasRole some (Role and RoleInInteraction Interaction), so queries can be decomposed for roles or participation in interactions, or participations in interactions with certain roles.

RESULT: The participation events and the role with which interactors participate in concrete interactions are decoupled. Also, each interaction has a given set of interactors and not more.

ADDITIONAL INFORMATION: See also the following paper: M. Dumontier. Biological situational modeling: Defining Molecular Roles in Pathways and Reactions. 2008. OWL Experiences and Design (OWLED-EU 2008).

REFERENCES:

- `http://odps.sourceforge.net`
- `http://www.cellcycleontology.org/`

URL: Interactor_Role_Interaction.owl

A.10 List ODP

ALSO KNOWN AS: Linked list.

CLASSIFICATION: Domain Modelling.

MOTIVATION: An ordered group of elements is a very intuitive modelling structure, yet the semantics of such a construct in OWL DL are complex. Biology is full of structures where the order of the elements is vital (e.g. parts of genes). If that order is altered (e.g. a change of the order of introns and exons in a gene) there can be serious damage in biological systems.

AIM: The List is used to model ordered elements, representing the semantics of the order: in this case the ODP will be used to build a gene starting from some elements of the Sequence Ontology: Promoter (SO:0000167), Terminator (SO:0000141), Intron (SO:0000188) and Exon (SO:0000147). For the sake of clarity a minimalist gene is built, with a very simple structure.

STRUCTURE: See Figure A.19.

SAMPLE: See Figure A.20.

ELEMENTS: The most important elements are the different classes that can be used to build the List (Promoter, Terminator, Intron and Exon) and the class that it is modelled using the List (in this case Gene). The needed relationships are: Contents (functional), Rest (transitive) and Next (functional and a subproperty of Rest).

IMPLEMENTATION: There is a Protege wizard available for easily creating lists.

RESULT: The result is the class Gene, with the elements in the proper order. Apart of being an efficient way of modelling ordered elements, Lists offer the possibility of creating a powerful classyfing system: Lists of plenty of kinds can be defined (e.g. definitions of the following type: any List containing elements A and B, not followed by C and then followed by two D-s) and they will be put in the correct position of the hierarchy of already defined lists. Using that procedure, for example, different protein fingerprints (lists of regular expressions) or different kinds of genes can be defined. The models can be queried, for example, with a given gene defined with a certain ordered combination of introns, exons,

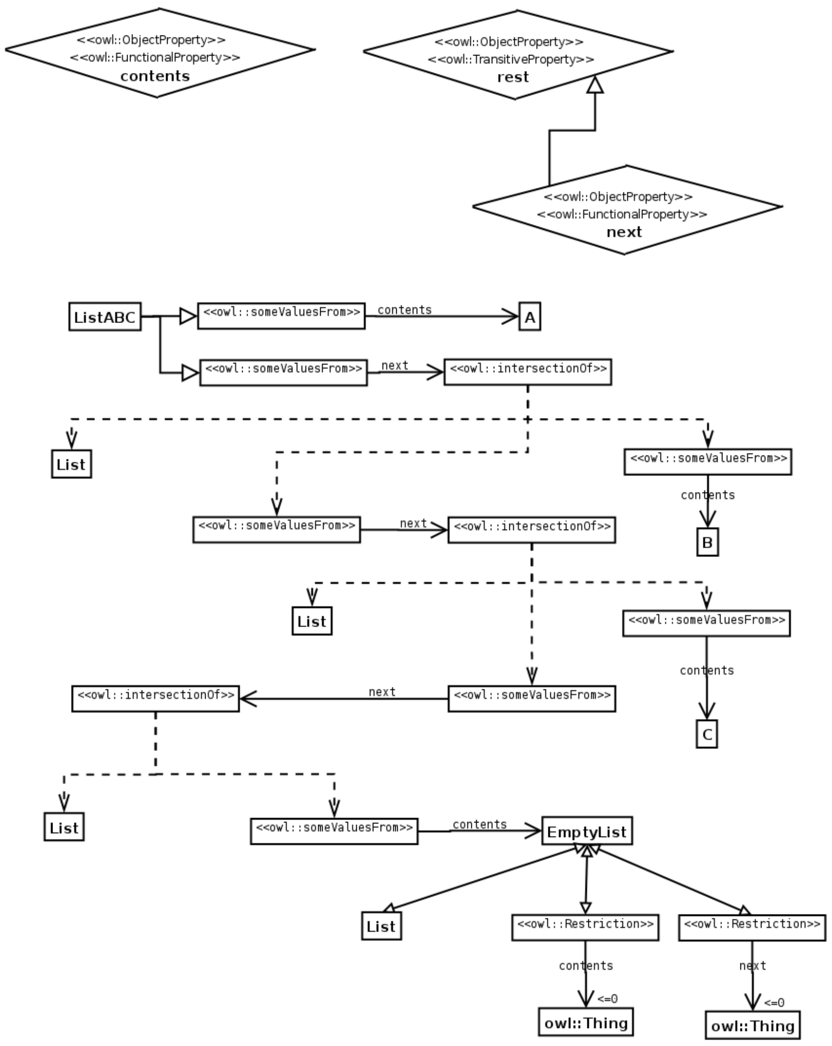


Figure A.19: Abstract structure of the List ODP.

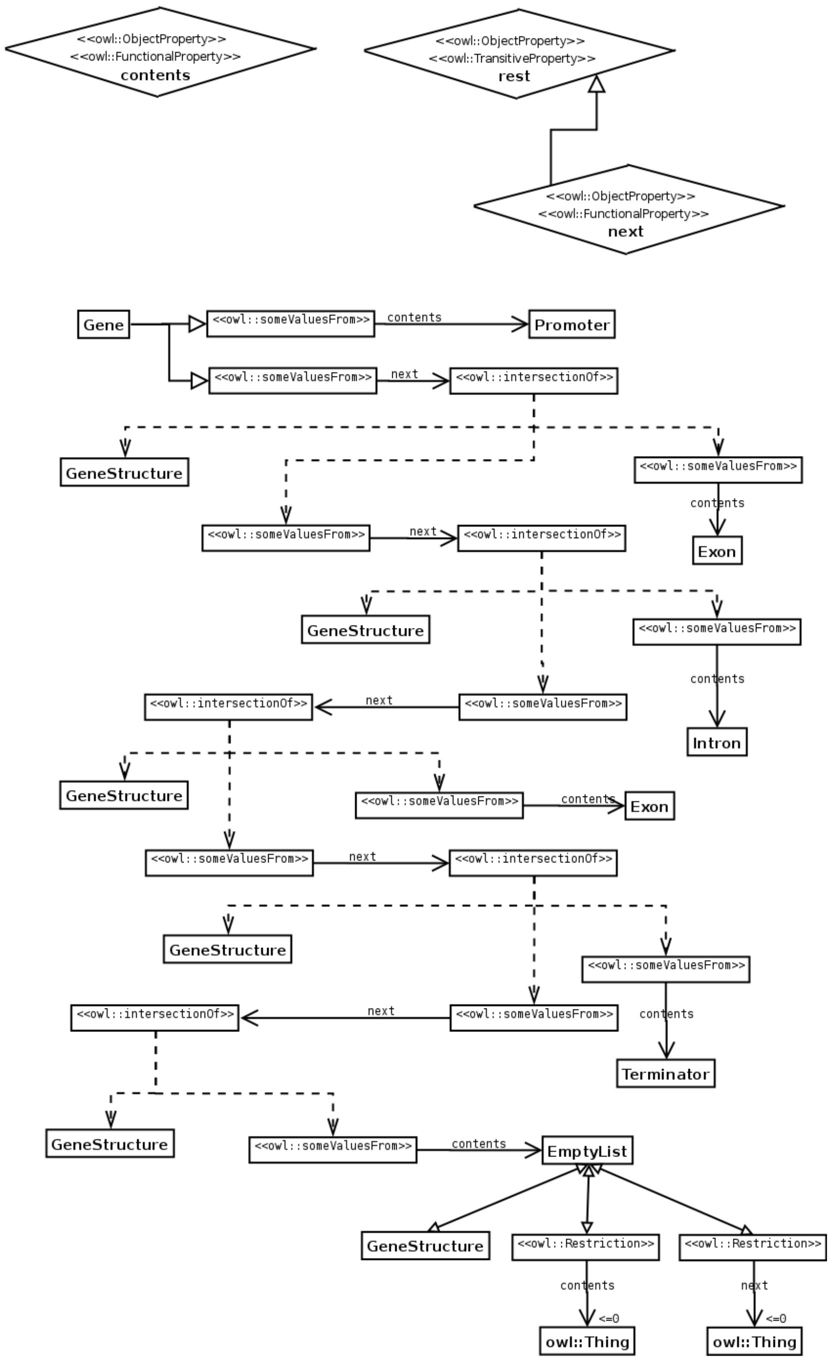


Figure A.20: Sample structure of the List ODP.

promoter and terminator to see in which position of the hierarchy is classified and to which genes does it relate.

SIDE EFFECTS: (i) If very long and complex lists are used there can be a decrease in reasoning performance. (ii) Maintenance of Lists is a very difficult task.

ADDITIONAL INFORMATION: The Linked List is one of the oldest and most widely used data structures in computer science; plenty of programming languages offer primitives similar to it. The Circularly Linked List is a List that ends up with the beggining of itself, creating a circle. The application of the Circularly Linked List in OWL DL has not been investigated yet. The wikipedia entry offers plenty of information on the subject: http://en.wikipedia.org/wiki/Linked_list.

REFERENCES:

- `http://www.co-ode.org/resources/tutorials/bio/`
- Nick Drummond, Alan Rector, Robert Stevens, Georgina Moulton, Matthew Horridge, Hai H. Wang, Julian Seidenberg. Putting OWL in Order: Patterns for Sequences in OWL. OWLed 2006.
- Robert Stevens, Mikel Egana Aranguren, Katy Wolnstencroft, Ulrike Sattler, Nick Drummond and Mathew Horridge. Using OWL to Model Biological Knowledge. International Journal of Human Computer Studies 2006, 65:7, 583-594.

URL: List.owl

A.11 Nary DataType Relationship ODP

CLASSIFICATION: Extension.

MOTIVATION: Numerical values can have different aspects. For example, a boiling point has a temperature value, a pressure, etc. This simple ODP should be used to model those cases.

AIM: To represent a datatype value with more than one aspect.

STRUCTURE: See Figure A.21.

SAMPLE: See Figure A.22.

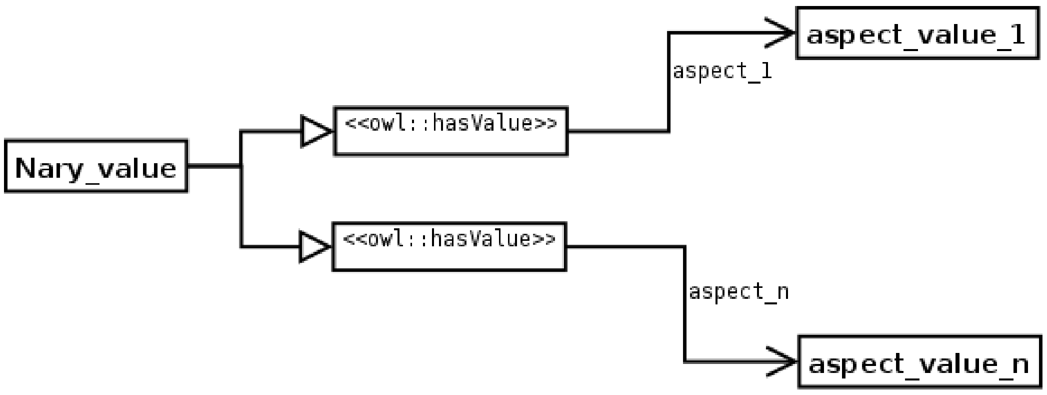


Figure A.21: Abstract structure of the Nary DataType Relationship ODP.

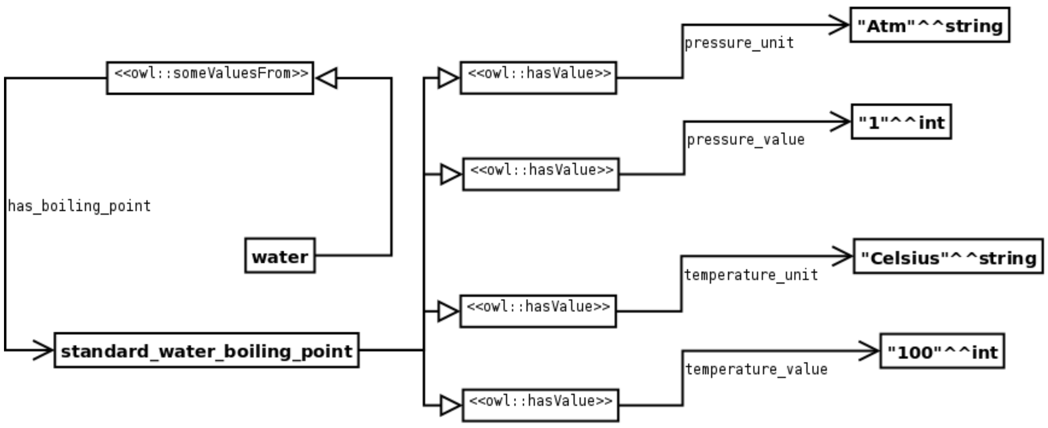


Figure A.22: Sample structure of the Nary DataType Relationship ODP.

ELEMENTS: The original value is reified (decomposed) in all the neccesary data type properties and values.

IMPLEMENTATION: The first step is to choose the datatype value that needs to be reified and create a class for it (e.g. StandardWaterBoilingPoint), then add a restriction (e.g. [Water HasBoilingPoint some StandardWaterBoilingPoint]) and all the neccesary datatype properties and restrictions to the reified class (e.g. [StandardWaterBoilingPoint partial HasUnit value celsius], [StandardWaterBoilingPoint partial HasValue value 100], etc.).

RESULT: After the reification a value with different aspects is represented in the ontology.

RELATED ODPS: Nary Relationship.

REFERENCES:

- `http://www.cs.man.ac.uk/~stevensr/menupages/ontologies.php`
- Bijan Parsia and Michael Smith. Quantities in OWL. OWLed 2008 EU.

URL: Nary_DataType_Relationship.owl

A.12 Nary Relationship ODP

CLASSIFICATION: Extension.

MOTIVATION: The biomedical domain is full of situations were relationships should hold between more than one element, but OWL only allows to express properties linking two individuals at a time. There can be a situation where a relationship and some properties of that relationship must be modelled; that can not be done in a direct manner with OWL. For example, a diagnosis has a result, a probability, and the person who has been diagnosed; A catalytic reaction has got a substrate, some products, catalytic constants and it is catalysed by an enzyme.

AIM: To express a relationship between more than one element. A Gene Ontology example can be found in the term GolgiToPlasmaMembrane CFTRproteinTransport, where there is a transport phenomenom which relates to three elements at the same time: the start (Golgi apparatus), the end (plasma membrane) and the

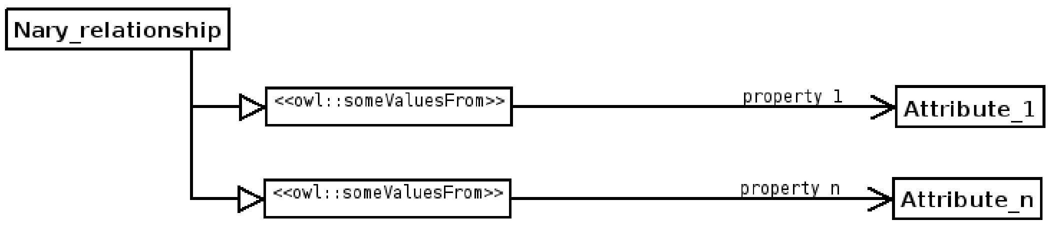


Figure A.23: Abstract structure of the Nary Relationship ODP.

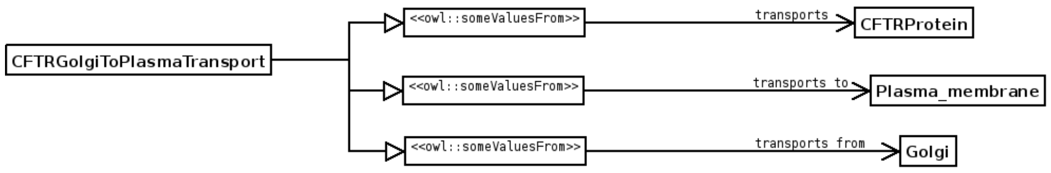


Figure A.24: Sample structure of the Nary Relationship ODP.

transportee (CFTR protein). The transport relation can not be modelled in OWL pointing to the three elements, so this ODP must be applied.

STRUCTURE: See Figure A.23.

SAMPLE: See Figure A.24.

ELEMENTS: The original elements of the Nary Relationship are conserved in classes and a new class is reified to model the Nary Relationship, in this case a class called CFTRGolgiToPlasmaTransport. The relationships of each element to the reified class are created: TransportsFrom, TransportsTo and Transports.

IMPLEMENTATION: There is a Protege wizard available for easily creating N-ary Relationships.

RESULT: After the reification a N-ary relationship is stated in the ontology.

SIDE EFFECTS: The class that holds the Nary relationship must have a clear name, because the maintainer must know that the class is not a proper class, otherwise the ontology becomes confusing. Therefore naming consistency should be maintained if different Nary relationships are created.

ADDITIONAL INFORMATION: It could be argued that this is not really an ODP, as n-ary relationships in OWL do not exist, and, therefore, this would be a naming ODP instead of a semantic ODP, as the key of the pattern is how the class holding the alleged n-ary relationship is named.

REFERENCES:

- Robert Stevens, Mikel Egana Aranguren, Katy Wolnstencroft, Ulrike Sattler, Nick Drummond and Mathew Horridge. Using OWL to Model Biological Knowledge. International Journal of Human Computer Studies 2006, 65:7, 583-594.
- `http://www.co-ode.org/resources/tutorials/bio/`
- `http://www.w3.org/TR/swbp-n-aryRelations/`

URL: Nary_Relationship.owl

A.13 Normalisation ODP

ALSO KNOWN AS: Untangling.

CLASSIFICATION: Good Practice.

MOTIVATION: There are ontologies where a given class can have plenty of superclasses, building a polyhierarchy. If all those subsumption relationships are directly stated by the ontology maintainer, two main problems rise: (i) the ontology becomes very difficult to maintain: whenever a subsumption must be deleted (because a class has changed) or created (because a new class has been created) it has to be done by hand; in a polyhierarchy the process becomes very inefficient and error-prone. (ii) the semantics are implicitly stated, not explicitly: any other ontologist or reasoner only knows that a class is a subclass of its superclasses, without knowing why.

AIM: To untangle a polyhierarchy, coding the subsumption relationships using restrictions rather than class-subclass relationships. The application example for this ODP is adapted from the Cell Type Ontology. In the example, the subsumption relationships that already are in the Cell Type Ontology are inferred by the reasoner instead of hard-coded. The term Neutrophil is used as an example class to show how a class can relate to different modules.

STRUCTURE: See Figure A.25.

SAMPLE: See Figure A.26.

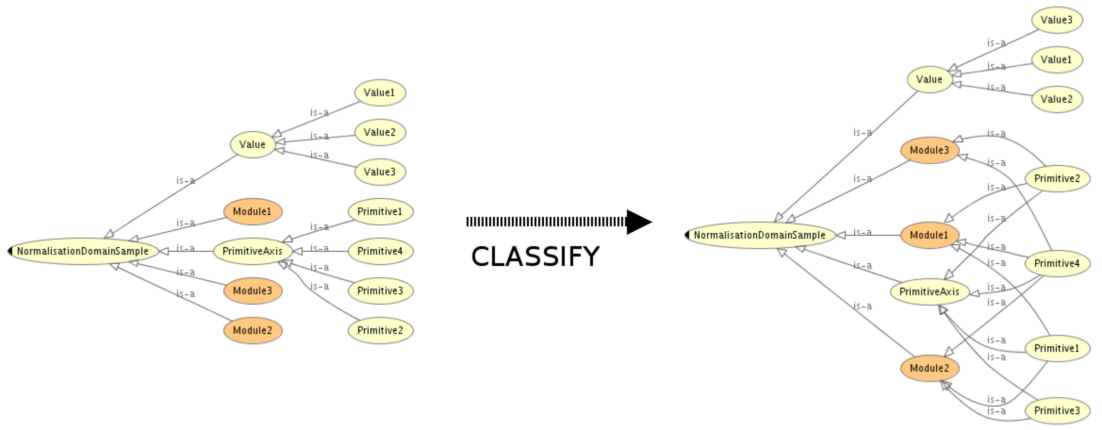


Figure A.25: Abstract structure of the Normalisation ODP.

ELEMENTS: The original classes of the ontology are divided in different axes. The conditions for each subsumption relationship are encoded as restrictions (e.g. [PerformsFunction some Defense]) that will relate the different modules.

IMPLEMENTATION: Identify the modules: group the classes. Create the modules, maintaining only one parent for any given primitive class and making primitive siblings disjoint. Redefine the classes (or define the newly added classes) according to the conditions for belonging to each module. Protege includes a wizard, the restrictions matrix, that helps in the process.

RESULT: The ontology gets untangled and becomes a collection of neat modules. The rest of the semantics are given by restrictions pointing to the modules, and the reasoner maintains the structure, avoding error-prone human maintenance of the polyhierarchy.

REFERENCES:

- Alan L. Rector. Modularisation of Domain Ontologies Implemented in Description Logics and related formalisms including OWL. K-CAP 2003.
- Alan L. Rector, Chris Wroe, Jeremy Rogers and Angus Roberts. Untangling Taxonomies and Relationships: personal and Practical Problems in Loosely Coupled Development of Large Ontologies. K-CAP 2001.
- `http://www.co-ode.org/resources/tutorials/bio/`

URL: Normalisation.owl

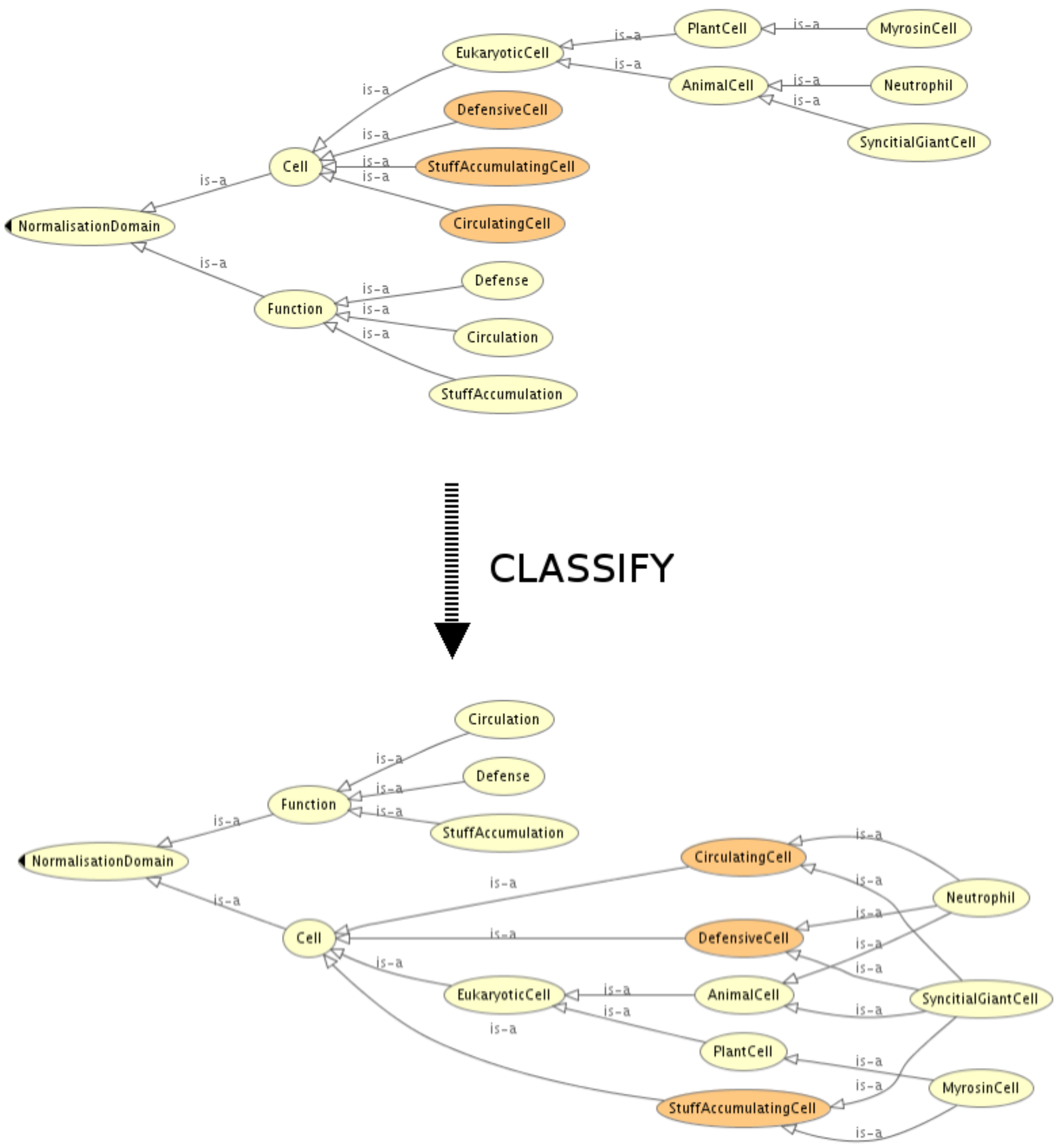


Figure A.26: Sample structure of the Normalisation ODP.

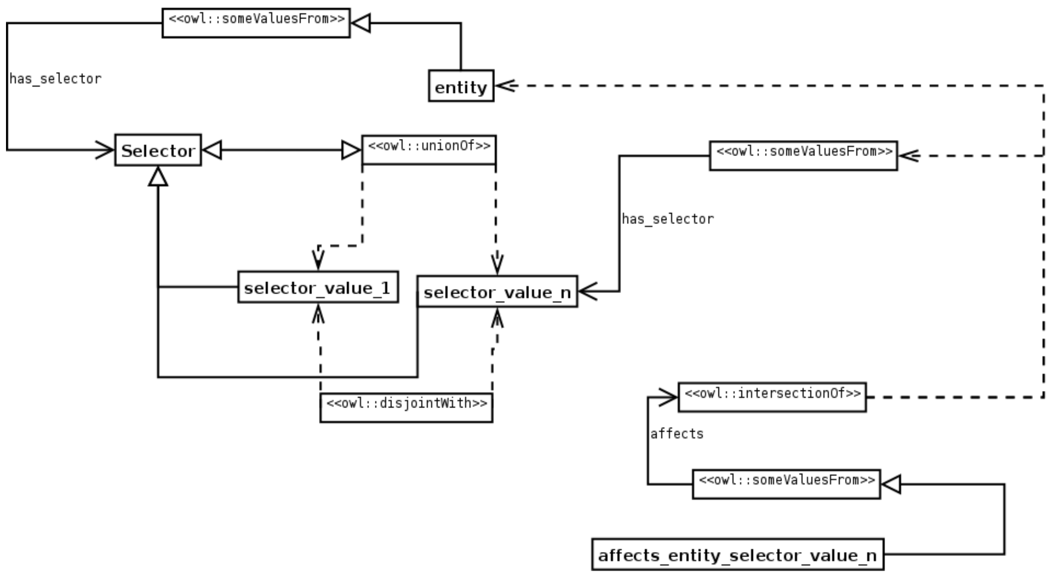


Figure A.27: Abstract structure of the Selector ODP.

A.14 Selector ODP

CLASSIFICATION: Good Practice.

MOTIVATION: Selectors are widely used in the biomedical domain, especially in the realm of anatomy. A selector is a modifier that can be used to select between identical entities, e.g. right and left hand. Selectors are usually associated with simmetry (left-right,anterior-posterior,lateral-medial) and sometimes hard coded in ontologies, that is, for example left hand and right hand are introduced as subclasses of hand, which adds an unecessary amount of classes. This ODP avoids such proliferation of classes.

AIM: To recreate selectors, that is refining entities that can be used to choose between to alternatives: for example, right or left hand.

STRUCTURE: See Figure A.27.

SAMPLE: See Figure A.28.

ELEMENTS: The main element is the selector class, be it Laterality (covered by Left and Right), AnteriorPosteriorSelector (covered by Anterior and Posterior), etc. A functional object property, e.g. HasLaterality, is used to add a selector to the classes of the domain hierarchy (e.g. hand can be left or right).

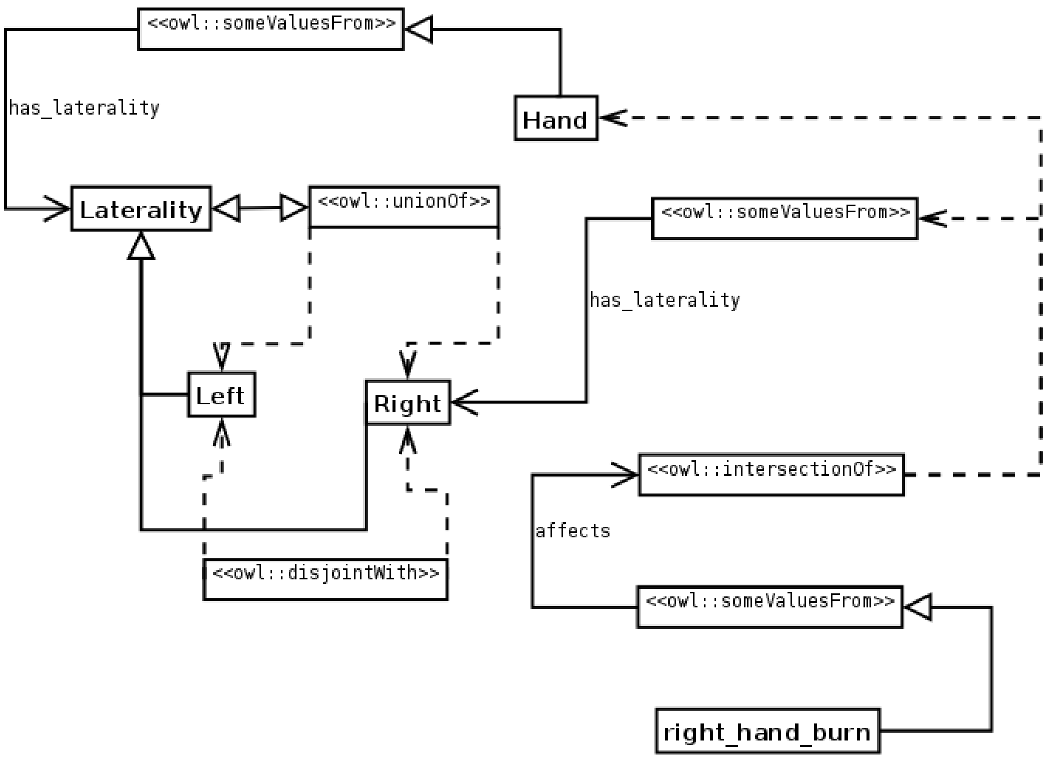


Figure A.28: Sample structure of the Selector ODP.

IMPLEMENTATION: Usually this ODP is implemented in already existing ontologies where selectors are implicit. For example, in the FMA ontology, the class Hand has the subclasses LeftHand and RightHand. Using this ODP, the classes LeftHand and RightHand can be deleted, and the class Hand is linked to the class Laterality via an existential restriction on the HasLaterality property.

RESULT: The original ontology is considerably reduced in size. If we want to refer the original entities, we can do it by reusing the HasSelector property. For example, if we want to define a burn on the right hand, we can use the following expression: [Burn and Affects some (Hand and HasLaterality some Right)].

SIDE EFFECTS: Depending on the selection procedure, information could be lost when deleting the subclasses (e.g. RightHand), as they can have further subclasses or interesting axioms.

ADDITIONAL INFORMATION: See also Entity-Feature-Value, Entity-Property-Value and Entity-Quality.

REFERENCES:

- Alan Rector (Personal Communication).
- Eleni Mikroyannidi. Abstracting and generalising a large anatomy ontology. MSc Dissertation, Computer Science, Uni. of Manchester. 2008.

URL: Selector.owl

A.15 Sequence ODP

CLASSIFICATION: Domain Modelling.

MOTIVATION: In biological knowledge there are events that happen one after the other in a single related sequence, such as the cell cycle. Sometimes the only important thing is what happens after or before a concrete event, without the concrete order of all the events (in that case we would need the List ODP, for example to compare different sequences of events).

AIM: To model a sequence of events, one after the other.

STRUCTURE: See Figure A.29.

SAMPLE: See Figure A.30.

ELEMENTS: The elements of this ODP are the classes that make up the sequence (in this case the phases of the cell cycle, thus G1, S, G2, M) and the four properties Precedes (transitive), ImmediatelyPrecedes (subproperty of Precedes, functional) PrecededBy (transitive) and ImmediatelyPrecededBy (subproperty of PrecededBy, functional).

IMPLEMENTATION: The sequence is created by adding restrictions in the properties ImmediatelyPrecededBy and ImmediatelyPrecedes for each phase, except in the last one (only ImmediatelyPrecededBy) and the first one (only ImediatelyPrecedes).

RESULT: The sequence of events is codified creating an structure that can be queried with queries such as [OccursAt some (PrecededBy some S)], if we want anything that happens after S, or [OccursAt some (ImmediatelyPrecededBy some S)], if we want to know what happens right after S but not later (thus not G2 or M).

RELATED ODPS: List ODP, AdaptedSEP ODP.

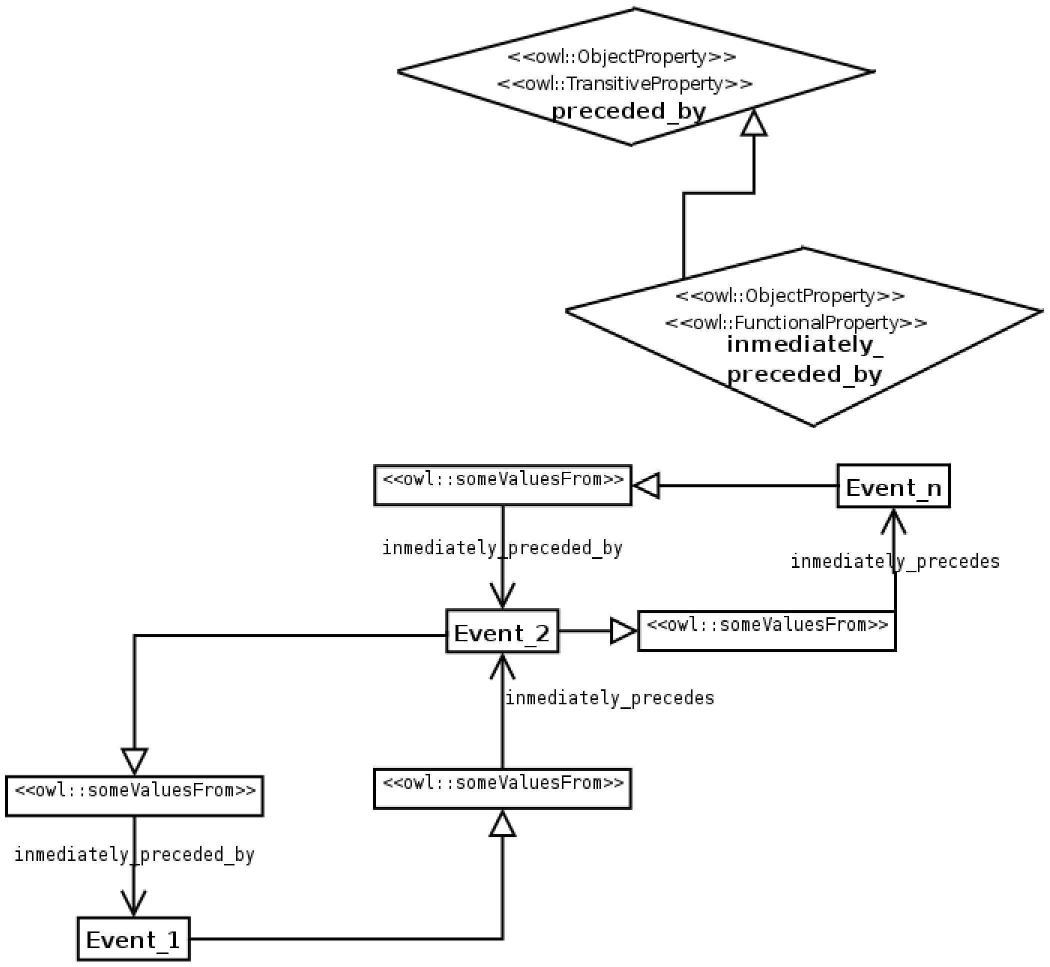


Figure A.29: Abstract structure of the Sequence ODP.

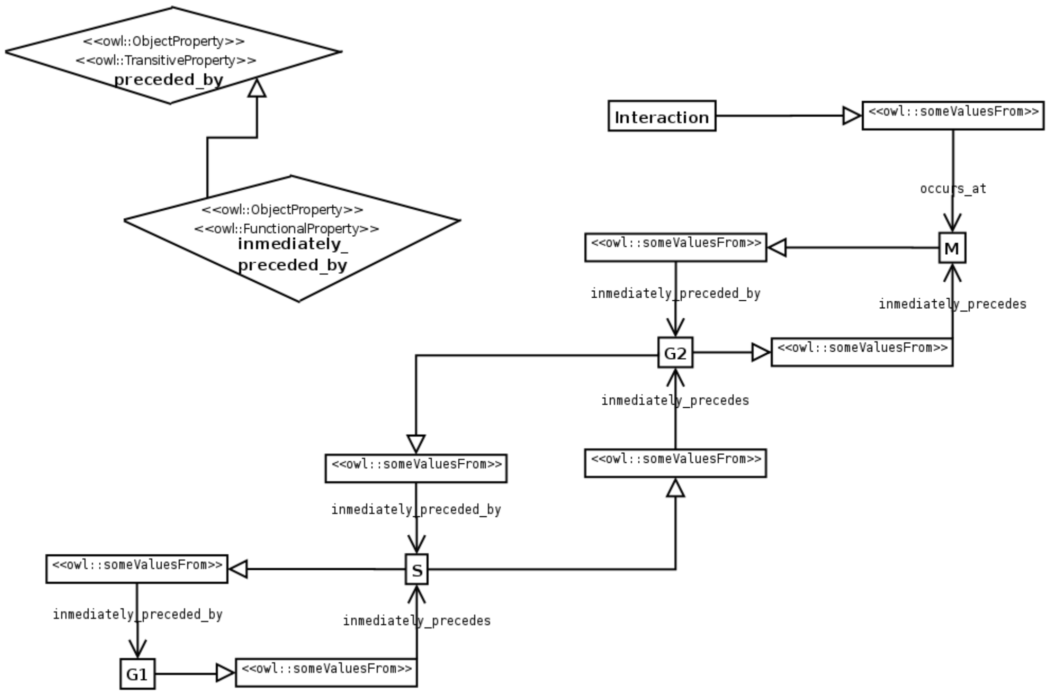


Figure A.30: Sample structure of the Sequence ODP.

ADDITIONAL INFORMATION: in theory only ImmediatelyPrecedes relationships should be asserted and the reasoner should infer the inverse relationship, but it does not work for the superproperty of the inverse. Also, for the inverse to work properly some kind of closure (onlysome) or defined classes are needed.

REFERENCES:

- `http://www.cellcycleontology.org`

URL: Sequence.owl

A.16 Upper Level Ontology ODP

ALSO KNOWN AS: Foundational ontology.

CLASSIFICATION: Good Practice.

MOTIVATION: Different ontologies of a given domain share very general types of concepts, like Substance, Modifier, etc. These types of concepts are grounded in philosophical criteria, like distinctions between Occurents and Continuants. The

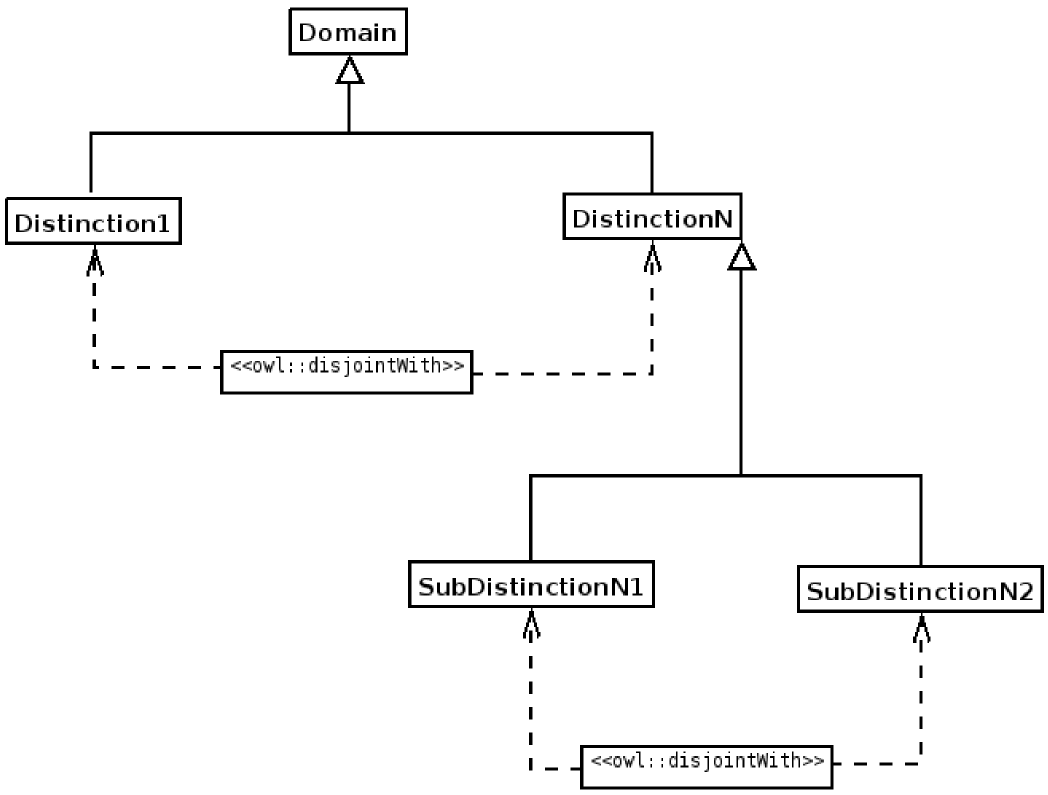


Figure A.31: Abstract structure of the Upper Level Ontology ODP.

different domain ontologies can thus be integrated in one Upper Level Ontology, each ontology having different relationships pointing to the concepts of the Upper Level Ontology. The Upper Level Ontology used here as an example is the Ontology of Biomedical Reality (OBR).

AIM: To create an ontology that can integrate different ontologies in itself.

STRUCTURE: See Figure A.31.

SAMPLE: See Figure A.32.

ELEMENTS: All the classes that represent a conceptual category.

IMPLEMENTATION: The different hierarchies of primitive classes must be asserted using disjoints.

RESULT: By endorsing to a given Upper Level Ontology when building a domain ontology the ontologists makes the integration of the ontology with other ontologies a much easier process. Besides, the ontology becomes a cleaner model

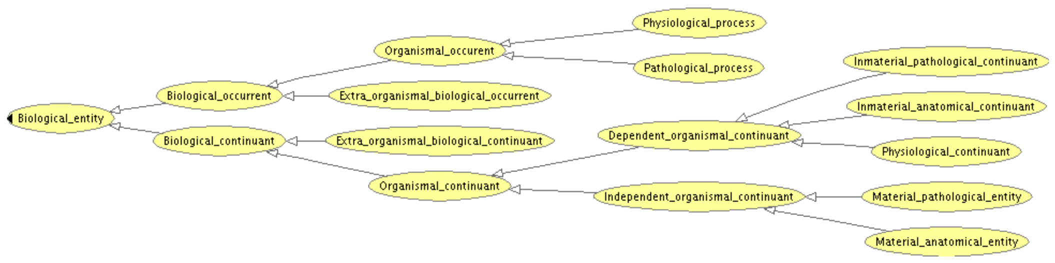

Biological_entity
Biological_occurrent
Biological_continuant
Organismal_occurent
Extra_organismal_biological_occurrent
Extra_organismal_biological_continuant
Organismal_continuant
Physiological_process
Pathological_process
Dependent_organismal_continuant
Independent_organismal_continuant
Inmaterial_pathological_continuant
Inmaterial_anatomical_continuant
Physiological_continuant
Material_pathological_entity
Material_anatomical_entity

Figure A.32: Sample structure of the Upper Level Ontology ODP.

with different modules.

SIDE EFFECTS: The ontology is committed to a concrete view of the knowledge domain (given by the Upper Level Ontology), and therefore the use and implantation of Upper Level Ontologies is very controversial.

REFERENCES:

- Barry Smith et al. A Strategy for Improving and Integrating Biomedical Ontologies. AMIA 2005.
- `http://www.co-ode.org/resources/tutorials/bio/`

URL: Upper_Level_Ontology.owl

A.17 Value Partition ODP

ALSO KNOWN AS: Enumeration, if it is built using individuals instead of classes.

CLASSIFICATION: Good Practice.

MOTIVATION: Reality is full of attributes of elements. For example, a person can be defined as being short, medium or tall, and the attribute height can just get those values. Height is said to be covered or exhausted by those values; the possible heights are only those three. Biology is full of such situations: metabolism can only be anabolism or catabolism, membrane transport can only be uniport, sinport or antiport, regulation is always positive, negative, and so forth.

AIM: To model values of attributes. In this example we model biological regulation, being negative or positive. PositiveRegulationOfCellKilling, from GO, is linked to the appropriate value.

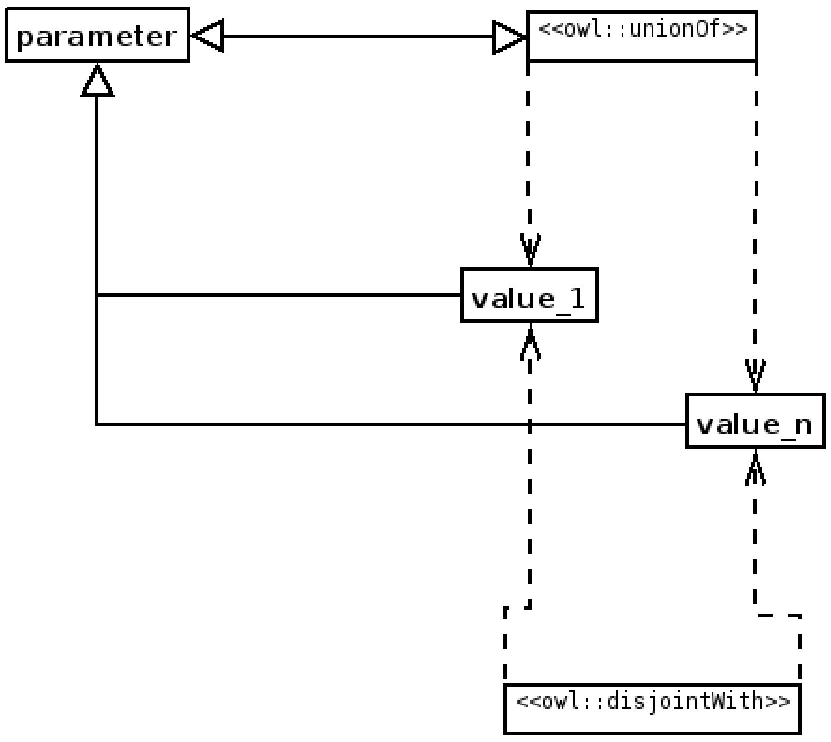


Figure A.33: Abstract structure of the Value Partition ODP.

STRUCTURE: See Figure A.33.

SAMPLE: See Figure A.34.

ELEMENTS: The main elements are the classes that make up the Value Partition itself: a class for the attribute and the subclasses for the values. In this case, Regulation, Positive, and Negative, respectively. The most important relationship is the one that links each element of the knowledge domain with the values of the Value Partition. In this case, IsRegulationOfType (functional).

IMPLEMENTATION: Identify the attributes every element must be described with. For each attribute, create a class under Modifier (or the pertinent upper level distinction that it is used in the ontology). In each attribute class create a subclass for every value and make them disjoint. Create a covering axiom defining the attribute class. Create the restrictions pointing to the values of the Value Partition.

RESULT: The attributes and the elements that are described or modified by the attributes get untangled: whenever a new element enters the domain (e.g. another

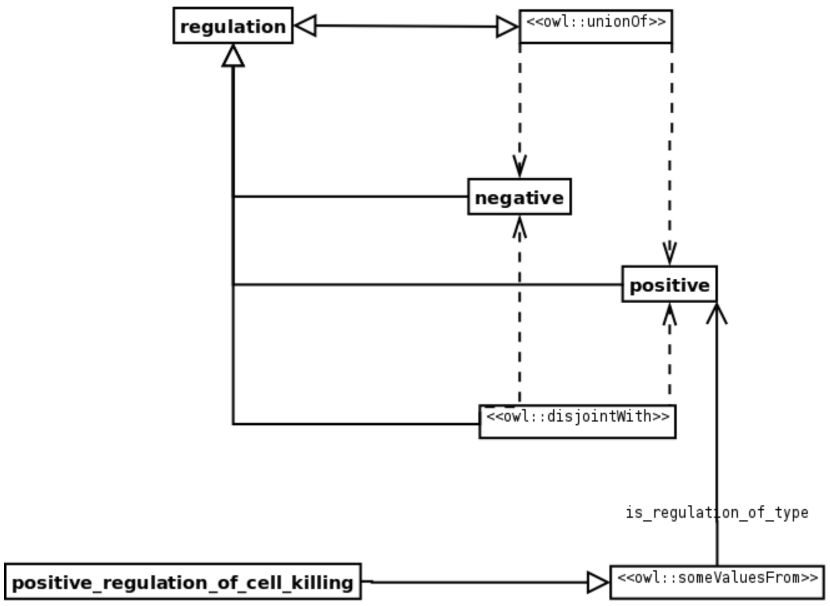


Figure A.34: Sample structure of the Value Partition ODP.

regulation phenomenon) it is only a matter of adding a restriction pointing to the pertinent Value Partition class. The values that can be given to a certain attribute are constrained, enforcing a better modelling.

ADDITIONAL INFORMATION: The Value Partition built with classes offers an advantage over the Enumeration (a Value Partition built with individuals): new subpartitions can be built for each of the value classes (e.g. very tall).

REFERENCES:

- `http://www.co-ode.org/resources/tutorials/bio/`
- `http://www.w3.org/TR/swbp-specified-values`

URL: Value_Partition.owl

Appendix B

Ontology quality values for CL and nCL

	CL	nCL
Formalisation	5	5
Relations	1	5
Cohesion	1	5
Tangledness	1	5
Redundancy	5	5
Structural accuracy	5	5
Domain coverage	5	3

Table B.1: Quality comparison of CL and nCL: structural.

	CL	nCL
Reference Ontology	3	5
Controlled Vocabulary	5	5
Schema and Value Reconciliation	1	5
Consistent Search and Query	1	5
Knowledge Acquisition	1	5
Clustering and Similarity	3	5
Indexing and Linking	5	3
Results Representation	1	1
Classifying Instances	1	5
Text Analysis	3	5
Guidance and Decision Trees	1	5
Knowledge Reuse	1	5
Inferencing	1	5
Interoperability	3	5

Table B.2: Quality comparison of CL and nCL: functionality.

	CL	nCL
Technological maturity	5	5
Knowledge maturity	5	3
Robustness	1	5
Authority	5	3

Table B.3: Quality comparison of CL and nCL: reliability.

	CL	nCL
Readability	1	1
Reusability	3	5

Table B.4: Quality comparison of CL and nCL: usability.

	CL	nCL
Stability	1	5
Analysability	1	5
Changeability	1	5
Testability	5	5

Table B.5: Quality comparison of CL and nCL: Maintainability.

	CL	nCL
Effectiveness	5	3
Popularity	5	1
Engagement	3	1

Table B.6: Quality comparison of CL and nCL: Quality in use.

	CL	nCL
Structure	3.28	4.71
Functionality	2.14	4.57
Reliability	4	4
Usability	2.5	3
Efficiency	3	1
Maintainability	2	5
Quality in use	4.33	1.66

Table B.7: Overall comparison of CL and nCL.