UNIVERSITY OF SOUTHAMPTON

Towards Ontology Design Patterns to Model Multiple Classification Criteria of Domain Concepts in the Semantic Web

by

Benedicto
Rodriguez Castro

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the
Faculty of Engineering and Applied Science
Department of Electronics and Computer Science

July 2012

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND APPLIED SCIENCE
DEPARTMENT OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Benedicto
Rodriguez Castro

This thesis explores a very recurrent modeling scenario in ontology design that deals with the notion of real world concepts that can be classified according to multiple criteria. Current ontology modeling guidelines do not explicitly consider this aspect in the representation of such concepts. Such void leaves ample room for ad-hoc practices that can lead to unexpected or undesired results in ontology artifacts. The aim is to identify best practices and design patterns to represent such concepts in OWL DL ontologies suitable for deployment in the Web of Data and the Semantic Web.

To assist with these issues, an initial set of basic design guidelines is put forward, that mitigates the opportunity for ad-hoc modeling decisions in the development of ontologies for the problem scenario described. These guidelines relies upon an existing simplified methodology for *facet analysis* from the field of Library and Information Science. The outcome of this facet analysis produces a Faceted Classification Scheme (FCS) for the concept in question where in most cases a *facet* would correspond to a *classification criterion.*

The Value Partition, the Class As Property Value and the Normalisation Ontology Design Patterns (ODPs) are revisited to produce an ontology representation of a FCS. A comparative analysis between a FCS and the Normalisation ODP in particular, revealed the existence of key similarities between the elements in the generic structure of both knowledge representation paradigms. These similarities allow to establish a series of mappings to transform a FCS into an OWL DL ontology that contains a valid representation of the classification criteria involved in the characterization of the domain concept. An existing FCS example in the domain of "Dishwasher Detergent" and existing ontology examples in the domain of "Pizza", "Wine" and "Fault" (in the context of a computer system) are used to illustrate the outcome of this research.

Contents

List of Figures

List of Tables

Listings

Declaration of Authorship

I, *Benedicto Rodriguez Castro*, declare that the thesis entitled *"Towards Ontology Design Patterns to Model Multiple Classification Criteria of Domain Concepts in the Semantic Web"* and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- parts of this work have been peer-reviewed and published as: (see section "Publications" on the next page.)

Signed:

Date: July 2012

Publications

An overview of the content of these publications and how they relate and map into the contributions of this thesis is provided on Section 1.5.

Bene Rodriguez-Castro, Hugh Glaser, and Leslie Carr. How to Reuse a Faceted Classification and Put It on the Semantic Web. In Peter Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Pan, Ian Horrocks, and Birte Glimm, editors, *The Semantic Web ISWC 2010*, volume 6496 of *Lecture Notes in Computer Science*, pages 663–678. Springer Berlin / Heidelberg, 2010b. ISBN 978-3-642-17745-3 - (*"Best Student Research Paper"* Nominee[1])

Bene Rodriguez-Castro, Hugh Glaser, and Les Carr. Faceted Classification Scheme ODP. In *The 2nd Workshop on Ontology Patterns (WOP2010) at the 9th International Semantic Web Conference (ISWC)*, November 2010a - (*"Best Pattern Award"* Winner[2])

Bene Rodriguez-Castro, Hugh Glaser, and Ian Millard. View Inheritance as an Extension of the Normalization Ontology Design Pattern. In *Workshop on Ontology Patterns in the 8th International Semantic Web Conference (ISWC 2009)*, September 2009

Bene Rodriguez-Castro and Hugh Glaser. Whose "Fault" Is This? Untangling Domain Concepts in Ontology Design Patterns. In *Workshop on Knowledge Reuse and Reengineering over the Semantic Web in the 5th European Semantic Web Conference*, June 2008c

Bene Rodriguez-Castro and Hugh Glaser. Untangling Domain Concepts in Ontology Design Patterns. In *Poster at the 5th European Semantic Web Conference*, June 2008a

Bene Rodriguez-Castro and Hugh Glaser. Whose "Fault" Is This? Untangling Domain Concepts in an Ontology of Resilient Computing . In *Fast Abstracts at the 7th European Dependable Computing Conference (EDCC-7)*, April 2008b

Hugh Glaser, Ian Millard, Bene Rodriguez-Castro, and Afraz Jaffri. Demonstration: Knowledge-enabled research infrastructure. In *4th European Semantic Web Conference*, 2007b

Ian Millard, Afraz Jaffri, Hugh Glaser, and Bene Rodriguez-Castro. Using a semantic mediawiki to interact with a knowledge based infrustructure. In *15th International Conference on Knowledge Engineering and Knowledge Management*, 2006

[1] http://iswc2010.semanticweb.org/awards
[2] http://ontologydesignpatterns.org/wiki/WOP:2010#Accepted_patterns

Acknowledgements

Many people have contributed directly and indirectly to this work and I thank them all.

My gratitude to all of them is endless and it has many recipients that are mentioned below.

Those that one way or another participated in the day by day progress and contributed to it: Hugh Glaser, Les Carr, John Darlington, Martin Hepp, Madalina Croitoru, Harith Alani, Sarvapali "Gopal" Ramchurn, Afraz Jaffri, Manuel Sicilia Morales, Antonio Jesus Rueda Ruiz, Manuel Salvadores, Ian Millard, Nick Gibbins and Martin Szomszor.

Everyone that participated in the online discussion "The notion of a 'classification criterion' as a class" via the mailing lists: "ontolog-forum@ontolog.cim3.net"[3] and "public-owl-dev@w3c.org"[4], in particular: Pat Hayes, John Sowa and Doug Foxvog.

The British Chapter of the International Society for Knowledge Organization (ISKO) and specially: Aida Slavic and Vanda Broughton.

All the anonymous reviewers for their time and feedback, because every single review had a positive impact on the present document.

To the projects I worked for and to those that I started but I could not finish, because all of them proved useful and constructive in their own right.

I don't want to forget all the mistakes I made along the way, for the priceless lessons they provided. More often than not, more valuable than those gained from successes.

And lastly, on the non-technical side, when nothing else would work, the simple things that helped me get over the next hurdle: squash, running, swimming, football (in all its dimensions), jazz music, (Bill Evans and Dave Brubeck), two unofficial partners of the University of Southampton (the Crown Inn and the Cowherds pubs) and of course, my family and all my friends.

To all of you: Thank You!

[3] http://ontolog.cim3.net/forum/ontolog-forum/2010-04/msg00051.html
[4] http://lists.w3.org/Archives/Public/public-owl-dev/2010AprJun/0009.html

To my mother, Dolores Castro Soto.

Nomenclature

BC2	Bliss Bibliographic Classification (2nd Edition)
CODeP	Conceptual (or Content) Ontology Design Pattern
CPV	Class as Property Value
CRG	Classification Research Group
D&S	Dependability And Security
DDC	Dewey Decimal Classification
FA	Facet Analysis
FC	Facet Classification
FCS	Faceted Classification Scheme
ISKO	International Society for Knowledge Organization
KB	Knowledge Base
KR	Knowledge Representation
LCC	Library of Congress Classification
LIS	Library and Information Science
MI	Multiple Inheritance
NeOn	Networked Ontologies (Project)
NOR	Non-ontological Resource
ODP	Ontology Design Pattern
OS	Operating System
OWL	Web Ontology Language
ReSIST	Resilience for Survivability in Information Society Technologies
RDF	Resource Description Framework
RDF-S	Resource Description Framework Schema
RKB	ReSIST Knowledge Base
RSM	Resource Space Model
SWBPD-WG	Semantic Web Best Practices and Deployment Working Group
UDC	Universal Decimal Classification
UI	User Interface
VP	Value Partition
W3C	World Wide Web Consortium

Chapter 1

Introduction

The main topic of this thesis deals with the problem of modelling *multiple classification criteria* of domain concepts to be represented in ontologies suitable for deployment in the Semantic Web. Part of what this introductory chapter will clarify is what this problem *means* in the context of Ontology Engineering and the Semantic Web. The notion of multiple classification criteria is presented using examples of domain concept representations found in disciplines such as Library Information Systems, Object-Oriented Design and more importantly, in Ontology Engineering. The natural connection between representing multiple classification criteria and the presence of *multiple inheritance* in the resulting models, is also examined. The concept of Ontology Design Patterns within Ontology Engineering is another key element acknowledged in this introductory review, because it provides an ideal modelling template for capturing the potential solutions that may result as part of this endeavour. To navigate readers through this chapter, Section 1.1 starts with a recollection of the rationale that led to, and motivated the focus of this work. Section 1.2 is then responsible of providing a detailed characterization of the problem in question, while Section 1.3 outlines the main research areas that will be considered in addressing such problem, that is, predominantly Ontology Engineering, in addition to Object-Oriented Design and Library Information Systems. The rest of sections, Sections 1.4, 1.5 and 1.6, conclude the chapter, putting forward the research questions, contributions and structure of this thesis respectively.

1.1 Motivation

The general problem of modelling multiple classification criteria of domain concepts in ontologies for deployment in the Semantic Web, was identified as part of the work carried out for a particular project. This project required the creation of an ontology model for a specific domain concept, whose characteristics, after a process of examination, provided the starting point that originated the main focus of this research. In essence, this focus can be seen as the result of extrapolating this ontology modelling challenge to any domain concept that shares similar characteristics to those of the concept that this section describes. The project in question and the aforementioned modelling use case, are presented throughout the rest of this section.

1.1.1 ReSIST: An EU Network of Excellence

ReSIST stands for Resilience for Survivability in Information Society Technologies (IST) and it is a Network of Excellence (NoE) project funded under the Sixth Framework Programme of the European Union (EU) ReSIST (2005–2008). One of the objectives of the ReSIST project is to create a Knowledge Base (KB) application in the domain of resilient computing, partly inspired by the features demonstrated by the semantic web application CS AKTive Space, and sharing many of the same requirements [Glaser et al. (2004); Shadbolt et al. (2004)].

The ReSIST Knowledge Base (RKB) provides an ontologically mediated web portal (the RKB Explorer[1]), that enables the end-user to browse and search different type of information in the area of resilient systems: projects, people, institutions, publications, communities of practice, courses, etc. [Anderson et al. (2007)]. The RKB Explorer employs various semantic web technologies to meet its goals such as RDF data repositories, RDF stores and ontologies. Further information of the main components, technologies and challenges found in the development of the RKB application can be found in Glaser et al. (2009, 2007b,a); Millard et al. (2006).

The RKB features several ontologies to assist with the interoperability of the various data sources that conform the knowledge of the application. Suitable ontologies have been reused to represent common concepts such as people, projects or publications, however, a key ontology had to be developed from scratch. This is an ontology on Dependability and Security (D&S), required to facilitate the exploitation of all knowledge related to concepts of computer resilience hosted by the ReSIST Project. It is certain aspects of the creation of the D&S ontology that will narrow the focus of this research.

1.1.2 The "Fault" Domain Concept in ReSIST

The D&S ontology is derived from the definitions and taxonomies presented in the paper "Basic Concepts and Taxonomy of Dependable and Secure Computing" Avizienis et al. (2004). The domain experts in the ReSIST Network agreed on the use of the cited reference as a valid source for the terminology and concepts to be represented in the D&S ontology. This approach allowed to *bootstrap* the knowledge acquisition phase required during the ontology development process. At the core of the D&S ontology

[1] http://www.rkbexplorer.com/

lies the domain concept of "Fault". The definition of "Fault" to be represented in the ontology is given by Avizienis et al. (2004) as follows:

> [In a computer system], a service is a sequence of the system's external states, a service failure means that at least one (or more) external state of the system deviates from the correct service state. The deviation is called an *error*. The adjudged or hypothesized cause of an error is called a *fault*.

The paper complements the definition of "Fault" presenting a taxonomy of faults. The taxonomy classifies all faults that may affect a system during its life according to *8 basic viewpoints* that lead to *16 elementary fault classes*. The tree hierarchy in Figure 1.1 shows (a) the 8 basic viewpoints of faults in the first level of the tree; and (b) the 16 elementary fault classes in the second level of the tree. Note that each fault viewpoint is covered by two mutually exclusive elementary fault classes.

If all combinations of these 16 elementary fault classes from the 8 basic viewpoints were possible, the total number of combined fault classes would be 256. However, not all combinations occur in reality and Avizienis et al. (2004) have identified *31 likely combinations* of the elementary fault classes (although, the authors also acknowledge that more combinations may be possible in the future). Figure 1.2 and 1.3 illustrate the composition of these 31 likely combination of faults (fault numbers 1 to 31) as a matrix and a tree representation respectively.

Figure 1.2 presents the following information:

- At the top of the matrix, three major partially overlapping groupings to which the 31 combined faults belong to, namely: Development Faults, Physical Faults and Interaction Faults.
- On the left side of the matrix, the 16 elementary fault classes from the 8 basic viewpoints of the taxonomy of "Fault" introduced in Figure 1.1.
- At the bottom of the matrix, the 31 likely combined faults numbered from 1 to 31.
- At the bottom of the figure, a series of boxes identifying 9 illustrative examples of known faults, namely: Software Faults, Logic Bombs, Hardware Errata, Production Defects, Physical Deterioration, Physical Interference, Intrusion Attempts, Viruses & Worms and Input Mistakes.

On the other hand, Figure 1.3 presents for the most part the same information as a tree view, specifically:

- On the left hand side of the tree, the 8 basic viewpoints of "Fault" shown in Figure 1.1.

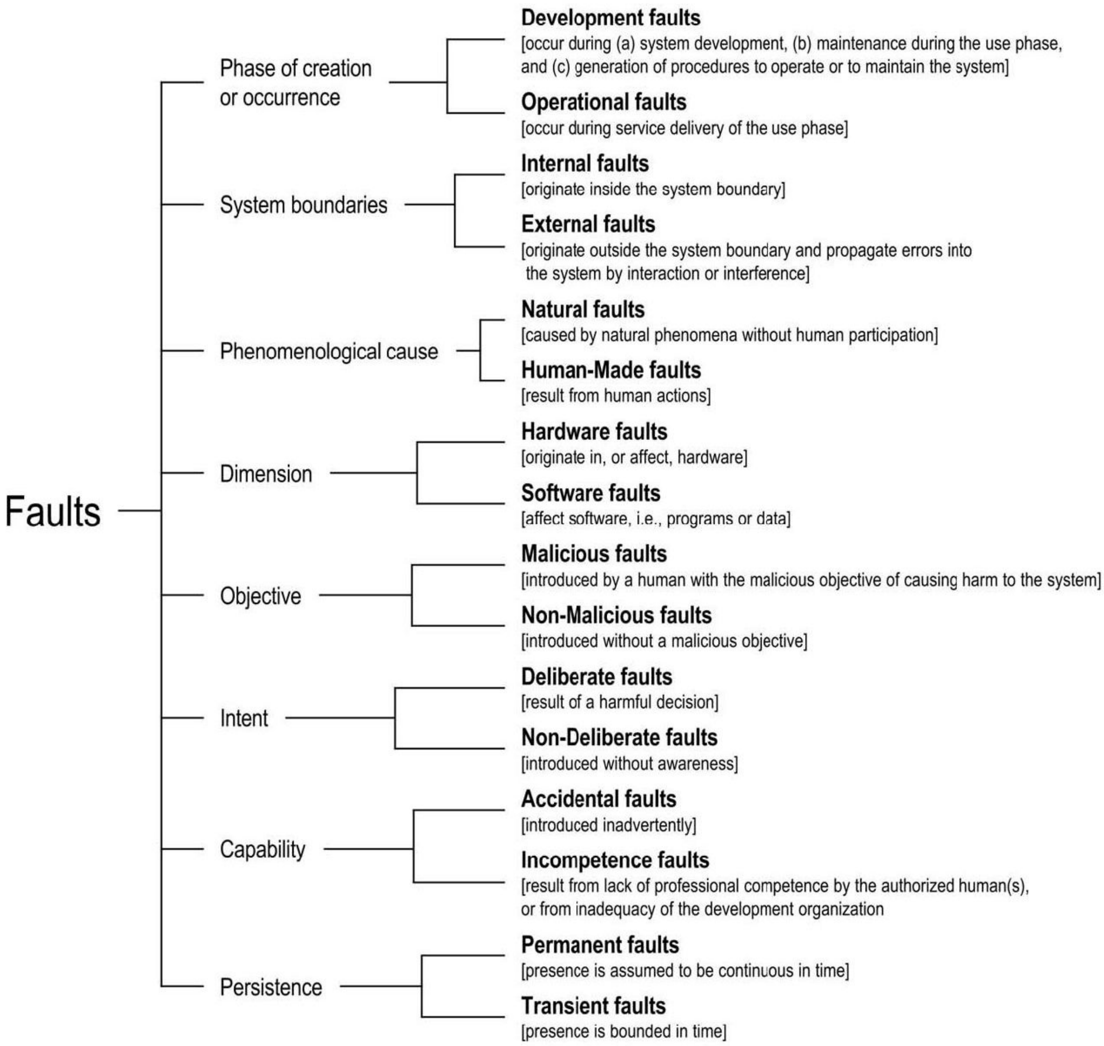


Figure 1.1: *"The elementary fault classes."* (Fig. 4 in Avizienis et al. (2004) p. 15)

- Every node of the tree denotes one of the 16 elementary fault classes shown in Figure 1.1.
- At the bottom of the tree, the leaf nodes, correspond to the 31 likely combined faults numbered from 1 to 31.
- At the bottom of the figure, the three major partially overlapping groupings to which the 31 combined faults belong to, again: Development Faults, Physical Faults and Interaction Faults.

Additional details and background information regarding each one of the concepts presented by these three figures is available in the original paper by Avizienis et al. (2004), but in essence, the three figures capture the definition and taxonomy of "Fault" to include in the D&S ontology as part of the ReSIST Knowledge Base.

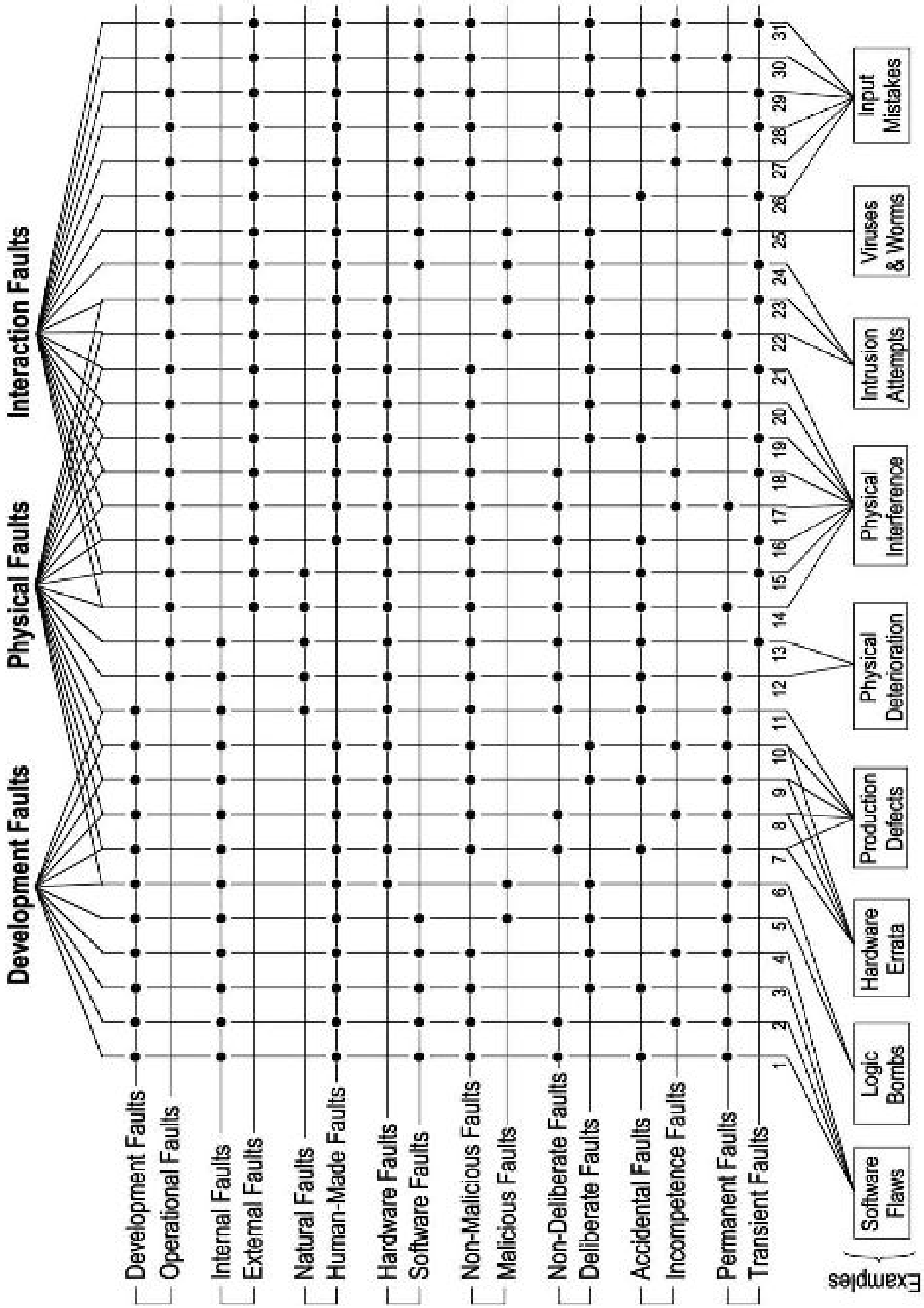


Figure 1.2: *"The classes of combined faults (a) Matrix representation."* (Fig. 5(a) in Avizienis et al. (2004) p. 16)

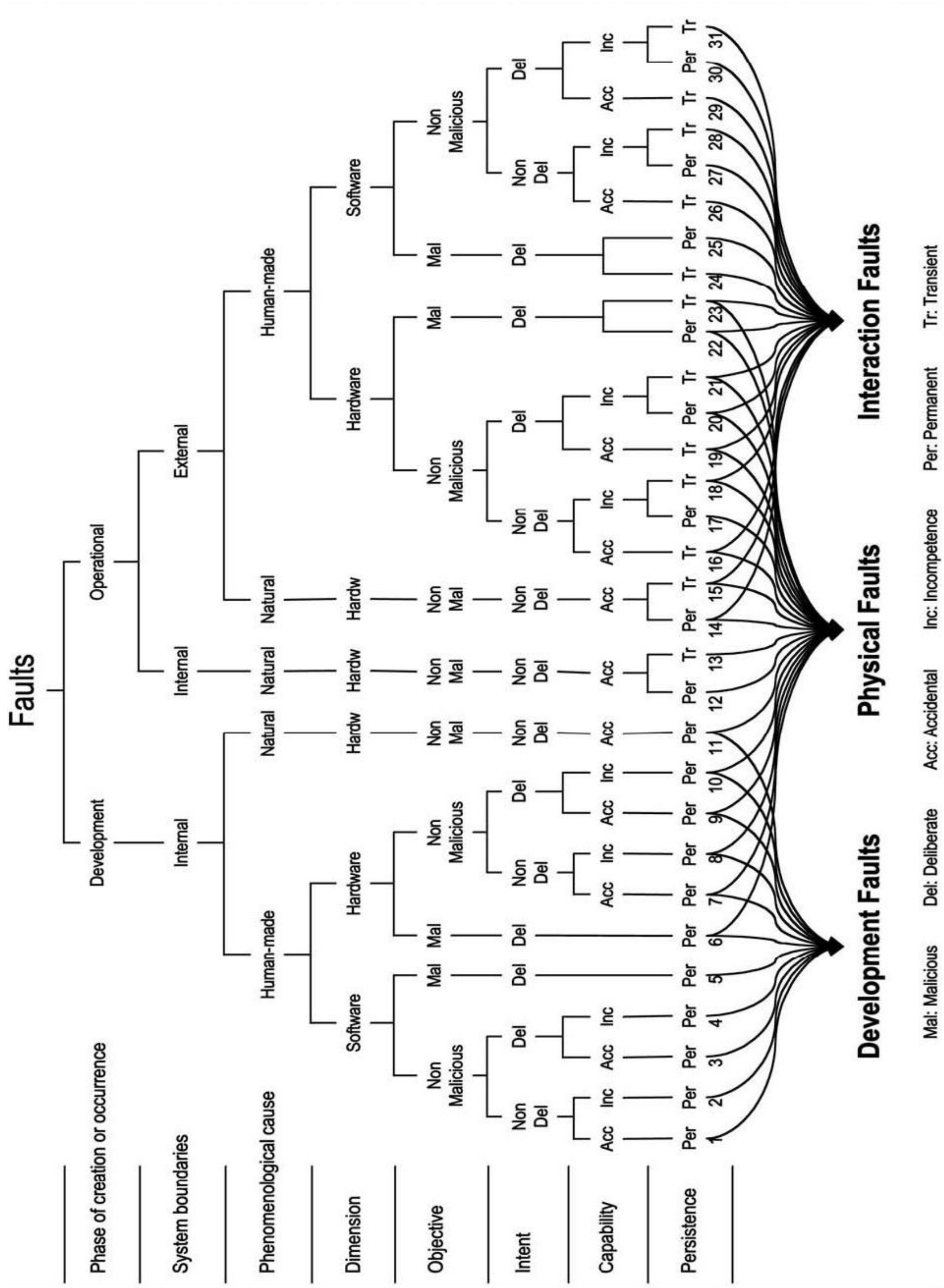


Figure 1.3: *"The classes of combined faults (b) Tree representation."* (Fig. 5(b) in Avizienis et al. (2004) p. 16)

It is important to note that this taxonomy of "Fault" that classifies all faults that may affect a system during its life, seems to suggest *multiple classification criteria* for all faults in this domain of discourse, such as: (a) the 16 elementary fault classes from the 8 basic viewpoints, (b) the 31 likely combinations of the elementary fault classes, (c) the 9 examples of known faults; or even (d) the three major overlapping groupings that the 31 likely combinations of fault belong to. And it is these multiple classification criteria of a particular domain concept and the possible approaches to model them in a given ontology that will narrow even further the focus of this research.

1.1.3 The "Fault" Ontology in ReSIST

Figures 1.1, 1.2 and 1.3 illustrate the different classification of the concept of "Fault" that should be captured in the D&S ontology for ReSIST. The background rationale of the figures in the context of dependable and secure computing can be further studied in Avizienis et al. (2004).

The ReSIST project intended several uses of the "Fault" ontology model including:

Scenario (a): the representation and classification of instances of faults in real world systems; and

Scenario (b): as a terminology or keyword index for publications, projects, research interests and the resilient mechanisms of computer systems.

Scenario (a) and *(b)* are specifically put forward because forthcoming sections throughout this work will refer back to them, discussing their involvement on the research carried out and how they are addressed.

1.2 Problem

This section outlines the problem of modeling multiple classification criteria and multiple inheritance in the framework of this research. That is, handling this aspect inherent to certain domain concepts when these are represented in an ontology model to be deployed in the Semantic Web. The section provides as well, some evidence on the percentage of existing ontology models currently available on the Web that might be subject this problem.

1.2.1 Multiple Classification Criteria

General examples of domain-specific concepts that exhibit the characteristics of multiple classification criteria described so far abound, going from a "Dishwasher Detergent" as in

Denton (2003), (which includes classification criteria such as “brand”, “scent”, “form”, etc.); to a “Sock” as presented by Broughton (2006), (which could be classified based on “material”, “function”, “length”, etc.). The list of examples can go on.

There are other examples that are particularly interesting because they are used in well-known ontology development literature using OWL and as it will be revealed throughout this research, they fit into the modeling scenario presented as well. They include: “Wine” by Welty et al. (2004), “Person” (in the context of family history relations) by Krötzsch et al. (2009), or “Pizza” by Horridge et al. (2009).

In the “Wine” ontology model, Welty et al. (2004) consider classification criteria such as: wines by type of grape, wines by region, wine by color, etc. Krötzsch et al. (2009) look at the concept of “Person” in the family history ontology model, based on: person by gender and person by type of kinship relation (parent, child, sibling). In the case of the “Pizza” ontology model, Horridge et al. (2009) think of pizzas in terms of: pizza by type of base, pizza by type and number of toppings and pizza by country. However, in none of them they refer explicitly to the various classification criteria of the domain concept they target, nor attempt to represent these criteria explicitly in the respective ontology models developed. Classification criteria are not discussed and they are modeled implicitly.

1.2.2 Multiple Inheritance

One assumption was made in the process of surveying the existing methodologies to build ontologies from scratch and the modeling of domain concepts subject to multiple classification criteria. The assumption is that a modeling scenario involving multiple classification criteria is very likely to lead to a scenario of *multiple inheritance*.

To illustrate this claim, consider for example an individual fault that belongs to the “Fault” universe of discourse described in previous sections: an individual fault in the real world that is a type of “Virus & Worm” (one of the known examples of faults presented at the bottom of Figure 1.2). Based on the relationship between “Virus & Worm” faults and the 16 elementary fault classes from the 8 basic viewpoints of the taxonomy of “Fault” visible in the matrix of Figure 1.2, a particular “Virus & Worm” individual fault is also a fault of type: “Operational”, “External”, “Human-made”, “Software”, “Malicious”, “Deliberate” and “Permanent”. Thus, every individual that belong to the “Virus & Worm” class of faults, is also a member of the “Operational”, “External”, etc. classes of fault, which is a typical case of multiple inheritance.

A similar rationale could be drawn for the domain concepts already mentioned, “Pizza”, “Wine” ,”Person”, “Dishwasher Detergent”, “Sock” or the many more open ended list of concepts prone to be characterized based on multiple classification criteria (or by extension, multiple inheritance) of key aspects that define them.

To obtain a better idea of the multiple inheritance landscape for the ontologies in the Web, Wang et al. (2006) shows the shape of class hierarchies for 1275 ontology files sampled in the survey, (688 OWL and 587 RDFS ontologies). Out of the 688 OWL ontologies, 122 (17.7%) were Directed Acyclic Graphs (DAGs), and 64 (9.3%) were multitrees[2]. This gives a total of 27% were most likely some type of multiple inheritance modeling in their class hierarchy is taking place. In the inferred ontology this number goes up to 30.2%. In the case of RDFS ontologies, out of 587 included in the survey, a total of 77 (13%), had a DAG (6.8%) or a multitree (6.3%), as the shape of their class hierarchy.

The combined result is that about 20% of all ontologies on the Web considered in the survey include some type of multiple inheritance modeling scenario. This value seems too low based on how common multiple inheritance occurs in the real world [Meyer (2000); Sowa (2000)]. Or, as John Sowa puts it in White et al. (2008)[3]:

> "Every animal, vegetable, and mineral on planet earth can be classified in an open-ended number of ways. Arbitrarily picking one and prohibiting the others is unnatural, confusing, and horribly difficult to use".

A possible interpretation for this could be due to a lack of best practice guidelines on how to model this problem, which in turn could be causing ontology developers to find creative ways to circumvent it.

On a similar study by d'Aquin et al. (2007), surveys indicate that the number of ontologies and their presence in the traditional Web increases rapidly according to the latest figures. The number of OWL and RDF-S ontologies available online is approximately 6200 and 1700 respectively (see Figure 1.4). These numbers are in the order of nearly ten times larger in the case of OWL ontologies and more than double for RDF-S when compared to the survey in Wang et al. (2006) about a year earlier. The latter reported 688 and 587 OWL and RDF-S ontologies respectively.

This seems to indicate that since the adoption of the OWL specification language as a W3C standard in 2004 [Dean and Schreiber (2004); McGuinness and van Harmelen (2004); Welty et al. (2004)], the ontology development community has been active and embraced the latest technology available in a detriment to its RDF-S predecessor. More importantly, it brings an interesting question to the forefront. How are these ontologies being built? What modeling problems and challenges are ontology developers facing and what approaches are they taking to solve them? As the number of ontologies present in the Semantic Web increases, the more important is to have in place guidelines to facilitate their construction and strengthen their processes to deliver the intended ontology artifact.

[2]Multitrees can be seen as a directed acyclic graph where each node can have a tree of ancestors and

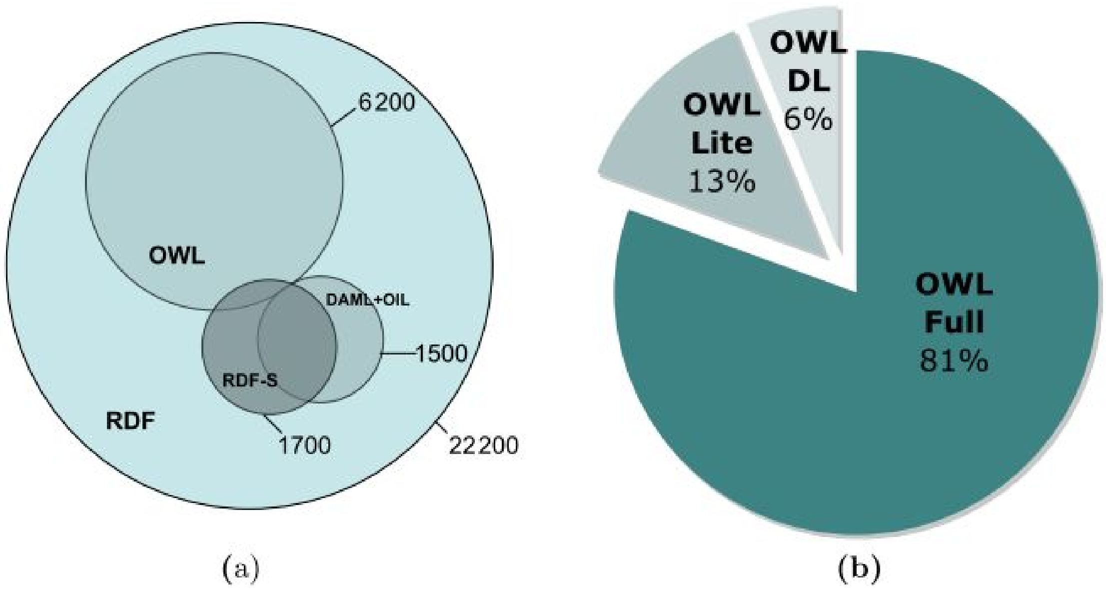


Figure 1.4: *"Usage of the ontology representation languages (a) and of the three OWL species (b)."* (Fig. 1 of d'Aquin et al. (2007) p. 3)

1.3 Approach

The starting point of this research involves conducting a survey of the existing methodologies to build ontologies from scratch in the context of the Semantic Web, with the objective of using the guidelines in place to model multiple classification criteria (or even multiple inheritance) of a target domain concept. The survey will focus mainly in the area of Ontology Engineering, with an emphasis on the topic of Ontology Design Patterns, and the areas of Object-Oriented Design and Faceted Classification due to their relevance with the modeling problem.

1.3.1 Ontology Engineering

Ontologies have emerged as one of the key components needed for the realization of the Semantic Web vision [Berners-Lee et al. (2001); Shadbolt et al. (2006); Alani et al. (2008)] and they bring with them a broad range of development activities that can be grouped into what is called Ontology Engineering [Hoekstra (2009); Gomez-Perez et al. (2004)]. A detailed overview of what an ontology is, including the evolution of its definition in the literature, can be found in Hoekstra (2009)(Chapter 4) or Gomez-Perez et al. (2004)(§ 1.2).

Ontology Engineering for the Semantic Web is a very active research area and has experienced remarkable advancements in recent years, although it is still relatively new

a tree of children. There cannot be a diamond structure in a mulitree [Wang et al. (2006)]

[3] http://ontolog.cim3.net/forum/ontolog-forum/2008-05/msg00051.html

compared to other engineering practices within Computer Science or other fields. The NeOn Project[4], whose aim is to advance the state of the art in using ontologies for large-scale semantic applications in distributed organizations, with the involvement of its 14 partners, has attempted to build consensus in the ontology research community regarding the processes and activities that are part of the Ontology Engineering field. Figure 1.5 illustrates the glossary of processes and activities that resulted from this endeavor. A brief description of what each activity entails is provided by Suarez-Figueroa et al. (2008); Suarez-Figueroa and Gomez-Perez (2008b,a). The glossary is not intended to be exhaustive given that new activities continue to appear as ontologies and the applications they are used for, continue evolving. It can be seen as an updated review of previous work by Gomez-Perez et al. (2004); Fernandez-Lopez et al. (2002, 1997).

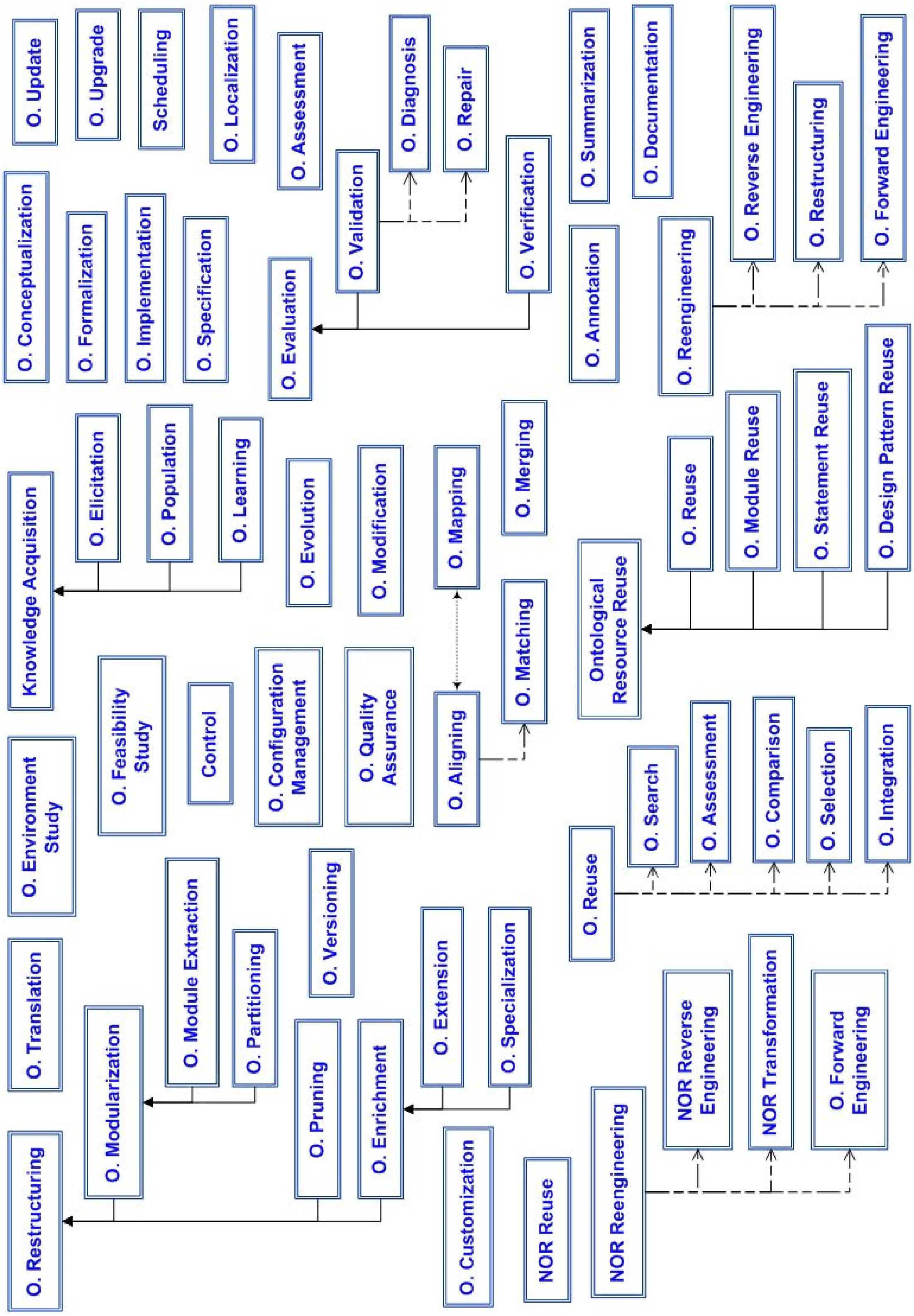
A constant ongoing effort in Ontology Engineering deals with harnessing the field with sound development practices analogous to those successfully employed in Software Engineering for decades. Some notable adaptations would include those by Vrandecic and Gangemi (2006), Gomez-Perez et al. (2004), Sure et al. (2003), Devedzić (2002) or Fernandez-Lopez et al. (1997). One of the objectives of this effort is to address areas of the ontology development process vulnerable to *ad-hoc* practices that could lead to unexpected or undesirable results in ontology artifacts, similar to the findings gathered by Rector et al. (2004) or Poveda et al. (2010).

The experience during my involvement in the ReSIST Project indicates that the conceptualization and representation of multiple classification criteria of domain concepts in the context of the Semantic Web, (such as "Fault" in the D&S ontology), is one of such areas in the Ontology Engineering field that is vulnerable to ad-hoc practices. As the literature survey accompanying this research suggests, there seems to be a lack of specific design guidelines for the ontological conceptualization and representation of the modeling scenario described, leaving ample room for ad-hoc practices and their negative consequences.

For example, common misconceptions when trying to represent several classification criteria are to use subsumption relations between classes when in fact a *part-of* relation would be in order, or to use subsumption to model relationships that are outside OWL DL expressivity altogether (i.e. a *meta-class* – a class whose elements are other classes[5] as presented by Foxvog (2005)).

Using the NeOn glossary of Ontology Engineering processes and activities in Figure 1.5 as a guide, the focus of this work is set on Ontology Conceptualization, Ontology Formalization, Ontology Implementation and (with a special emphasis) Ontology Design Patterns (ODPs) [Hammar and Sandkuhl (2010); Egana-Aranguren (2009); Hoekstra (2009); Allemang and Hendler (2008); Presutti et al. (2008); Gangemi (2005); and the

[4] http://www.neon-project.org
[5] http://ontolog.cim3.net/forum/ontolog-forum/2010-04/msg00066.html



- solid line arrows denote “type of”
- dashed line arrows denote “a process divided into activities”
- dotted line arrows denote “synonymy”

Figure 1.5: *“NeOn Glossary of Processes and Activities”.* (Figure 4 in Suarez-Figueroa et al. (2008) p. 15)

Semantic Web Best Practices and Deployment Working Group (SWBPD-WG)[6] of the World Wide Web Consortium W3C (2004–2005)].

Ontology Design Patterns have evolved within Ontology Engineering from the notion of *design pattern*, defined in Gangemi (2005) as "archetypal solutions to design problems in a certain context" and they are justifiably receiving a significant amount of attention by ontologists due to the preceding success achieved by design patterns in the context of Software Engineering [Gamma et al. (1995)]. ODPs will be reviewed thoroughly as part of Section 2.

1.3.2 Object-Oriented Design and Faceted Classification

Not only the area of Ontology Engineering and Ontology Design Patterns was explored. Given the characteristics of the modelling issue, two other areas were included in the survey because of their relevance: (a) Object-Oriented Design; and (b) Faceted Classification in the field of Library and Information Science (LIS).

Object-Oriented Design [Meyer (2000); Rumbaugh et al. (1991)] was included because of the prominent role that multiple inheritance has in this area, where its pros and cons has been discussed at length during decades. In addition, Gamma et al. (1995) introduced the paradigm of design patterns to Object-Oriented Design receiving a significant amount of attention and justifiably becoming the precursor and a point of reference for Ontology Design Patterns in Ontology Engineering.

Faceted Classification [Ranganathan (1960); Vickery (1960); Broughton (2004, 2006)] was considered because it represents a natural fit to the notion of multiple classification criteria. In fact, Ranganathan (1960) conceived Facet Classification as a consequence to his discontent with the inability of traditional enumerative bibliographic classification systems, such as the Dewey Decimal Classification (DDC)[7] and the Library of Congress Classification (LCC)[8], to support compound subjects. Subjects that could be classified according to multiple views, topics, attributes or *criteria*.

One of the main topics covered as part of the related research review in Section 2, is how these three different fields of practice, Ontology Design Patterns, Object-Oriented Design and Faceted Classification, overlap on their approach to address multiple classification criteria in the domain concepts of the conceptual models that they produce.

[6]http://www.w3.org/2001/sw/BestPractices/
[7]http://www.oclc.org/dewey/
[8]http://www.loc.gov/catdir/cpso/lcco/

1.4 Thesis Statement

As stated in previous sections, ontologies are one of the key components for the realization of the Semantic Web vision and the number of new ontologies being built is rapidly increasing in recent years. A constant ongoing effort of the Ontology Engineering community is to strengthen the field with sound development practices similar to those successfully employed in Software Engineering already, aiming at minimizing the opportunities for ad-hoc practices that could lead to suboptimal ontology artifacts. A common trait of many domain concepts being represented in these ontologies, is that they can be modeled based on multiple classification criteria (or in some cases, by extension, multiple inheritance) of key characteristics that define them. Under these premises, this thesis tries to provide answers to the research question below:

> ***Research Question 1***: Are there consistent and systematic techniques and guidelines to represent multiple classification criteria (or to some extent multiple inheritance) of domain concepts in ontology models suitable for deployment in the context of the Semantic Web? What could be learnt from fields such as Object-Oriented Design and Faceted Classification, which have already been exposed to the design of multiple classification criteria conceptual models for much longer than Ontology Engineering?

Ideally, the outcome of this endeavor seeks to provide the Ontology Engineering community means to strengthen the representation of multiple classification criteria of domain concepts, based on the use of consistent and systematic modeling decisions in detriment of ad-hoc practices.

A sensible approach to address *Research Question 1* is to explore the use of Ontology Design Patterns (ODPs). Design-pattern driven ontology construction, whether manual or (partially) automated, relies on the availability of curated repositories of ODPs adequately characterized. However, patterns that are not fully detailed, or that leave opportunity for ambiguity, may not be applied properly or consistently, which can lead to interoperability issues among the ontology models involved. Under these premises, and in the context of *Research Question 1*, this thesis considers answering the following:

> ***Research Question 2***: Are there ODPs that could be applied to represent multiple classification criteria of domain concepts? If so, are they fully detailed or is there opportunity for ambiguity? In the case of having several ODPs, how do they relate to each other and what could be learnt from this?

In order to bring all this work full circle from the generic to the specific, it will be also important firstly, to evaluate the findings to these research questions; and secondly, test how they fit into the context of the "Fault" domain concept of the D&S ontology of ReSIST. That is, both *Scenario (a)* and *(b)* as described in Section 1.1.3.

In summary, the mission of this thesis could be stated as follows:

> "Achieve a significant advancement in the current Ontology Engineering state of the art, with regards to the modeling of domain concepts prone to be characterized by *multiple classification criteria* (or by extension, *multiple inheritance*), suitable for deployment in the *Semantic Web*. Such advancement, seeks to promote the use of existing or new Ontology Design Patterns, learning from fields of practice already familiar to the conceptualization of *multiple classification criteria*, specifically Object-Oriented Design and Library Information Science".

1.5 Contribution

There are several novel contributions in the work presented here that can not be found in the existing literature. This section details all of them and for traceability purposes, highlights: (a) where they appear throughout this thesis; (b) previous work published by myself et al. relevant to a given contribution, (also available in the front matter section "Declaration of Authorship" in chronological itemized format); and (c) the research question(s) they target from Section 1.4.

Table 1.1 summarizes these novelties together with the traceability information mentioned above and serves as a schematic guide to the subsections that follow. For convenience, the acronyms and abbreviations used, are reproduced here: Research Question (RQ), Ontology Design Pattern (ODP), Faceted Classification Scheme (FCS), Class as Property Value (CPV), Value Partition (VP), Normalisation (Norm.), Dependability and Security (D&S).

Faceted Classification Scheme ODP

Perhaps the most notable contribution is a new reengineering Ontology Design Pattern (ODP) to transform an existing Faceted Classification Scheme (FCS) of a given concept from a specific domain, into an OWL DL ontology model (Chapter 6). Leveraging on the methodology of *facet analysis* and the existing Normalisation ODP, the new FCS reengineering ODP puts forward a systematic guideline to convert a well-known, widely spread non-ontological resource such as a FCS, into an ontological model that meets the best practice criteria set out by the Normalisation ODP and thus, suitable for deployment in the *Semantic Web*. This guideline provides a partial solution to one of the challenges in Ontology Conceptualization, Formalization and Implementation: minimize

Contribution	Content	Research Question	Previous Publication
Faceted Classification Scheme ODP	Chapter 6	RQ 1	Rodriguez-Castro et al. (2010b,a, 2009)
Visual notation of ODPs	Section 3.1	RQ 2	Rodriguez-Castro et al. (2010b,a)
Decoupling of the CPV ODP	Chapter 3	RQ 2	
Alignment of VP and CPV ODPs	Chapter 4	RQ 2	Rodriguez-Castro and Glaser (2008c,b,a)
Alignment of Norm., CPV and VP ODPs	Chapter 5	RQ 2	
Evaluation of FCS ODP ("Dishwasher Detergent", "Pizza", "Wine" and "Fault" ontology models)	Chapter 6, 7, 8, 9	RQ 1	
"Fault" ontology model (ReSIST D&S ontology)	Chapter 9	RQ 1, 2	Rodriguez-Castro and Glaser (2008c,b,a); Glaser et al. (2007b); Millard et al. (2006)

Table 1.1: Summary of contributions.

the opportunity for ad-hoc practices that could lead to unexpected or undesired ontology model artifacts when representing multiple classification criteria of a target domain concept. Chapter 6 initially evaluates the new pattern using an existing FCS example in the domain of "Dishwasher Detergent".

The contributions in Chapter 6 cater to answer *Research Question 1*, and were first published as a conference research paper in Rodriguez-Castro et al. (2010b) and condensed as a workshop poster in Rodriguez-Castro et al. (2010a). It is worth noting that both references were well received by the community at their respective venues, given that the former was nominated as *"Best Student Research Paper"*[9], and the latter received the distinction of *"Best Pattern Award"*[10]. One more preliminary effort that led to the Faceted Classification Scheme ODP in Chapter 6, was published as a workshop poster in Rodriguez-Castro et al. (2009). This poster explored a specific pattern in Object-Oriented Design to model multiple classification criteria and a tentative adaptation to the Ontology Engineering field. Thus, fitting as well into the resource portfolio to address *Research Question 1*.

The completion of Chapter 6 was possible thanks to the concatenation of a series of supplementary, incremental, and partial contributions that take place in various existing ODPs. These are summarized below.

[9] http://iswc2010.semanticweb.org/awards

[10] http://ontologydesignpatterns.org/wiki/WOP:2010#Accepted_patterns

Visual Notation of ODPs

Introduction of a simple visual *notation*, yet expressive enough, to illustrate the generic structure and elements that characterize all ODPs covered. One of the motivations for this notation is to facilitate the visual side-by-side comparison of the conceptual elements of several ODPs simultaneously. This type of comparison is important because it helps to identify structural and semantic relationships, or the lack thereof, across different ODPs. Section 3.1 details this notation in full and a preliminary version was already used in Rodriguez-Castro et al. (2010b,a).

Decoupling of the CPV ODP

The characterization of the generic structure of the Class As Property Value (CPV) ODP as featured in Approach 4 of Noy (2005) (Chapter 3). This structure generalizes the possible uses, implementation and applicability of the pattern and it allows to decouple Approach 4 of the CPV ODP into two versions: (a) a most generic version, where the meaning of a class used as value of a property is re-interpreted or overloaded; and (b) a simplified version, where the meaning of a class used as value of a property is preserved.

Alignment of the VP and CPV ODPs

The characterization of the generic structure of the Value Partition (VP) ODP featured in Rector (2005), Presutti et al. (2008) and Egana-Aranguren (2009) (Chapter 4). The generic structure is presented in a similar notation to that of the Class As Property Value ODP. This allows to perform a comparative analysis between the two patterns identifying: (a) differences and similarities between the two at the structural and semantic level; and (b) how the Value Partition ODP can be described as a refinement or specialization of the simplified version of the Class As Property Value ODP where the meaning of a class used as value of a property is preserved.

A prelude to the work concerning the relationship between the CPV and the VP ODPs in Chapters 3 and 5, was originally published in Rodriguez-Castro and Glaser (2008c,b,a).

Alignment of the Norm., VP and CPV ODPs

The characterization of the generic structure of the Normalisation ODP as introduced in Rector (2003) and detailed in Egana-Aranguren (2009) (Chapter 5). The generic structure of the Normalisation ODP described here, expands that found on the literature by identifying: (a) how to include multiple *modules* or *semantic axes* in the application of the pattern; and (b) all the possible implementations of the pattern in OWL. The examples of ontology normalisation in the literature dealt with only one module or semantic axis and with only one of the possible implementations in OWL, making it unclear how to use or apply the pattern outside of those constraints.

Thanks to this characterization of the generic structure of the three ODPs under a similar notation it is possible to perform a comparative analysis among all three of them (Chapter 5). As a result, the analysis reveals significant ontological alignments in the main elements of the patterns that indicate mainly: (a) how a single instantiation of the Normalisation ODP can be formed by combining together multiple instantiations of the Value Partition ODP and(or) the Class As Property Value ODP; and (b) how a similar set of OWL idioms is employed by three different ODPs to handle three different target modeling scenarios.

Contributions contained in the previous three sub-sections together with the current one, seek to collectively address *Research Question 2* by expanding the characterization and understanding of the three ODPs revisited and compared.

Evaluation of FCS ODP

There are various examples to evaluate the new FCS ODP. The first example can be found in Chapter 6, and uses an existing Faceted Classification Scheme in the domain of “Dishwasher Detergent” to build the associated ontology model from scratch. Part of this evaluation was previously featured in Rodriguez-Castro et al. (2010b).

The second is located in Chapter 9. The target domain is the concept of “Fault”, and in this case a new Faceted Classification Scheme is built from scratch as well as the corresponding ontology model as per the FCS ODP.

The third and fourth examples correspond to Chapters 7 and 8 respectively. The approach here is slightly different. Two popular existing ontology models in the Ontology Engineering literature, one for the concept of “Pizza” and another for the concept of “Wine”, are analysed and decomposed to reveal the underlying multiple classification criteria (and hence, the underlying Faceted Classification Scheme) that both ontology models implicitly include. The purpose of this approach is to demonstrate: (a) the alignments identified between a generic FCS and the Normalisation ODP that allowed the creation of the FCS ODP; and (b) that the FCS ODP could also be applied to build both ontology models.

All of these examples are presented as a partial solution to the issues raised within *Research Question 1*.

"Fault" Ontology Model

Chapter 9 is put forward to test all the contributions claimed throughout this work to address the problem of modeling multiple classification criteria of ontology domain concepts, promoting a pattern-driven ontology construction. To bring everything together, the domain concept chosen is the "Fault" ontology model developed for the D&S ontology in the ReSIST project. Once again, the FCS ODP is applied to deliver the ontology model of "Fault", and the various alignments identified among the Class as Property Value, the Value Partition, and the Normalisation ODPs are exhibited as part of the ontology building process.

The purpose of Chapter 9 is to address both *Research Question 1* and *2*, and using the "Fault" concept as a use case, seeks to do so in the framework of both *Scenario (a)* and *(b)* as outlined in Section1.1.3.

An early version of parts of the evaluation dealing with the alignment between ODPs can be found in Rodriguez-Castro and Glaser (2008c,b,a), while part of the background that motivated *Scenario (a)* and *(b)* of Section1.1.3 is featured in Glaser et al. (2007b); Millard et al. (2006).

1.6 Thesis Structure

The additional material contained throughout this document is organized as follows:

Chapter 2 presents a brief overview of relevant work in the areas of ontology creation methodologies, Ontology Design Patterns (ODPs), Object-Oriented Design, and Faceted Classification regarding how they handle design scenarios that involve multiple classification criteria of concepts and to some extent multiple inheritance.

Chapter 3 revisits a well known pattern in ontology design: the Class As Property Value ODP. It presents the elements that form the pattern in a generalized structure, and it characterizes the role of two key notions: *interpretation* and *terminology*. It then, generalizes the pattern showing how it can be used to accommodate multiple interpretations, multiple terminologies, or on the contrary, how the notion of interpretation and terminology can be conflated into a single element.

Chapter 4 revisits another popular pattern in ontology design: the Value Partition ODP. It presents the elements that form the pattern in a generalized structure, and it goes through a comparative analysis that results in identifying interesting alignments between the Value Partition and the Class As Property Value ontology models.

Chapter 5 revisits the Normalisation ODP following a similar procedure to that in Chapters 3 and 4. It characterizes the generic structure of the pattern and performs a comparative analysis among this and the Value Partition and Class As Property Value ODPs. The result puts forward additional alignments among the three models very relevant to the overall aim of this work.

Chapter 6 introduces Facet Analysis and Faceted Classification. A different approach for the conceptualization of a domain from the field of Library and Information Science. The chapter characterizes the elements of a generic Faceted Classification Scheme (FCS) and again, a comparative analysis between this and the generic structure of the Normalisation ODP reveals a series of mappings between the two representation paradigms that allows to transform a given FCS into a normalised OWL DL ontology model. An existing example of a FCS in the domain of "Dish Detergent" is retrieved from a very significant paper by Denton (2003) and used to illustrate the process.

Chapters 7, 8 and 9 provide three examples of three different domain concepts ("Pizza", "Wine" and "Fault") that fit the modeling problem of the representation of multiple classification criteria. The first two examples, "Pizza" and "Wine" come along with an ontology model already built for a tutorial and educational purpose. They will be examined from a *reverse engineering* point of view, highlighting occurrences of the various ODPs revisited and introduced throughout this work; and how they align to or deviate from the generic structure of these patterns in the terms they were presented. On the other hand, the last example, "Fault", requires an ontology model to be built from scratch.

Chapter 7 examines a very popular ontology model in the ontology design literature. The model represents the concept of "Pizza" and it is used as a tutorial for the Protege[11] free open source ontology editor, illustrating at the same time the main features of the W3C OWL specification Horridge et al. (2004) Horridge et al. (2009).

Chapter 8, looks into the second ontology example, also well-known in the ontology bibliography. It represents the concept of "Wine" and it is used as an initial tutorial on how to create your first ontology Noy and McGuinness (2001) and as a guide to go through the features of the first version of the W3C OWL specification, OWL 1.0 Welty et al. (2004). Nonetheless, the ontology model used for this particular evaluation focuses on the latter in Welty et al. (2004).

The last evaluation example in Chapter 9, addresses the concept of "Fault" as introduced in the motivation section of this work and it builds the ontology model of "Fault"from scratch. The example presents how the background knowledge provided by Avizienis et al. (2004) to represent the concept of "Fault" fits into the generic structure of a Faceted Classification Scheme, and how this FCS can be converted into an OWL DL ontology model applying the transformation guidelines put forward in previous chapters.

Chapter 10 covers the conclusions gathered from this endeavor and proposes additional

[11] http://protege.stanford.edu/

opportunities for improvement and paths for further investigation.

Chapter 2

Related Research

This chapter presents a survey and a critical review, of previous work that can contribute to answer the research questions raised in Section 1.4. As stated in the introductory chapter, not only the area of Ontology Engineering has been covered in the context of the design problem under study, but also Object-Oriented Design and Faceted Classification from Library and Information Science (LIS). Figure 2.1 illustrates the overlap among the three conceptual modeling paradigms surrounding the scenario of *multiple classification criteria*. It is this particular overlap among the three that will be the target of the related research that follows.

2.1 Ontology Engineering

Ontology Engineering provides several methodologies and approaches to build ontologies from scratch. A comprehensive survey of the most relevant methodologies is provided

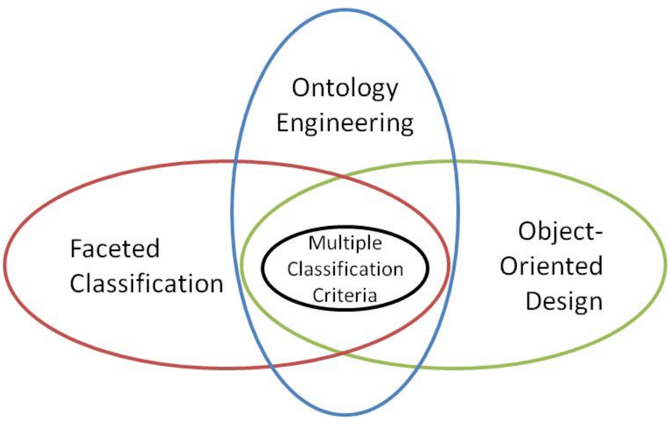


Figure 2.1: Overlap of Three Conceptual Modeling Paradigms regarding Multiple Classification Criteria.

by Fernandez-Lopez and Gomez-Perez (2002), Corcho et al. (2003), Pinto and Martins (2004), Cristani and Cuel (2005), Dahlem et al. (2009) and Dahlem and Hahn (2009).

The results from the various surveys indicate that different methodologies provide different levels of detail on the various activities that conform the ontology building development process. However, some of them do not look in detail into the activities of Ontology Conceptualization, Ontology Formalization or Ontology Implementation referred to in Section 1.3.1; and those that do, such as Gomez-Perez et al. (2004), Sure et al. (2003), Uschold and King (1995) or Gruninger and Fox (1995), do not discuss in depth the modeling problem subject of this research (or its possible solutions). They look at the ontology building process in broader terms, from a higher level perspective, or from the point of view of what role in the overall development lifecycle a given activity plays and what dependencies it has with others. In addition, the methodologies referenced above are dated prior to the adoption of the Web Ontology Language (OWL) by the W3C as the recommended ontology implementation language, thus, they do not take into account modeling elements specific to OWL. These factors cause these methodologies to deviate from the requirements behind the research questions in Section 1.4. They provide consistent and systematic steps to build an ontology model, but not at the required level of detail to transfer these consistent and systematic practices to address the modelling scenario of multiple classification criteria in the context of a Semantic Web built predominantly upon W3C standards.

A partial exception to the previous arguments is the methodology proposed by Noy and McGuinness (2001). The ideas behind their guide *"Ontology Development 101"* are inspired in the literature of object-oriented design, such as Rumbaugh et al. (1991). They propose a step by step methodology to develop an ontology from scratch but they also facilitate guidelines to questions more in line to the scope of this research, namely: how many siblings in a class hierarchy are too many and how few are too few? when to introduce a new class versus a new property? or a new class versus an individual? even the modeling of multiple inheritance is acknowledged. Although their guidelines throughout the guide are based on frame-based systems rather than on standard W3C Semantic Web technologies, their rationale could be easily extrapolated to the latter.

In summary, there are several ontology construction methodologies available in the literature, however in general they do not provide enough detailed information about the phases of Ontology Conceptualization, Formalization and Implementation. None of them treat these activities in the context of standard W3C Semantic Web technologies either and none of them addresses the specific scenario of modeling multiple classification criteria. Therefore, let us zoom in on the granularity level of the Ontology Engineering activities being considered and move to the area of Ontology Design Patterns (ODPs).

2.2 Ontology Design Patterns (ODPs)

Within the area of Ontology Engineering, an activity that is receiving a significant amount of attention is Ontology Design Patterns (ODPs). This is strongly due to the preceding success in the field of Software Engineering of the renowned book by Gamma et al. (1995) *"Design Patterns - Elements of Reusable Object-Oriented Software"*, (authors which are also known as the *Gang-of-Four* or *GoF*).

Similarly to how ODPs evolved from the concept of design patterns in Object-Oriented design introduced by Gamma et al. (1995), the latter were strongly influenced and inspired in the work of Alexander et al. (1977) in the domain of urban architecture and civil engineering. Alexander et al. (1977) initially defined a design pattern as:

> "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice".

This definition positions design patterns as a successful solution to a recurrent design problem. They document and record design experience, allowing practitioners, whether in urban architecture, object-oriented systems or now Ontology Engineering, not having to start a new design from scratch, developing an optimal design faster.

The notion of design pattern in the work of Alexander et al. (1977) and later in Gamma et al. (1995), is brought to the Ontology Engineering field by Gangemi (2005). The author already uses the term Ontology Design Pattern and puts forward a series of foundational and core patterns encoded using the W3C OWL language.

The characterization of ODPs in Gangemi (2005) was the precursor to the work in Presutti et al. (2008), producing one of the first catalogues of ODPs. The catalogue is available online[1], published as an open collaboration portal to foster discussion surrounding the various aspects of ODP development. Furthermore, Blomqvist (2007, 2008, 2009); Blomqvist et al. (2009, 2010) attempt to automate the ontology construction process relying on the repository of ODPs advanced by Gangemi (2005) and later expanded in Presutti et al. (2008).

Hoekstra (2009) provides a thorough survey of the Ontology Engineering landscape, motivated by the development of a core ontology for the legal domain, referred to as Legal Knowledge Interchange Format (LKIF) Core. The survey revisits some of the various ODPs introduced by Gangemi (2005) and formalize them further. In addition, Hoekstra (2009) discusses in detail three particular patterns used in the development LKIF Core, and although critical of viewing patterns as the *holy grail* of ontology building, the

[1] http://ontologydesignpatterns.org/

author acknowledges the benefits that their use bring into the ontology development process.

Egana-Aranguren (2009) considers ODPs as a key element to the vast knowledge representation needs in the field of bio-ontologies, ontologies used in life sciences. To assist bio-ontologists, Egana-Aranguren (2009) puts together a catalogue of ODPs built upon experiences in the development of large biological ontologies, with online[2] presence as well. The author demonstrates that the application of ODPs improves the quality of the resulting ontology models in various areas.

More pragmatic works aimed at ontology practitioners, are bringing ODPs from the research domain into the Semantic Web mainstream. Allemang and Hendler (2011, 2008) cover various ODPs at different levels of granularity in their review and examples of the core technologies involved in the development and deployment of Semantic Web applications and systems.

There is yet, another indicative factor of the attention that the Ontology Engineering community has devoted to ODPs recently. As Hammar and Sandkuhl (2010) reveals, the number of ODP publications in the research track and workshops of the three main Semantic Web conferences, namely the International, European (now Extended) and Asian Semantic Web Conference, has considerable increased since the 2005 edition until 2009 (year of the last edition sampled in the survey); going from the 4 papers in 2005 to 24 in 2009.

2.2.1 Public Catalogs of ODPs

For the purpose of this research two repositories of ODPs in the context of standard W3C Semantic Web technologies have been considered: (a) the public catalog of ODPs focused on the biological knowledge domain developed as part of Egana-Aranguren (2009); and (b) the library of ODPs developed by the NeOn Project[3] published in Presutti et al. (2008) and Suarez-Figueroa et al. (2007).

A significant contribution to both of these repositories, is the documents released by the W3C Semantic Web Best Practices and Deployment Working Group (SWBPD-WG)[4], namely: Noy (2005), Rector (2005), Rector et al. (2005) and Noy and Rector (2006). They already explored the paradigm of a design pattern solution to a given modelling problem as characterised by Gamma et al. (1995) but now in the context of semantic technologies W3C Standards.

The presentation and structure of the ODPs included in the library gathered in Suarez-Figueroa et al. (2007) and Presutti et al. (2008) can be seen as an extension and evolution

[2] http://odps.sourceforge.net/
[3] http://www.neon-project.org/
[4] http://www.w3.org/2001/sw/BestPractices/

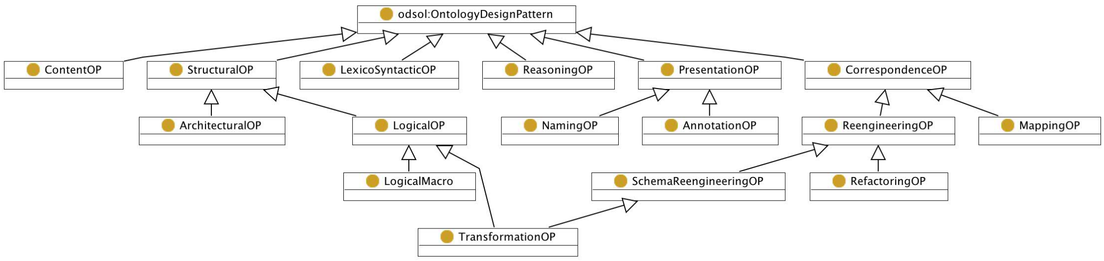


Figure 2.2: *"Ontology Design Patterns types"*. (Figure 2.2 in Presutti et al. (2008) p. 19).

to the work of Gangemi (2005). Gangemi defines the notion of Conceptual (or Content) Ontology Design Patterns (CODePs) and includes several examples, such as the Participation, Role-Task and Design-Artifact ODPs.

In Presutti et al. (2008) several levels of ontology patterns are discussed in the context of networked ontologies. They present six families of ODPs based on the kind of problem that they address and they can be represented at different levels of formality. These are: *Structural*, *Correspondence*, *Content*, *Reasoning*, *Presentation*, and *Lexico-Syntactic* ODPs.

The six families of ODPs can be seen at the first level of the taxonomy exhibited in Figure 2.2. The online catalog supporting the library of ODPs uses the taxonomy in Figure 2.2 to organize the various content in relation to the patterns.

Conversely, the public catalog of ODPs released as part of Egana-Aranguren (2009)[5], focused on the biological knowledge domain and it reflects experiences in the development of large ontologies in the area. This separate catalogue classifies all ODPs in three main different groups based on their functionality:

- Extension ODPs by-pass the limitations of the knowledge representation language, in this case OWL and they include: Exception, Nary Relationship, Nary DataType Relationship.
- Good Practice ODPs are applied to obtain a more robust, cleaner and easier to maintain ontology. They include: Entity-Quality, Entity-Property-Quality, Entity-Feature-Value, Selector, Value Partition, Defined Class Description, Normalization, Upper Level Ontology, Closure
- Domain Modelling ODPs offer ways of modelling concrete requirements of the domain being represented. They include: List, Adapted SEP (Structure - Entity - Part), Interactor-Role-Interaction, Sequence, Composite Property Chain.

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

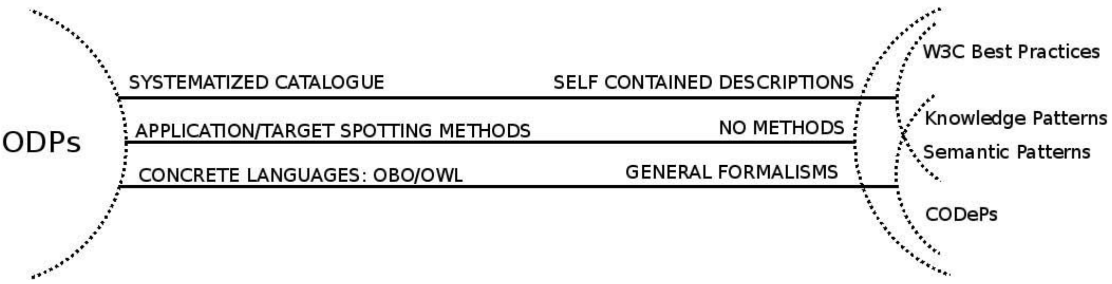


Figure 2.3: "Comparison of proposed ODPs (left) and previous work (right). Three criteria are used for comparison: application and target spotting methodologies, documentation and types of formalisms." (Figure 11 of Egana-Aranguren et al. (2008) p. 11).

Ironically, these two existing classifications of ODPs provide another ideal example of multiple criteria to classify the abstraction of a certain domain concept, in this case the "ODP" concept itself!

The differences between the classification of ODPs in Presutti et al. (2008) and Egana-Aranguren (2009) has to do with the criteria applied. The classification in Egana-Aranguren (2009) is based on *the way they are used*, while the classification in Presutti et al. (2008) is based on *the kind of problem they address*. Additional comparisons between both approaches are captured in Figure 2.3 originally included in Egana-Aranguren et al. (2008), where the author considers his catalog of ODPs complementary to the notion of Conceptual (or Content) Ontology Design Patterns (CODePs) in Gangemi (2005), which in turn can be seen similar to the notion of Content ODPs as described in Presutti et al. (2008).

Another aspect where the catalogs from both camps differ is in the template used to describe each pattern. The template in both Egana-Aranguren (2009) (initially introduced in Egana-Aranguren (2005)) and Presutti et al. (2008) is inspired in the original structure presented by Gamma et al. (1995), although they differ slightly in their respective adaptation. Table 2.1 provides a side-to-side comparison of all the sections identified in these three templates. Some of the sections mapped between the two OPD templates may not be an exact match however with simplicity in mind, sections with the most overlap were combined together.

It is reasonable to think that as the research surrounding ODPs consolidates and the application of these becomes common practice in the Ontology Engineering community, the two catalogs including their templates will unify and expand into a single repository.

<table>
<tr><th>Object-Oriented Design Pattern</th><th colspan="2">Ontology Design Pattern</th></tr>
<tr><td>Gamma et al. (1995)</td><td>Egana-Aranguren (2005)</td><td>Presutti et al. (2008)</td></tr>
<tr><td colspan="2">Name and classification</td><td>Name</td></tr>
<tr><td colspan="3">Intent</td></tr>
<tr><td colspan="3">Also known as</td></tr>
<tr><td colspan="2">Motivation</td><td>Example</td></tr>
<tr><td colspan="2">Applicability</td><td>Requirements</td></tr>
<tr><td colspan="2">Structure</td><td>Diagram</td></tr>
<tr><td>Participants</td><td>Elements</td><td rowspan="2">Elements</td></tr>
<tr><td>Collaboration</td><td>Relationships</td></tr>
<tr><td colspan="3">Consequences</td></tr>
<tr><td colspan="2">Implementation</td><td>n/a</td></tr>
<tr><td colspan="2">Sample code</td><td>Building block</td></tr>
<tr><td colspan="3">Known uses</td></tr>
<tr><td colspan="3">Related patterns</td></tr>
<tr><td rowspan="3">n/a</td><td colspan="2">References</td></tr>
<tr><td>n/a</td><td>Extracted from</td></tr>
<tr><td>Additional information</td><td>n/a</td></tr>
</table>

Table 2.1: Side-to-side comparison of sections in templates to describe design patterns by author.

2.2.2 Conclusions

From all the ODPs explored, three in particular stand out for their applicability to the representation of multiple classification criteria (or multiple inheritance) of domain concepts. These are:

- The Class As Property Value ODP introduced by Noy (2005) and referred to in Suarez-Figueroa et al. (2007)(§ 2.1.1.4) and Presutti et al. (2008)(§ 2.2.1).
- The Value Partition ODP introduced by Rector (2005), developed further in Egana-Aranguren (2005)(§ 4.3.2.2) and Egana-Aranguren (2009)(§ A.17); and referred to in Suarez-Figueroa et al. (2007)(§ 4.2.16).
- The Normalisation ODP introduced by Rector (2003), and shaped further into a pattern structure in Egana-Aranguren (2005)(§ 4.3.2.1) and Egana-Aranguren (2009)(§ A.13).

The importance of the first two patterns stems from the aim by both of using a taxonomy of concepts in the ontology model to annotate or characterize a separate set of concepts placed in a separate taxonomy of the same ontology. In fact, both use the same technique of employing *anonymous individuals* as values for the properties that relate the concepts from the separate taxonomies. Despite these similarities there are subtle differences between the two, which implications have to be considered. These similarities, differences and their implications in the context of domain concepts prone to be represented based on the multiple classification criteria that define them, are discussed and presented in subsequent chapters thanks to a comparative analysis between the two patterns.

The importance of the Normalisation ODP stems from its aim to automate the management of *multiple inheritance* relations among the concepts in the ontology model. As described in Section 1.2.2, multiple inheritance (and hence poly-hierarchies in the ontology) can be a symptom of the existence of *multiple classification criteria* in the representation of the concepts that participate in such relations.

The Normalization ODP analyzes the implications of having a high number of multiple inheritance relations and it refers to the notion of modeling different *semantic axes* as the cause that can lead to poly-hierarchical structures or a *tangled* ontology. It then outlines a effective step by step procedure that would *untangle* the ontology becoming a collection of independent modules easy to maintain.

However, the normalisation mechanism focuses more on the implementation side of the modeling problem. It looks at the consequences of having multiple *semantic axes* but not as much at the *ontological* aspects that may have introduced those axes in the first place or at the characteristics that those axes may present. The notion of *semantic axis* seems closely related to what it is referred hereto as *classification criterion.*

Interestingly enough, there are also similarities in the generic structure of the Normalisation ODP when compared to the previous two patterns: Class As Property Value and Value Partition. Again, a comparative analysis among the three of them reveals key ontological alignments in the main elements of the patterns such that a single instantiation of the Normalisation ODP can be formed by the combination of multiple instances of the other two patterns. The results of this analysis is presented in the chapters to follow.

2.3 Object-Oriented Design

Another topic of research involved in the modeling of numerous classification criteria is multiple inheritance. Multiple inheritance is often the most common manifestation of multiple classification criteria and it has been an aspect of extensive research in the field of Object-Oriented Design and programming.

However, there are crucial differences between the field of object-oriented application design and ontology construction that condition to what extent the findings in the object-oriented paradigm can be extrapolated to the ontology modeling world. Noy and McGuinness (2001) already acknowledge these differences even though their methodology to build an ontology model was inspired in object-oriented design principles such as Rumbaugh et al. (1991).

A detailed discussion regarding the differences between the two disciplines takes place in Oberle et al. (2006) which covers ontology development from an object-oriented developer point of view. Table 2.2 reproduces Section 3.3 of the cited reference, which summarizes the main similarities and differences between Object Oriented languages and standard W3C Semantic Web languages.

2.3.1 Multiple Inheritance (MI)

A very informative analysis regarding the need for MI in object-oriented languages, and in the C++ language particularly, takes place in Cargill (1991) and Waldo (1991). Cargill claims *no* need for MI based on the lack of an example that will prove the requirement for it and provides comprehensive mechanisms that do not require MI to achieve the same functionality. Waldo on the other hand, identifies three different types of MI: *implementation*, *interface* and *data*. He defines each one of them as follows:

Implementation inheritance. It is characterized as the relationship a derived class has with its base class when some of the functions of the derived class are delegated to functions that have been implemented in the base class.

Interface inheritance. The reason for using this sort of inheritance is to allow the same functional interface to be presented by all objects that are members of classes that derive from that class.

Data inheritance. It allows the derivation of a new class that shares only data members with no implication that the functions that can be called on instances of such a derived class or the behavior of those instances will have anything in common with the base.

According to this distinction, Cargill is solely referring to *implementation inheritance*. At the same time, Waldo provides a compelling example of interface and data MI that cannot be addressed by Cargill alternatives which sustains the need for the feature in the C++ language.

Unlike in the case of C++, the Java object-oriented language opted for not allowing multiple inheritance across classes. In Java, a class can only inherit behavior and implementation from a single parent class. However, Java introduces the concept of interface conformance. Java interfaces could be seen as abstract classes, (where no implementation is provided). Java allows classes to implement or conform to multiple interface classes, which in turn can provide certain support for the type of multiple inheritance labeled by Waldo (1991) as *interface inheritance*. Tempero and Biddle (2000) provides an overview of different implementation techniques to simulate MI in the Java language and the limitations that still exists. The MI simulation is achieved by combining single inheritance, delegation and interface conformance.

Object-Oriented Languages	OWL and RDF
Domain models consist of classes, properties and instances (individuals). Classes can be arranged in a subclass hierarchy with inheritance. Properties can take objects or primitive values(literals) as values.	
Classes and Instances	
Classes are regarded as types for instances.	Classes are regarded as sets of individuals.
Each instance has one class as its type. Classes cannot share instances.	Each individual can belong to multiple classes.
Instances can not change their type at runtime.	Class membership may change at runtime.
The list of classes is fully known at compile-time and cannot change after that.	Classes can be created and changed at runtime.
Compilers are used at build-time. Compile-time errors indicate problems.	Reasoners can be used for classification and consistency checkingat runtime or build-time.
Properties, Attributes and Values	
Properties are defined locally to a class (and its subclasses through inheritance).	Properties are stand-alone entities that can exist without specific classes.
Instances can have values only for the attached properties. Values must be correctly typed. Range constraints are used for type checking.	Instances can have arbitrary values for any property. Range and domain constraints can be used for type checking and type inference.
Classes encode much of their meaning and behavior through imperative functions and methods.	Classes make their meaning explicit in terms of OWL statements. No imperative code can be attached.
Classes can encapsulate their members to private access.	All parts of an OWL/RDF file are public and can be linked to fromanywhere else.
Closed world: If there is not enough information to prove a statement true, then it is assumed to be false.	Open world: If there is not enough information to prove a statement true, then it may be true or false.
Role in the Design Process	
Some generic APIs are shared among applications. Few (if any) UML diagrams are shared.	RDF and OWL have been designed from the ground up for the Web. Domain models can be shared online.
Domain models are designed as part of a software architecture.	Domain models are designed to represent knowledge about a domain,and for information integration.
UML, Java, C# etc. are mature technologies supported by many commercial and open-source tools.	The Semantic Web is an emerging technology with some open-source tools and a handful of commercial vendors.
Miscellaneous Features	
Instances are anonymous insofar that they cannot easily be addressed from outside of an executing program.	All named RDF and OWL resources have a unique URI under which they can be referenced.
UML models can be serialized in XMI, which is geared for exchangeamong tools but not really Web-based. Java objects can be serialized intovarious XML-based or native intermediate formats.	RDF and OWL objects have a standard serialization based on XML, with unique URIs for each resource inside the file.

Table 2.2: *"A Comparison of OWL/RDF and Object-Oriented Languages"* (Section 3.3 in Oberle et al. (2006)

2.3.1.1 The Bridge Pattern

The work of Gamma et al. (1995) provides a valuable framework for the description and development of Object-Oriented software design patterns. Many of the principles it outlines have been reused and adapted to the construction of ODPs. A review of the patterns put forward, reveals two *structural* patterns, the *Adapter* and the *Bridge*, together with the notion of *mixin class*, that rely on the use of multiple inheritance.

The *Bridge* pattern is particularly interesting because it addresses a modelling scenario that aligns to the characteristics of the multiple classification criteria described hereto; consisting on the decoupling of an abstraction from its implementation.

Consider a situation where there is a taxonomy of abstract user-interface (UI) elements to represent (an abstract Window, an abstract IconWindow, etc.) and a taxonomy of operating system (OS) platforms to be supported (X-Window System, IBM Presentation Manager, etc.). The implementation of all the UI elements for all OS platforms results in what Rumbaugh et al. (1991) denominated as *nested generalization*.

The *Bridge* solves this nested generalization separating these two classification criteria of a given class (UI element, OS platform) and enabling clients to select an abstract UI element and an implementation in a particular OS platform (Figure 2.4) This technique is referred to by Amborn (2004)(§ 5.2) as a *facet-oriented design* with two *facets*: Abstraction and Implementor.

This conceptualization coincides with the design principles of a Faceted Classification Scheme (FCS) used in Library and Information Science as subsequent sections will discuss and to some extent includes similarities to the Class As Property Value ODP by Noy (2005), where two independent taxonomies in the ontology model are intended to be *bridged* via a property.

2.3.1.2 View Inheritance

Meyer (2000) is regarded as a foundational resource in the design and construction of object-oriented software. The author covers in detail from his extensive experience most, if not all, key aspects to consider when building a software artifact using an object-oriented methodology: classes, objects, inheritance, generic programming (also referred to as the use of templates), design patterns, etc. Meyer attempts to justify what in his opinion constitutes optimal design choices, laying out the principles that support his decissions. One of the topics discussed extensively is the correct use of inheritance and multiple inheritance. His elaborate explanations linking an abstraction in the real world to an object-oriented conceptual model facilitates the reader task to grasp complex design scenarios.

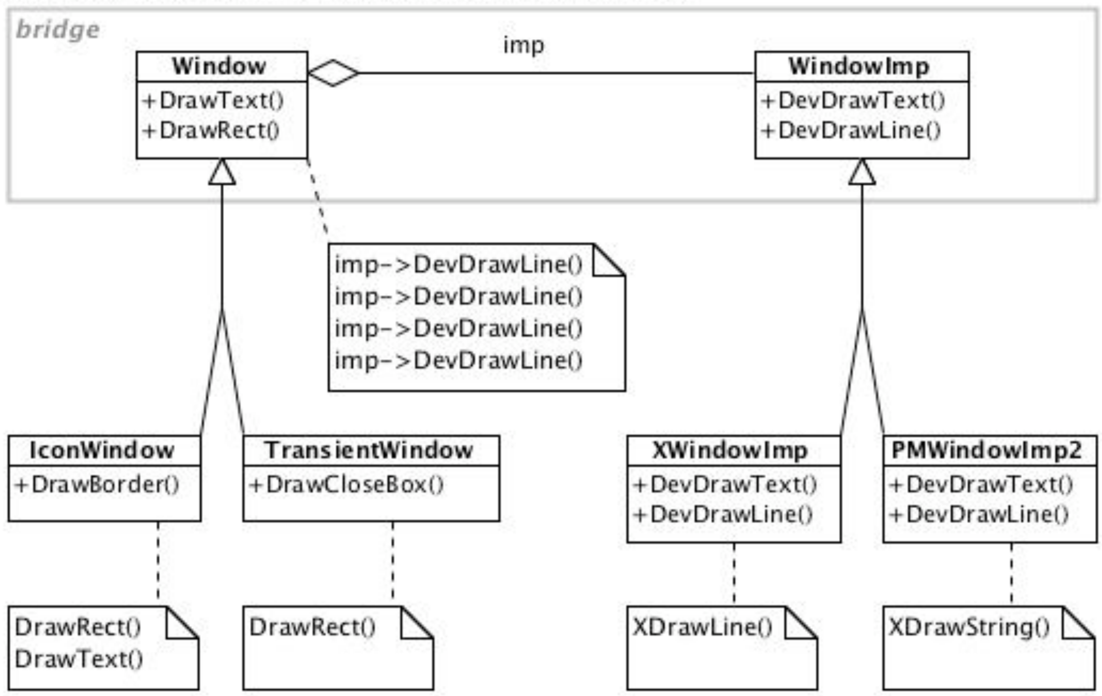


Figure 2.4: Bridge Pattern example (reproduction of Figure in Gamma et al. (1995)(Chapter 4, § Motivation, p. 152).

Meyer characterized 12 different kinds of inheritance in object-oriented design grouped into three categories, that he defines as follows:

Model inheritance, which reflects "is-a" relations between abstractions in the model.

Software inheritance, which expresses relations within the software itself rather than the model.

Variation inheritance, which describes a class by how it differs from another class (a special case that may pertain either to the software of to the model).

Figure 2.5 presents the complete taxonomy, including the 12 kinds of inheritance. Each type was originally defined in Meyer (1996) and detailed even further later in Meyer (2000)(§ 24.5).

Meyer also addresses the modeling of multiple inheritance in the object-oriented world. The whole Chapter 15 of Meyer (2000) is devoted to this topic. He covers various scenarios where multiple inheritance is the natural mechanism to use to achieve the ability to combine several abstraction into one; a situation favored by the *construction-box approach* to software development in the object-oriented paradigm. This is supported from his experiences building general-purpose reusable software libraries.

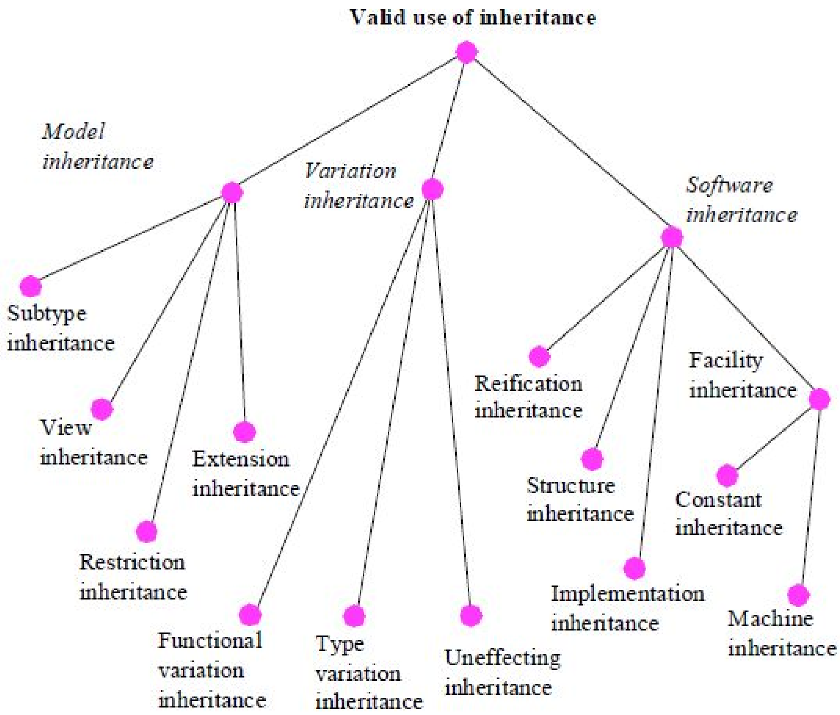


Figure 2.5: "Classification of the valid categories of inheritance" (Figure 1 in Meyer (1996) and Figure in Meyer (2000)(§ 24.5, p. 824).

On the other hand, Meyer is aware of the technical problems that arise for a language that supports multiple inheritance, namely, name clashes due to *repeated inheritance*. Repeated inheritance occurs whenever a class is a descendant of another class in more than one way, which can cause some potential ambiguities. Snyder (1987) labeled such scenario as the *diamond problem*[6], Sakkinen (1989) referred to it as *fork-join inheritance*, or Truyen et al. (2004) as the *common ancestor problem*.

But more importantly, for every drawback that multiple inheritance may entail, Meyer discusses sophisticated mechanisms that an object-oriented language should support as a viable solution. Thus, for every case of a name clash that repeated inheritance may introduce, he puts forward an approach to overcome it based on the *renaming* of the features from the various classes involved in the ambiguity.

From the 12 types of inheritance identified by Meyer, there is one in particular that, as in the case of the Bridge pattern, is relevant to the research questions hereto. This is *View Inheritance* or *classification through views*, as detailed in Meyer (2000)(§24.10 p. 851). The design scenario that View Inheritance addresses is phrased in very similar terms to that of the "Fault" domain concept of ReSIST, such as:

> Perhaps the most difficult problem of using inheritance arises when al-

[6] http://en.wikipedia.org/wiki/Diamond_problem

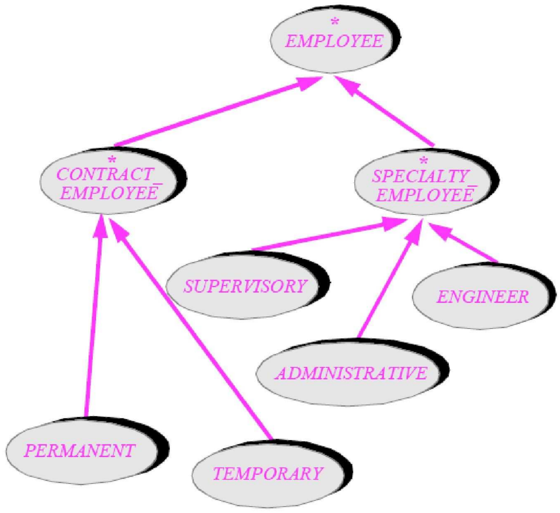


Figure 2.6: “Classification through views” (Figure in Meyer (2000)(§ 24.10, p. 853).

ternative criteria are available to classify the abstractions of a certain application area. Meyer (2000)(§ 24.10, p. 851).

To illustrate this scenario, Meyer uses the example of a class `Employee` in a personnel management system that can be classified based on two different criteria: (a) by contract type, such as permanent versus temporary; and (b) by job type, such as engineering, administrative, managerial.

Similarly to the motivation example of the Bridge pattern, this can be seen again as a case of what Rumbaugh et al. (1991) called *nested generalizations*, even though, Meyer does not refer to it in these terms. The author positions View Inheritance as a solution to represent the possible combinations of classes that may result from the Employee example, leveraging the benefits of multiple inheritance. The class hierarchy of the Employee example is given in Figure 2.6. The separation of the *contract type* and *job type* criteria (symbolized by the classes `Contract_Employee` and `Specialty_Employee` respectively), aligns with the separation of *abstraction* and *implementation* described in the Bridge pattern, or to the mentioned *faceted-oriented design* of Amborn (2004)(§ 5.2) with two facets again, in this case: *contract type* and *job type*.

In Meyer’s opinion, the use of View Inheritance is *not* a beginner’s mechanism and it is appropriate when the following three conditions are met:

- The various classification criteria are equally important, so any choice of a primary one would be arbitrary.
- Many possible combinations are needed.

- Reusability. The classes under consideration are so important as to justify spending significant time to get the best possible inheritance structure. This applies in particular when the classes are part of a reusable library with large reuse potential.

Interestingly enough, Meyer acknowledges an alternative design to View Inheritance very similar to what he calls the *handle-based design pattern*, which in turn aligns to the generic structure proposed by Gamma et al. (1995) in the Bridge pattern.

2.3.2 Conclusions

This overview to the design, use and justification of multiple inheritance in the area of Object-Oriented Design, reveals that object-oriented technologies are not foreign to the abstract modelling of multiple classification criteria.

The notion of *nested generalization* characterized by Rumbaugh et al. (1991) is closely related to the existence of multiple classification criteria.

Gamma et al. (1995) puts forward the Bridge pattern and Meyer (2000) puts forward the technique of View Inheritance to tackle the drawbacks of nested generalization modelling scenarios. In fact, Meyer acknowledges an alternative to View Inheritance, the handle-based design pattern, that aligns to the main implementation proposed by the Bridge pattern.

Both, the Bridge pattern and View Inheritance can be seen as a faceted-oriented design with various facets, one for each classification criterion involved in the nested generalization scenario in the terms described by Amborn (2004)(§ 5.2).

The tree representation of the "Fault" domain concept in Figure 1.3, reveals the *explosive* combination of likely types of faults that the nested generalization of the 16 elementary fault classes may cause.

The representation of the "Fault" domain concept in the D&S ontology of ReSIST can benefit from the good practices to handle nested generalizations captured in the Bridge pattern and in View Inheritance, provided that these can be adapted to standard W3C semantic technologies suitable for the ReSIST project and thus, for deployment in the Semantic Web.

2.4 Faceted Classification

So far, we have seen existing techniques and patterns in both, Ontology Engineering and Object-Oriented Design, to deal with modelling scenarios that involve abstractions of concepts subject to multiple classification criteria and multiple inheritance.

Ontology Engineering puts forward the Normalisation ODP to untangle the poly-hierarchies that may exist in ontology models due to the existence of a high number of multiple inheritance relations among the concepts involved by identifying "semantic axes" to decouple such relations.

Object-Oriented Design puts forward the Bridge Pattern and View Inheritance to handle the modeling of nested generalizations and decouple the abstraction/implementation aspects of a concept or other available classification criteria that needs to be represented.

However, there is an important element in the application of these techniques that is acknowledged but not fully discussed, which has to do with questions such as: *what constitutes a semantic axis?* Or *why do nested generalizations occur in the first place?* And when building an ontology from scratch, *how do you identify that the concepts to represent are subject to multiple semantic axes or nested generalizations?* Facet Classification and Facet Analysis from Library and Information Science, can help with that part of the equation and assist ontologists to find answers to these questions.

The Normalization ODP, the Bridge Pattern and View Inheritance focus more on the implementation side in a post-conceptualization or post-modelling phase, while Faceted Classification and Facet Analysis can assist in the conceptualization or modelling phase to identify and characterize the *reasons* why the application of the former techniques is appropriate.

Not surprisingly, the origins of Faceted Classification and Facet Analysis are rooted to the works of Ranganathan (1933, 1960, 1967) as a consequence of his disappointment with traditional enumerative bibliographic classification systems to support subjects that could be classified according to multiple views, topics, attributes or criteria (*compound subjects*). As a result, S. R. Ranganathan released his Colon Classification in 1933, which is regarded as the first *universal* (or general) FCS. Colon Classification is viewed as a universal scheme as it is intended to cover the whole body of human knowledge.

2.4.1 What Is a Facet?

The available literature concerning Faceted Classification and Facet Analysis is very vast dating from Ranganathan's initial works. Notable reviews of the field, have subsequently followed such as Vickery (1960) and Broughton (2004). More recently, La Barre (2006) also provided an excellent recapitulation of the use and evolution of Faceted Classification and Facet Analysis over the years. More importantly, it studied the topic in the context of the design and construction of websites in the *traditional* World Wide Web.

The core element of a Faceted Classification is the notion of *facet*. The definition of facet has evolved over the years, since its introduction by Ranganathan (1933, 1967).

Ranganathan introduced the concept of facet, as part of his methodology to conduct facet classification, which included a series of 46 canons, 13 postulates, and 22 principles.

One of such canons, "The Canon of Differentiation", as noted in Spiteri (1998) states that "when dividing an entity into its component parts, it is important to use characteristics of division (i.e., facets) that will distinguish clearly among these component parts". As an example, the entity "human beings" and the characteristic of division "gender" are used, which will produce 2 distinctive components parts. Thus, Ranganathan introduces the notion of facet of a given entity as a characteristic of division of that entity.

A very similar definition emerged from the Classification Research Group (CRG) in the UK. The CRG expanded on the works by Ranganathan and developed a similar methodology to create and implement a faceted classification. Spiteri (1998) aggregated the principles of faceted classification published by the CRG across multiple sources in a wide span of years [Broughton (2011)]. One of such principles is the "Principle of Division", which states that "a facet must represent *only one characteristic of division* of the parent universe". A definition very much in line with that in "The Canon of Differentiation" by Ranganathan and reiterated by two renowned members of the CRG: Broughton (2006) (p. 53, 59); and Vickery (2008) (§ 4, p. 148; § 5, p. 150).

From Spiteri (1998), it follows that both Ranganathan and the CRG, laid out additional characteristics that a facet should meet, such as being *mutually exclusive* (with respect to the other facets in the domain of discourse, and *permanent*. Mutual exclusivity in this sense, should follow naturally provided that each facet actually aligns to the main principle of representing one single characteristic of division. The characteristic of permanence on the other hand, refers to facets reflecting permanent qualities of the target entity they aim to classify.

This notion of permanence is particularly interesting from an ontology modelling point of view, as it evokes the notion of *essence* in the OntoClean methodology by Guarino and Welty (2009). OntoClean was developed precisely to assist ontology practitioners on evaluating the correctness of ontological relations, and as per the methodology, "a property of an entity is essential to that entity if it must be true of it in every possible situation". As Spiteri (1998) shows, Ranganathan illustrates the notion of permanence of a facet using *breed* as a characteristic of division of a "Dog", given that a dog can not cease of being of one particular breed for as long as it is a dog. At the same time, Guarino and Welty (2009) uses *being human* as an essential property of a "Person" for the same reason. A person can not cease of being a human for as long as it exists. Such alignments between facet analysis and ontology modelling, reinforces one of the main ideas that are part of this research, namely that facet analysis and faceted classification can play an important role in Ontology Engineering.

Kwasnik (1999) (p. 39) provided another relevant aspect regarding the concept of facet. The author noted that "the notion of facets rests on the belief that there is more than one way to view the world." La Barre (2006) (§ 2.1, p. 47) also alludes to this aspect of facets as a mean to support multiple viewpoints when refers to them as "the ability to analyze an entity in a way that enables one to view it from every conceivable angle". This perspective of facet as a mean to support multiple *viewpoints*, is worth highlighting because it resonates with the terminology used by Avizienis et al. (2004) to define the concept of "Fault" and its associated taxonomy. The taxonomy of "Fault" is presented based on 8 basic *viewpoints* from which different fault classes can be characterized (Figure 1.1).

The idea of multiple *viewpoints* (as facets) seems to apply as well to the other examples of domain concepts previously mentioned and found in the literature. For instance, the type of grape, color, region of origin for "Wine" in Welty et al. (2004) or the type of topping, or base for "Pizza" in Horridge et al. (2009), appear as valid alternative viewpoints to consider in the respective ontological representation of those concepts. Although in these cases, the authors do not refer to the notion of multiple viewpoints explicitly when approaching the conceptualization of "Wine" and "Pizza", as Avizienis et al. (2004) do when describing the concept of "Fault".

This definition of facet, as a *principle of division* or *multiple viewpoint* of a particular domain, suggests a significant similarity to: (a) the notion of *semantic axis* in the Normalisation ODP (Sections 2.2 and 5); and (b) the idea of *classification through views* to address *nested generalizations* that is put forward by View Inheritance in Object-Oriented Design (Section 2.3). Gnoli (2008) already notes how the notion of facet and Facet Analysis has proved useful in other disciplines outside LIS, ranging from Philosophy, Psychology, Linguistics, Musicology to specially Computer Science (i.e. Graphical User Interfaces or Information Systems). In Computer Science for example, simplified models of faceted approaches have been commonplace across graphical user interfaces of web-based applications and services for several years until now [La Barre (2006); Uddin and Janecek (2007); Vickery (2008)]. As Gnoli concludes, facets can be seen as "a natural way or analyzing and organizing any kind of concepts".

In that sense, this research also aims to explore how Facet Analysis and Faceted Classification can be useful outside the LIS field. More specifically, the idea is to leverage on the conceptual alignments between these key elements (facet, semantic axis, nested generalization) within each domain modelling paradigm respectively (Faceted Classification, Ontology Design, Object-Oriented Design) to reveal any evidence on how they can aid our ultimate goal: identifying ontology design patterns to model multiple classification criteria of domain concepts in the Semantic Web.

2.4.2 Examples of Faceted Classification Schemes

The aforementioned Colon Classification by Ranganathan (1933, 1960, 1967) is regarded as the leading exponent of a universal (general) FCS. S.R Ranganathan, devised 5 main facets for the classification system, labeled as: Personality, Matter, Energy, Space and Time (PMEST). Each one of the facets represents a main principle of division of the target subject under classification. Personality refers to the core topic studied by the subject. Matter denotes the notion of substance that might be involved in the target item. Energy stands for processes, activities or actions, and finally the omnipresent characteristics of Time and Space.

In addition to Colon Classification, other notable example of universal FCSs include the second Bliss Classification (BC2) developed by the Classification Research Group (CRG) in the UK, and released by Mills and Broughton (1977). The BC2 expands the 5 initial facets of the Colon Classification into 13, namely: thing, kind, part, property, material, process, operation, agent, patient, product, by-product, space and time. It was the view of the BC2 editors that the original five facets introduced by Ranganathan did not suffice to accommodate all aspects of knowledge, specially in certain disciplines.

Lastly, it is also worth noting the Universal Decimal Classification (UDC) originally put forward by Otlet and Fontaine (1905). The UDC was derived from the Dewey Decimal Classification (DDC), and was not originally conceived as a faceted classification. Nonetheless, UDC incorporated a synthetic nature from the beginning that gave the scheme a sense of faceted-like features. The system provided a series of operators that allowed to combine multiple class descriptors for a target subject. Moreover, UDC has undergone plenty of subsequent revisions, such as McIlwaine and Williamson (1994), that have stressed the features and functionality of a proper faceted classification approach.

An important characteristic of these three examples, Colon Classification, BC2 and UDC, is that they are designed as universal (or general) FCSs. Yet, not all FCSs have to be universal. As remarked by Denton (2003), an FCS can be used to classify all human knowledge or simply the clothes in your wardrobe. Broughton (2006) also refers to more specific FCS for a particular subject or domain like for example "Sock". Wild et al. (2009) refers to "simplistic domains as exemplars" as a way to illustrate FCSs using simple concepts along the line of "Wine" or "confectionary ingredients", which is the case in Wilson (2006)[7], or "Dishwasher Detergent" as in Denton (2003).

Drawing an analogy between Library Information Science and Ontology Engineering, universal (faceted) classification schemes in LIS could be seen at a similar level of granularity as upper (or top-level) ontologies in Ontology Engineering [Gomez-Perez et al. (2004)]. Upper ontologies aim to represent concepts at the highest level of abstraction possible so that they would apply to any conceivable domain. By contrast, a FCS for

[7] http://www.facetmap.com/

a simple domain could be seen at a similar level of granularity as a domain-specific ontology, aimed at represent only the concepts relevant to the target domain.

This distinction between universal or domain-specific is key, given that the scope of this thesis is not set on universal representation schemes. This thesis is set on the representation of specific domain concepts according to multiple classification criteria. As such, domain-specific FCSs are favored throughout this review of the LIS field.

2.4.3 Faceted Classification and Ontology

Kwasnik (1999) (p. 40-42), in her overview of the role of classification in Knowledge Representation, performs a comparison of the strengths and weaknesses of four different classification schemes: hierarchical, trees, paradigms and finally, faceted classification. The author lists several features in favour of faceted classification, such as: (a) they do not require complete knowledge of the entities or their relationships; (b) they are hospitable (can accommodate new entities easily); (c) they are flexible; (d) they are expressive; (e) they can be ad-hoc and free-form; and (f) they allow many different perspectives on and approaches to the elements to be classified. Conversely, she acknowledges three major disadvantages: (a) the difficulty of choosing the right facets; (b) the lack of the ability to express the relationships between them; and (c) the difficulty of visualizing it all.

In fact, Faceted Classification has been noticed in the past in the Ontology Engineering community. John Sowa, as part of one of his regular involvement in the Ontolog Forum[8] (an open virtual community of practice devoted to advancing the field of Ontology Engineering), praises the virtues of Ranganathan's facet-oriented Colon Classification and states:

> Ranganathan's colon classification system is of fundamental importance. I would strongly urge everybody with any interest in ontology to get a basic acquaintance with the system[9].

Sowa justifies his statement in the same post because in his opinion: "Single inheritance is hopelessly inadequate for classification". To emphasize this idea, he closes his intervention as follows:

> Summary: Every single-inheritance ontology is obsolete. Any single-inheritance system that is currently in use should be replaced or updated to a multiple-inheritance system in order to make it suitable for further development and extension[9].

[8] http://ontolog.cim3.net/

[9] http://ontolog.cim3.net/forum/ontolog-forum/2008-06/msg00028.html (John F. Sowa, 2008/06/22)

There has been previous efforts to bridge the modelling of faceted classification systems into ontologies. One such example is Tzitzikas et al. (2006). To the features of Faceted Classification outlined by Kwasnik (1999), Tzitzikas et al. (2006) add three additional advantages from a computational point of view: conceptual clarity, compactness (they takes less space) and scalability (easier to update and maintain). In addition, Tzitzikas proposes two extensions to a faceted ontology to infer valid and invalid combination of facet terms that can assist with the indexing and navigation of the system employing his approach.

Going beyond Faceted Classification, there has been previous approaches to convert other types of classification schemes (non-faceted per se), into RDF-S and OWL ontologies. Hepp and de Bruijn (2007) presents GenTax, a methodology for deriving ontologies from schemata such as hierarchical classifications, thesauri or inconsistent taxonomies. However, such methodology is out of the main scope of the research hereto, which is bound to the proposal of similar procedures for the transformation of domain-specific faceted classification schemes in line with the representation of multiple classification criteria.

Even in broader terms, the research of Garcia-Silva et al. (2008) presents a pattern based approach to derive a light-weight ontology model for various types of Non-Ontological Resources (NORs) beyond classification schemes. These include glossaries, lexicons, thesauri and even folksonomies. The approach consists of a framework, in which one of the steps deals with the transformation of a specific NOR into an ontology model based on a pre-existing catalog of re-engineering patterns. As of now, the catalog does not include yet a pattern to re-engineer faceted classification schemes per se, although if such a pattern was to be developed, it can be added to the catalogue and used in the overall framework. The findings that will be revealed in the chapters ahead can contribute to this catalogue of re-engineering patterns for the faceted classification scheme NOR.

The two sections that follow, explore two previous existing methods respectively, that derive an ontology model from a faceted classification scheme specifically. The first approach is based on the Resource Space Model (RSM) by Zhuge et al. (2008). The second, is based on the universal faceted classification schemata proposed by Bhattacharyya (1979) and used by Giunchiglia et al. (2009) as background knowledge to build a lightweight ontology model. The two approaches are compared to the methodology proposed in Chapter 6 as part of the results of this research.

2.4.4 Resource Space Model (RSM)

Previous work that defines mappings between different semantic models include Zhuge et al. (2008). The authors perform a rigorous and comprehensive comparative analysis between the primitive elements of three semantic models: the Semantic Web Ontology

Language (OWL), the Relational Database Model (RDBM), and the Resource Space Model (RSM). Based on the identified mappings between every two models, a detailed set of criteria is provided to transform one of them into the other. The most relevant to us is the mapping between RSM and OWL because of its similarities with the conversion between a FCS and OWL that we propose here.

The RSM is defined as a semantic model for specifying, organizing and retrieving diverse multimedia resources by classifying their contents according to different partition methods and organizing them according to a multidimensional classification space. A FCS is also a multidimensional classification space and comparing the primitive elements of a FCS and a RSM the following mapping is instantly revealed:

- The domain or universe of discourse of the FCS (the target domain concept) corresponds to the overall resource space, the RS element in the RSM.
- A facet in the FCS corresponds to an axis X_i in the RSM.
- A facet term in the FCS corresponds to a coordinate C_i in the RSM.
- A facet is covered and exhausted by the set of terms associated to it in a FCS. The same principle holds in a RSM for an axis and the set of coordinates associated to it, $X_i = \langle C_{i1}, C_{i2}, ..., C_{in} \rangle$.
- An item to be classified by the FCS corresponds to a point p in the RSM.

These mappings show that a generic FCS can be converted into a RSM, which in turn can be converted into an OWL model using the RSM to OWL mappings in Zhuge et al. (2008). Now there are two possible paths to convert a FCS into an OWL model.

- Path 1: FCS to RSM via the mappings above and RSM to OWL via the mappings in Zhuge et al. (2008). Let us refer to this OWL model as O_1.
- Path 2: FCS to OWL via the mappings presented in Chapter 6 and Rodriguez-Castro et al. (2010b) using the Normalization ODP. Let us refer to this OWL model as O_2.

There are important differences between the ontologies O_1 and O_2. An important difference is due to the RSM to OWL conversion in Zhuge et al. (2008). RSM describes mainly classification semantics and as the authors explain, this means that there is no semantic loss when converting from RSM to OWL but there might be semantic loss when transforming an OWL model that includes richer semantics into a RSM. This also means that, in terms of W3C standards, the expressivity level of the resultant OWL model O_1, will be within the RDF Schema or OWL Lite boundary.

On the other hand, the ontology O_2 is within OWL DL and presents richer OWL semantics than O_1, provided by the Normalization ODP. These additional OWL DL semantics in O_2 enable one of the main features of the normalization pattern such as the automatic classification and maintenance of complex subsumption relations by a reasoner. So while O_1 is a valid OWL description of the FCS that it is based on, O_2 using the proposed method in Chapter 6 provides additional semantics at the OWL DL level that support a richer description and additional features of the classification criteria considered in the initial FCS.

2.4.5 Faceted Lightweight Classification Ontology

Previous work that made use of facet analysis in Library and Information Science to build computational ontologies includes Giunchiglia et al. (2009). Giunchiglia et al. introduces the concept of Faceted Lightweight Classification Ontology as "a lightweight (classification) ontology where each term and corresponding concept occurring in its node labels must correspond to a term and corresponding concept in the background knowledge, modeled as a faceted classification scheme".

Similarities to the approach presented in Chapter 6 and Rodriguez-Castro et al. (2010b) include:

- The use of a FCS to model certain background knowledge and to derive an ontology based on it.
- Each concept in the ontology model obtained using the method in Chapter 6 also corresponds to a concept in the source FCS.

There are important differences where the approach in Giunchiglia et al. (2009) deviates from that in Chapter 6 and Rodriguez-Castro et al. (2010b) probably due to the different type of problems that both are trying to address respectively. Giunchiglia et al. are trying to counteract the lack of interest and difficulties on the user side to build and reuse ontologies while the concern in the latter, focuses on identifying explicit guidelines to represent the notion of multiple classification criteria in domain concepts. Additional differences include:

- The expressive level for the resultant ontology model in the method presented in Chapter 6 hereto is OWL DL. In contrast, Giunchiglia et al. (2009) focuses on lightweight classification ontologies which expressive level would loosely correspond to no more than RDF Schema in terms of W3C Standards. Key features provided by the Normalization ODP found in the former method, can not be implemented using solely RDF Schema semantics.

- The type of FCS used in Giunchiglia et al. (2009) is based on the universal faceted classification system by Bhattacharyya (1979). On the other hand, the approach in Chapter 6 focuses on simpler custom domain-specific FCSs to serve as a starting point for the initial proof of concept. This helped limiting the complexity of the classification criteria to consider and represent in the corresponding ontology.

2.4.6 Conclusions

It is important to note, however, that the focus of this research is not set on universal faceted classification schemes such as that of Ranganathan of the Second edition of the Bliss Bibliographic Classification because their granurality is too coarse and beyond the scope of the type of domain concepts that are being considered: “Fault”, “Pizza”, “Wine”, “Dishwashing Detergent”, “Sock”, etc. Such approach could be seen analogous to attempting the representation of these type of very domain-specific concepts using an existing upper ontology, which by definition are domain agnostic.

Instead, the focus of this research is narrowed down to domain specific Faceted Classification Schemes (FCSs) such as that developed by Spiteri (1998) and illustrated with an example in the domain of “Dishwasher Detergents” by Denton (2003). Spitieri put forward a *“Simplified Model for Facet Analysis”*, that is based on the same principles that led Ranganathan and the Classification Research Group to develop their universal schemes, yet they are presented in a more accessible format.

Based on the presented review of Faceted Analysis and Faceted Classification together with its applicability to disciplines outside of the LIS field, allows one to think that Ontology Engineering can not afford not to include this powerful knowledge organization technique as an important component of its tool-set.

More specifically, this research aims to demonstrate how Facet Analysis and Faceted Classification Schemes, can assist the ontology creation process when attempting to represent domain concepts prone to be viewed according to multiple classification criteria.

Chapter 3

Revisiting the Class As Property Value ODP

This chapter presents a revision to the Class As Property Value ODP introduced by Noy (2005) and further revisited in Presutti et al. (2008)(§ 2.2.1).

The revision here concentrates on one of the specific approaches of the pattern discussed in Noy (2005). An abstract general structure of the ontology schema behind this approach is proposed and a generic implementation in OWL of the elements in that structure is also provided. This generic structure allows one to expand the pattern to decouple the notion of *interpretation* and *terminology* that some elements of the pattern perform in various scenarios.

This revision of the Class As Property Value ODP will be also used to illustrate the alignments among this approach of the pattern, the Value Partition ODP and the Normalisation ODP. These alignments are revealed by a comparative analysis of the generic ontology schema behind these patterns, that is key to the contributions of this research and to be discussed in subsequent chapters.

3.1 Visual Notation to Characterize ODPs

Prior to the revision of the Class As Property Value ODP, this section presents a simple notation that will be used to characterize the various ODPs and ontology models that will be examined hereafter. The motivation to introduce this notation is driven by simplicity, a light-weight footprint while being fit-for-purpose; and evolved from the need of *illustrating* examples of ODPs in a plain text format that facilitated collaboration via in-line comments and peer-reviews in open virtual settings such as specialized mailing lists [1], [2].

Its main features can be summarized as follows:

- To convey *at-a-glance* the key conceptual elements that participate in the ODP.
- To convey likewise the structure of how these elements are arranged in the ODP (the subsumption hierarchy).
- The characterization of the ODP in general terms (so that it can be populated with an indefinite number of specific examples).
- The use of a plain-text based representation that: (a) provides a visually richer representation than the corresponding RDF/XML or N3 version; and (b) enables side-by-side visual comparison of the conceptual elements of several ODPs simultaneously.
- To capture only the subset of the W3C OWL specification needed to address the specific design issue under discussion.
- To serve as an additional resource to ease collaboration among a wider audience of ontology practitioners that may be familiar with the W3C OWL specification but not necessarily with the underlying Description Logic formal theory in Baader et al. (2003) (for example, in the field of Faceted Classification or Knowledge Organization Systems in Library and Information Science).

At the moment, this notation is not aimed at supporting the full range of OWL constructs or a proposal to become a standard in the ontology development community. Such goals are beyond the scope of this work. Yet, I have found it to be an effective tool to *illustrate* design issues in ODPs in virtual forums, which led me to use it previously in Rodriguez-Castro et al. (2010b).

The rest of this section describes the symbols and the elementary building blocks that compose the notation to characterize ODPs, according to how they will appear in the various figures going forward to illustrate different design aspects of the patterns involved.

A brief inventory of the symbols introduced is presented below:

- The symbol "`|--`" denotes one of the relations: rdfs:subClassOf, rdfs:subPropertyOf or rdf:type; based on the elements involved.
- The symbol "`(≡)`" denotes that the adjacent element is a *defined* owl:Class.

[1] http://ontolog.cim3.net/forum/ontolog-forum/2010-04/msg00051.html
[2] http://lists.w3.org/Archives/Public/public-owl-dev/2010AprJun/0009.html

- The symbol "`(◊)`" denotes that the adjacent element is an owl:NamedIndividual.
- The expression "`[ :`*Element* `| (◊) :`*Element* `]`" denotes that :Element can be either an owl:Class or an owl:NamedIndividual.
- The symbol "`(I)`" denotes that the adjacent element is an owl:Class, but it is *implicit* or *hypothetical*, not implemented explicitly and not part of the asserted ontology model.
- The symbol "`(≡)(P)`" denotes that the adjacent element is a *defined* owl:Class and is implemented as a *value partition*.
- Expressions between parenthesis "`(...some text ...)`" denotes some comment, annotation or relevant description in natural language.

What follows, is a detailed description of the elementary building blocks that employee these symbols and their ontological meaning.

Class or Property Subsumption

Listing 3.1 depicts that :Element is an owl:Class subsumed by owl:Thing.

```
owl:Thing
    |-- :Element
```

Listing 3.1: :Element is an owl:Class

Listing 3.2 depicts that :Element1 is an owl:ObjectProperty subsumed by owl:topObjectProperty. (The construct would be analogous in the case of an owl:DatatypeProperty).

```
owl:topObjectProperty
    |-- :Element
```

Listing 3.2: :Element is an owl:ObjectProperty

More generally, Listing 3.3 depicts that :Element1 subsumes :Element2, where:

- If :Element1 is an owl:Class, then :Element2 is an owl:Class and the subsumption relation represented is rdfs:subClassOf.
- If :Element1 is an owl:ObjectProperty (or owl:DatatypeProperty), then :Element2 is an owl:ObjectProperty (or owl:DatatypeProperty) and the subsumption relation represented is rdfs:subPropertyOf.

Therefore, in these two cases, the symbol "`|--`" denotes either the relation rdfs:subClassOf or rdfs:subPropertyOf based on the elements involved.

```
:Element1
    |-- :Element2
```

Listing 3.3: :Element1 subsumes :Element2

Individual Instantiation

Listing 3.4 depicts the instantiation relation rdf:type, where :Element2 is an owl:Individual of the owl:Class :Element1.

In these case, the symbol "|--" denotes the relation rdf:type because of the elements involved.

```
:Element1
    |-- (◊) :Element2
```

Listing 3.4: :Element2 is an owl:NamedIndividual

Defined or Primitive Class

Listing 3.5 depicts that :Element2 is a *defined* owl:Class subsumed by the owl:Class :Element1.

```
:Element1
    |-- (≡) :Element2
```

Listing 3.5: :Element2 is a *defined* owl:Class

The concept of *defined* class, and by contrast *primitive* class, are detailed in Rector (2003); Rector et al. (2004); Horridge et al. (2009).

A *defined* owl:Class refers to a class that makes explicit the properties that suffice to infer whether a given owl:NamedIndividual is a member of the class. In practical terms, a *defined* owl:Class represents a bidirectional implication and participates in *at least one* owl:equivalentClass relation in the ontology model. Being a member of the class implies exhibiting certain properties and exhibiting certain properties implies being a member of the class.

On the other hand, a *primitive* owl:Class refers to a class that makes explicit the properties that a given owl:NamedIndividual asserted to be a member of the class, will exhibit. In that sense, a *primitive* class represents a one-way implication and it does *not* participate in any owl:equivalentClass relation in the ontology model. Being a member of the class implies exhibiting certain properties but exhibiting certain properties does *not* imply being a member of the class.

Because of the information that they provide in order to infer class membership, primitive classes are also known as *partial* classes and defined classes are also known as *complete* classes.

All owl:Class elements that appear in the notation being specified here are *primitive* classes by default, unless they are annotated with the *defined* class symbol "(≡)".

Class Subsumption or Individual Instantiation

Listing 3.6 depicts that :Element2 can be *either* an owl:Class *or* an owl:Individual, where:

- In case of an owl:Class, :Element2 is subsumed by the owl:Class :Element1.
- In case of an owl:Individual, :Element2 is an instance of the owl:Class :Element1.

```
:Element1
    |-- [ :Element2 | (◊) :Element2 ]
```

Listing 3.6: :Element2 is *either* an owl:Class *or* an owl:NamedIndividual

Implicit Class Not Asserted

Listing 3.7 uses the symbol "(I)" to depict that :*Element* is an owl:Class but *implicit* or *hypothetical* and not implemented explicitly in the asserted ontology model.

```
owl:Thing
    |-- (I) :Element
```

Listing 3.7: :*Element* is an *implicit* owl:Class (not asserted in the ontology model)

Partition

Listing 3.8 uses the symbol "(≡)(P)" to depict that :Element2 is a *defined* owl:Class and is implemented as a *value partition*.

```
:Element1
    |-- (≡)(P) :Element2
        |-- [ :Element_i | (◊) :Element_i ]
```

Listing 3.8: :Element2 is an owl:Class implemented as a *value partition*

The concept of *value partition* is detailed in full in Chapter 4, based on the work from Rector (2005).

In brief, the term *value partition* is derived from the notion of *partition of a set* in mathematics [3, 4], where a given set is divided into a non empty finite number of subsets that cover (also referred to as exhaust) the given set. That is: (a) all subsets are mutually exclusive to each other (also referred to as pairwise disjoint); and (b) the union of all subsets is equivalent to the given set being partitioned.

In terms of the notation in Listing 3.8:

- "`(≡)(P)` :Element2" is an owl:Class that represents the element being partitioned.
- "`(≡)(P)` :Element2" is also a *defined* owl:Class because it is equivalent to the union of all elements :$Element_i$.
- :$Element_i$ is either an owl:Class or an owl:NamedIndividual that represents each one the elements that partitions the :Element2 owl:Class. (As Chapter 4 will present following Rector (2005), OWL allows to implement a value partition using a series of either owl:Class or owl:NamedIndividual elements).
- All elements :$Element_i$ are pairwise disjoint.
- The union of all elements :$Element_i$ is equivalent to the :Element2 owl:Class.

Note that Listing 3.8 reuses and combines the notation in Listing 3.5 and Listing 3.6 to depict a value partition. From the definition of value partition, it follows that the partitioned class is also a defined class. In that sense, the use of the combined symbol "`(≡)(P)`" could be seen as redundant, given that symbol "`(P)`" alone, implies "`(≡)`". Nonetheless, for clarity, both would be displayed to depict a value partition class.

In summary, these are all the symbols and building blocks that conform the notation to be used hereafter to characterize, illustrate and compare the various ODPs in the scope of this research.

3.2 Class As Property Value ODP

Noy (2005) presents five different approaches on how a hierarchy of classes can be used as the value of a property. The modelling scenario occurs when a hierarchy of classes is to be reused as a terminology to annotate individual elements of other domain concepts in the ontology. To illustrate this scenario, an example of an existing hierarchy of classes in the domain of "animals" is used as a subject index to annotate the topic of a collection of specific book instances. The author provides a discussion of the implications of each

[3] http://en.wikipedia.org/wiki/Partition_of_a_set
[4] http://encyclopedia2.thefreedictionary.com/Partition

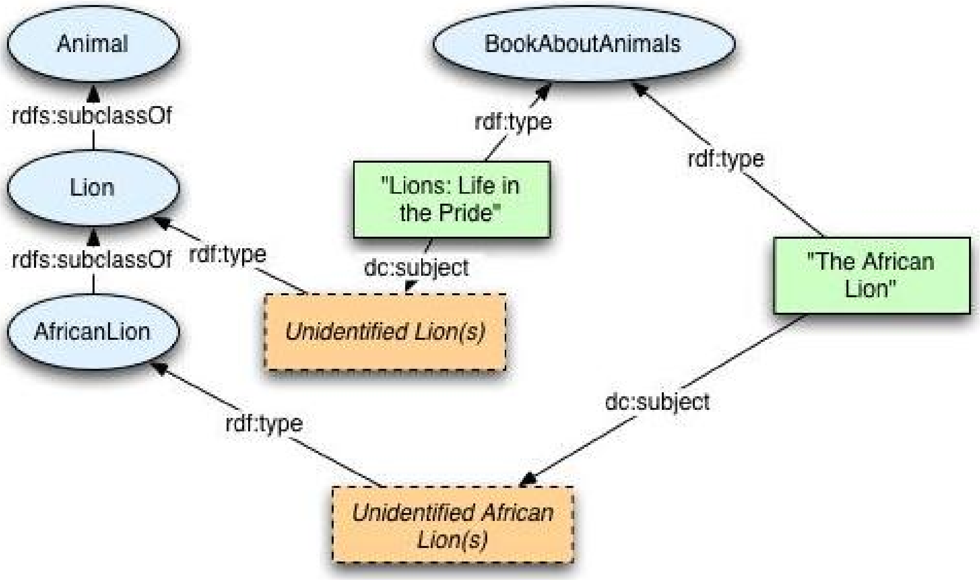


Figure 3.1: *"Using members of a class as values for properties"* (Fig. 4 in Noy (2005)).

approach for the resulting ontology model in the context of RDF-S and the various semantic expressivity levels of OWL (Lite, DL or Full).

The revision of the pattern throughout this chapter focuses on the fourth approach of Noy (2005), entitled "Approach 4: Create a special restriction in lieu of using a specific value". Figure 3.1 depicts a partial ontology model illustrating the scenario in question. An existing subsumption hierarchy formed by the classes :Animal, :Lion and :AfricanLion is reused as values of the property dc:subject to annotate the subject of a collection of individual books ("Lions: Life in the Pride" and "The African Lion"), that are members of a separate subsumption class hierarchy of the ontology formed by :Books, :BookAboutAnimals, etc.

There are several aspects of Approach 4 by Noy (2005) based on the proposed implementation of the classes in the hierarchy subsumed by :Books and their instances that is particularly important when compared to the Value Partition and Normalisation ODP, namely:

- The expressivity of the ontology model in the pattern is within OWL DL.
- The use of anonymous individuals in the :Animal class hierarchy as the value of the property dc:subject, for the instances of the class :BooksAboutAnimals. Figure 3.1 depicts these anonymous individuals as "Unidentified Lion(s)" and "Unidentified African Lion(s)".
- The use implicitly, of another semantic *interpretation* in the real world of the reused class hierarchy subsumed by :Animal. In theory, any instance of :Animal

```
owl:Thing
    |-- (I) :Interpretation_a (Implicit. Not Asserted)
        |-- :Terminology_i
            |-- :T_iClass_j
                |-- (... rest of potential subclasses)
    |-- :TargetDomainConcept (or :TDC)
        |-- (≡) :I_aTerminology_iTDC
        |-- (≡) :I_aT_iClass_jTDC
        |-- (... rest of defined subclasses of :TDC based on
                 an existing interpretation _:Interpretation_a^I
                 and all existing subclasses of :Terminology_i)
        |-- [:SpecificTDC_x | (◊) :SpecificTDC_x]

owl:topObjectProperty
    |-- :hasInterpretation_a
```

Figure 3.2: Generic structure of "Approach 4" in Noy (2005).

represents an *actual* animal in the real world but in the case of the pattern, it could also stand for a generic animal interpreted as a *book subject*.

The rest of this section abstracts the ontology schema behind Approach 4 of Noy (2005) providing a generic structure that accommodates all the elements that participate in the pattern and a generic implementation that preserves all the characteristics of the approach.

3.2.1 Structure and Elements

Figure 3.2 illustrates the generalized structure of the abstract ontology schema behind Approach 4 of the Class As Property Value ODP by Noy (2005) using the notation introduced in Section 3.1. The structure includes a generic representation of all the elements that conform the pattern, maintaining the same functionality and semantic expressivity of Noy's example. This generic structure will be used from hereon to anchor the discussion of the pattern.

The generic structure and elements used in Figure 3.2 are explained below. Additionally, Figure 3.3 applies the proposed generic structure to the example in Approach 4 of Noy (2005), mapping the elements from the example in Figure 3.1 to their corresponding counterpart of the proposed generic structure in Figure 3.2. The ontology model used in Figure 3.3 for the example in Noy (2005) is provided by the author and available online (Approach 4[5]). Therefore:

- $:Terminology_i$ denotes the top class of the terminology, the vocabulary of the hierarchy of concepts that will be used as property values to annotate other elements

[5] http://www.w3.org/TR/swbp-classes-as-values/books4.owl

```
owl:Thing                                              (ODP Generic Structure)
   |-- (I) :Subject (Implicit. Not asserted)           (:Interpretation1)
      |-- :Animal                                         (:Terminology1)
         |-- :Lion                                           (:T1Class1)
            |-- :AfricanLion                                    (:T1Class1Class1)
   |-- :Book                                           (:TargetDomainConcept)
      |-- (≡) :BookAboutAnimals                           (:I1Terminology1TDC)
      |-- (≡) :BookAboutLions                             (:I1T1Class1TDC)
      |-- (≡) :BookAboutAfricanLions                      (:I1T1Class1Class1TDC)
      |-- (◊) :TheAfricanLionBook                         (:SpecificTDC1)
      |-- (◊) :LionsLifeInThePrideBook                    (:SpecificTDC2)

owl:topObjectProperty
   |-- dc:subject                                      (:hasInterpretation1)

owl:topDataProperty
   |-- :bookTitle
```

Figure 3.3: Placement of the elements in the Approach 4 example of the Class As Property Value ODP by Noy (2005), in relation to the generic structure in Figure 3.2.

in the ontology model.

- $:T_iClass_j$, $:T_iClass_jClass_k$, etc. denote the concepts or terms that form the terminology, the vocabulary or the hierarchy of concepts represented by $:Terminology_i$.

- $:Interpretation_a$ denotes the *implicit intended semantic reinterpretation* given to the hierarchy of classes subsumed by the terminology $:Terminology_i$ that will be used as values for the object property $:hasInterpretation_a$. In other words, it indicates the concept that would correspond to the *expected* range of the the object property $:hasInterpretation_a$. The $:Interpretation_a$ element is *implicit* or *hypothetical* and *not* asserted in the ontology model. Displaying this element in Figure 3.2, aims to illustrate that reusing a class as a property value may have the side-effect of modifying *implicitly* the intended original semantics of the class in the new context. This element is further explained in Section 3.2.3.

- $:hasInterpretation_a$ denotes the object property that will use the classes in the terminology $:Terminology_i$ as values. The rest of this chapter explores two distinct scenarios, based respectively on the alignment or deviation between: (a) the expected semantics of the range of this property (captured by the *hypothetical* class $:Interpretation_a$); and (b) the original intended semantics of the classes used as values (those subsumed by $:Terminology_i$).

- $:TDC$ denotes the target domain concept (the scope and universe of discourse of the pattern). The elements of the class $:TDC$ are annotated by the property $:hasInterpretation_a$ using the classes in the terminology $:Terminology_i$ as values.

- $:I_aTerminology_iTDC$, $:I_aT_iClass_jTDC$, $:I_aT_iClass_jClass_kTDC$, etc. denote a defined subclass of the target domain concept $:TDC$, whose elements contain an interpretation based on $:Interpretation_a$, of the class $:Terminology_i$, $:T_iClass_j$, $:T_iClass_jClass_k$, etc. respectively, via the object property $:hasInterpretation_a$. Note that there is a *one-to-one* relationship between one of these defined classes and the specific class from the $:Terminology_i$ hierarchy of classes that it derives from. The generic naming notation of these defined classes attempts to capture their intended semantic in the ontology. The prefix "I_a" attempt to convey that the class is an interpretation based on $:Interpretation_a$ and the suffix "TDC" the condition of subclass of $:TDC$.

- $:SpecificTDC_x$ denotes the element of the target domain concept $:TDC$ that is annotated using concepts of the terminology $Terminology_i$ as values of the object property $hasInterpretation_a$. Note, that there might be a *one-to-many* relationship between an element $:SpecificTDC_x$ and various concepts provided by the terminology $Terminology_i$. An element $:SpecificTDC_x$ can be implemented as either an owl:Class or an owl:NamedIndividual, depending on its intended semantic in the ontology model.

Based on the generic structure, elements and notation from Figure 3.2 just introduced, the mappings shown in Figure 3.3 for the concrete example in Noy's Approach 4, can be further described as follows:

- $:Terminology_1$, $:T_1Class_1$ and $:T_1Class_1Class_1$ are populated by the classes :Animal, :Lion and :AfricanLion respectively, denoting the terminology that will be reused as a *subject* of, in this case a book.
- $:Interpretation_1$ is populated by a class $:Subject$ that is not asserted, denoting the implicit interpretation of the :Animal terminology as a *subject* of, in this case a book.
- $:hasInterpretation_1$ is populated by the object property dc:subject, denoting that the elements that participate as value of this property will be considered as a *subject* of, in this case a book. In this case, the concept expected to be the range of the property dc:subject (a book subject represented in Figure 3.3 by the hypothetical class $:Subject$), deviates from the concept to be reused as the value of the property (an animal represented in Figure 3.3 by the classes :Animal, :Lion and :AfricanLion).
- $:TDC$ is populated by the class :Book, denoting and delimiting the universe of discourse of the elements that use the :Animal class hierarchy as value of the property dc:subject.
- $:I_1Terminology_1TDC$, $:I_1T_1Class_1TDC$ and $:I_1T_1Class_1Class_1TDC$ are populated by the defined classes :BooksAboutAnimals, :BooksAboutLions and

Class As Prop. Value	**OWL Implementation**
:$Interpretation_a$	(Implicit. Not asserted)
:$Terminology_i$	owl:Class (primitive)
:T_iClass_j	
:$T_iClass_jClass...$	
:TDC	
:$I_aTerminology_iTDC$	owl:Class (defined) (≡)
:$I_aT_iClass_jTDC$	
:$I_aT_iClass_jClass...TDC$	
:$SpecificTDC_x$	owl:Class (primitive)
	owl:NamedIndividual (◊)
:$hasInterpretation_a$	owl:ObjectProperty

Table 3.1: OWL Implementation of the Class As Property Value ODP.

:BooksAboutAfricanLions. Each class represents a subclass of the target domain concept :Book interpreted as a :*Subject* of the hierarchy of classes formed by :Animal, :Lion and :AfricanLion, via the object property dc:subject.

- :$SpecificTDC_x$ is populated by the individuals of the target domain concept :Book, associated to the book titles "Lions: Life in the Pride" and "The African Lion", which are annotated using concepts of the :Animal class hierarchy as values of the object property dc:subject.

Table 3.1 summarizes the elements of the generic structure of the Class As Property Value ODP in Figure 3.2 and their corresponding OWL implementation.

3.2.2 Implementation

This section provides a template implementation of the main elements of the Class As Property Value ODP that enable the pattern to meet its objective. The implementation preserves the semantic of the corresponding elements of Noy's Approach 4, except that in this case is formulated in terms of the elements of the generic structure of the pattern introduced in Figure 3.2.

The first element is one of the generic defined classes, in this case :$I_aT_iClass_jTDC$, that in Approach 4 of Noy (2005) corresponds to the class :BookAboutLions.

Definition 3.1. The implementation of a generic defined class :$I_aT_iClass_jTDC$ is given as follows:

```
:IaTiClassjTDC
    rdf:type owl:Class ;
    rdfs:subClassOf :TDC ;
    owl:equivalentClass [ rdf:type owl:Restriction ;
                          owl:onProperty :hasInterpretationa ;
```

```
                        owl:someValuesFrom :T_iClass_j
                      ] .
```

Listing 3.9: Implementation of a generic defined class :$I_aT_iClass_jTDC$

The implementation of the defined class :$I_aTerminology_iTDC$ is analogous to the implementation of :$I_aT_iClass_jTDC$ above replacing the class :T_iClass_j in the owl:someValuesFrom restriction, for the class :$Terminology_i$. The same principle applies to any other potential subclass in :$Terminology_i$ hierarchy of classes, such as :$I_aT_iClass_jClass_k$, etc.

The second element is :$SpecificTDC_x$, that in Approach 4 of Noy (2005) is implemented as :LionsLifeInThePrideBook, the owl:NamedIndividual associated to the book title "Lions: Life in the Pride".

Definition 3.2. The generic element :$SpecificTDC_x$ can be implemented as either an owl:NamedIndividual or an owl:Class. Both implementations are given below respectively:

```
:SpecificTDC_x
    rdf:type :TDC ,
             owl:NamedIndividual ;
             [ rdf:type owl:Restriction;
               owl:onProperty :hasInterpretation_a ;
               owl:someValuesFrom :T_iClass_j
             ]
             [ ... and rest of existential restrictions on property :hasInterpretation_a
                   for every class :T_iClass_j that participates in
                   the description of the individual :SpecificTDC_x
             ] .
```

Listing 3.10: Implementation of :$SpecificTDC_x$ as an owl:NamedIndividual

```
:SpecificTDC_x
    rdf:type owl:Class ;
    rdfs:subClassOf :TDC ,
                    [ rdf:type owl:Restriction ;
                      owl:onProperty :hasInterpretation_a ;
                      owl:someValuesFrom :T_iClass_j
                    ] ,
                    [ ... and rest of existential restrictions on :hasInterpretation_a
                          for every class :T_iClass_j that participates in
                          the description of the class :SpecificTDC_x
                    ] .
```

Listing 3.11: Implementation of :$SpecificTDC_x$ as an owl:Class

The implementation given of these elements results in an ontology model that maintains the same characteristics as the ontology model that results from Noy's Approach 4:

- The expressivity of the ontology is within OWL DL.
- The use of anonymous individuals in the $:Terminology_i$ class hierarchy, as the value of the object property $:hasInterpretation_a$, for the elements of the target domain concept $:TDC$.
- The implicit use of another semantic interpretation of the $:Terminology_i$ class hierarchy. The interpretation in question is linked to the semantic of the object property $:hasInterpretation_a$.

Noy (2005) acknowledges the danger of having a different interpretation of an existing hierarchy of classes than the originally intended and cautions users of the pattern to be aware of the interoperability issues that this situation may entail.

3.2.3 The term *Interpretation*

The use of the term *interpretation* made by this research is similar to that given in Noy (2005). It is used at the conceptual modelling level to refer to the intended meaning given to the conceptual elements in the ontology model.

The notion of *interpretation*, represented by the generic element $:Interpretation_a$ in the conceptual abstraction of the pattern of Figure 3.2, seeks to illustrate the fact that there are two different scenarios being coupled in the analysis of the pattern by Noy (2005). These two scenarios are: (a) using a class as the value of a property, which is the essential motivation for creating the pattern; and (b) reinterpreting the intended original meaning of a class being used as the value of a property. It is trivial to notice that scenario (b) implies scenario (a), yet this chapter and the notion of *interpretation* here, aims to show that scenario (a) does *not* have to imply (b). In other words, decoupling the fact that using a class as the value of a property, does not have to imply that the intended original meaning of such class has to be reinterpreted.

All examples in all five approaches of the CPV ODP in Noy (2005) couple these two different scenarios. It is the view of this research that treating these subtle two variants of the CPV ODP separately is important and renders a more informative understanding of its suitability and implications when applied in a given ontology model, in line with the expectations of *Research Question 2*.

The following two sections expands on the notion of *interpretation* and *terminology* in this pattern beyond the discussion in Approach 4 of Noy (2005). The sections rely on the generic structure of the pattern introduced in Figure 3.2 to guide the discussion. More specifically, Figure 3.2 and Section 3.3, where the concept of $:Interpretation_a$ and $:Terminology_i$ deviate, are put forward to characterize the CPV ODP within scenario (b) above. Conversely, Figure 3.8 and Section 3.4, where the concept of $:Interpretation_a$ and

:$Terminology_i$ conflate, are put forward to characterize the CPV ODP within scenario (a) above.

3.3 Multiple Interpretation and Terminology

Ontology designers need to be aware of the implicit nature of the :$Interpretation_a$ element and its repercussions. This additional implicit interpretation overloads the representation of instances in the :$Terminology_i$ concept hierarchy. For example, an instance of a class :T_iClass_j in the :$Terminology_i$ concept hierarchy could be of type: (a) what the class :T_iClass_j explicitly stands for; or (b) what the implicit interpretation of the class :T_iClass_j by the property :$hasInterpretation_a$ stands for. Using Noy (2005)(§Approach 4) to illustrate this scenario, an instance of the class :Lion can represent either and *actual* lion (as the class :Lion was originally intended for) or the *interpretation* of a lion as the *subject* of a specific book.

It is important to note that the Class As Property Value ODP can accommodate in the same ontology model, multiple terminologies for the same implicit interpretation but also, multiple interpretations for the same terminology concept hierarchy. Both cases are captured in the notation given to the generic structure of the pattern presented in Figure 3.2.

3.3.1 Multiple Terminology

For example, in the case of multiple terminologies for the same interpretation, consider the model from Noy (2005)(§Approach 4), where an additional terminology :$Terminology_2$ is added, whose top concept is the class :Habitat (a plausible use-case scenario). The terminology provided by the :Habitat hierarchy of classes[6] can also be implicitly interpreted as a :Subject. It then can be used as values for the property dc:subject to annotate the instances of the class :Book in the example. A graphical representation of such scenario is fairly straight forward extending the elements of the model in Figure 3.3 accordingly. See Figure 3.4.

3.3.2 Multiple Interpretation

Perhaps less intuitive is the case of multiple interpretations of the same terminology concept hierarchy. For example, consider the scenario where the terminology formed by the :Animal class hierarchy in Approach 4 of Noy (2005) is intended to be used as values

[6] Source: The Habitats Classification Scheme (Version 3.0) of the International Union for Conservation of Nature (UICN) Red List of Threatened Species - http://www.iucnredlist.org/technical-documents/classification-schemes/habitats-classification-scheme-ver3 (last visitied on August, 2011)

```
owl:Thing                                                     (CPV ODP Generic Structure)
   |-- (I) :Subject (Implicit. Not asserted)                  (:$Interpretation_1$)
      |-- :Animal                                                (:$Terminology_1$)
         |-- :Lion                                                  (:$T_1Class_1$)
            |-- :AfricanLion                                           (:$T_1Class_1Class_1$)
      |-- :Habitat                                               (:$Terminology_2$)
         |-- :Forest                                                (:$T_2Class_1$)
         |-- :Savanna                                               (:$T_2Class_2$)
         |-- :Shrubland                                             (:$T_2Class_3$)
         |-- (... rest of :Habitat subclasses)                      (:$T_2Class_j$)
   |-- :Book                                                  (:$TargetDomainConcept$)
      |-- (≡) :BookAboutAnimals                                  (:$I_1Terminology_1TDC$)
      |-- (≡) :BookAboutLions                                    (:$I_1T_1Class_1TDC$)
      |-- (≡) :BookAboutAfricanLions                             (:$I_1T_1Class_1Class_1TDC$)
      |-- (≡) :BookAboutHabitats                                 (:$I_1Terminology_2TDC$)
      |-- (≡) :BookAboutForestHabitats                           (:$I_1T_2Class_1TDC$)
      |-- (≡) :BookAboutSavannaHabitats                          (:$I_1T_2Class_2TDC$)
      |-- (... rest of defined subclasses of :Book               (:$I_1T_2Class_j...TDC$)
               based on the interpretation :Subject
               and all existing subclasses of :Habitat ...)
      |-- (◊) :TheAfricanLionBook                                (:$SpecificTDC_1$)
      |-- (◊) :LionsLifeInThePrideBook                           (:$SpecificTDC_2$)

owl:topObjectProperty
   |-- dc:subject                                             (:$hasInterpretation_1$)
```

Figure 3.4: Example of the same re-interpretation of multiple terminologies for the same Target Domain Concept.

of an additional object property :hasCharacter for the target domain concept :Book. There are *two* interpretations of the :Animal terminology required. One is :Animal as a “subject” and another is :Animal as a “fictional character”.

Figure 3.5 provides a graphical representation of what an ontology model of such scenario might look like. Some important notes regarding Figure 3.5:

- The elements :*Subject* and :*Character* are implicit and are not asserted in the ontology model.
- The elements of the :Animal concept hierarchy are asserted only once in the ontology model. In the figure they are listed twice but simply to illustrate the different use that each implicit interpretation (:*Subject* and :*Character*) makes of the concept hierarchy.
- As the generic structure of the pattern instructs, there are two elements :$I_aT_iClass_jTDC$ for each :T_iClass_j element (class :Lion). One for the interpretation :*Subject* (the defined classes :BookAboutAnimals, :BookAboutLions and :BookAboutAfricanLions) and another one for the interpretation :*Character* (the defined classes :BookStarringAnimals, :BookStarringLions and :BookStarringAfricanLions).

`owl:Thing`	(CPV ODP Generic Structure)
`  |-- (I) :`*Subject* `(Implicit. Not asserted)`	$(:Interpretation_1)$
`    |-- :Animal`	$(:Terminology_1)$
`      |-- :Lion`	$(:T_1Class_1)$
`        |-- :AfricanLion`	$(:T_1Class_1Class_1)$
`  |-- (I) :`*Character* `(Implicit. Not asserted)`	$(:Interpretation_2)$
`    |-- :Animal`	$(:Terminology_1)$
`      |-- :Lion`	$(:T_1Class_1)$
`        |-- :AfricanLion`	$(:T_1Class_1Class_1)$
`  |-- :Book`	$(:TargetDomainConcept)$
`    |-- (≡) :BookAboutAnimals`	$(:I_1Terminology_1TDC)$
`    |-- (≡) :BookAboutLions`	$(:I_1T_1Class_1TDC)$
`    |-- (≡) :BookAboutAfricanLions`	$(:I_1T_1Class_1Class_1TDC)$
`    |-- (≡) :BookStarringAnimals`	$(:I_2Terminology_1TDC)$
`    |-- (≡) :BookStarringLions`	$(:I_2T_1Class_1TDC)$
`    |-- (≡) :BookStarringAfricanLions`	$(:I_2T_1Class_1Class_1TDC)$
`    |-- (◊) :TheAfricanLionBook`	$(:SpecificTDC_1)$
`    |-- (◊) :LionsLifeInThePrideBook`	$(:SpecificTDC_2)$
`    |-- (◊) :TheLionWhoWantedToLoveBook`	$(:SpecificTDC_3)$
`owl:topObjectProperty`	
`  |-- dc:subject`	$(:hasInterpretation_1)$
`  |-- :hasCharacter`	$(:hasInterpretation_2)$

Figure 3.5: Example of multiple interpretations of the same Terminology for the same Target Domain Concept.

Based on the model from Figure 3.5, and the implementation of a $:SpecificTDC_x$ element given in Definition 3.2, the classes in the :Animal concept hierarchy can be used as *values* to indicate the subject and the main character for the elements :LionsLifeInThePrideBook and :TheLionWhoWantedToLoveBook.

For example, the implementation of :TheLionWhoWantedToLoveBook applying Definition 3.2 would be:

```
:TheLionWhoWantedToLoveBook
    rdf:type :Book ,
             owl:NamedIndividual ;
             [ rdf:type owl:Restriction;
               owl:onProperty :hasCharacter ;
               owl:someValuesFrom :Lion
             ]
             [ rdf:type owl:Restriction;
               owl:onProperty cd:subject ;
               owl:someValuesFrom :Lion
             ] .
```

Listing 3.12: Implementation of :TheLionWhoWantedToLoveBook with two interpretations of the :Animal concept hierarchy

Assuming that the implementation of the defined classes :BookAboutLions and :BookStarringLions follows that in Definition 3.1, this implementation of :TheLionWhoWantedToLoveBook, not only uses the classes in the :Animal concept hierarchy as values for the properties dc:subject and :hasCharacter (as intended by the CPV ODP), but also allows an OWL DL reasoner to infer that :TheLionWhoWantedToLoveBook is of type :BookAboutLions and :BookStarringLions (another key feature of the pattern as presented in Noy (2005)(§Approach 4)).

3.4 Conflation of Interpretation and Terminology

There is yet another special case that greatly simplifies the generic structure of the Class As Property Value ODP given in Figure 3.2. Consider a scenario where the original interpretation of the terminology to reuse, conflates with the actual interpretation of what the terminology originally represents. In general terms, a scenario in which the implicit semantic of :$Interpretation_a$ conflates with the explicit semantic of what :$Terminology_i$ originally represents. This case is somewhat hard to visualize in the example from Noy (2005)(§Approach 4) with the elements :Book, :Animal and dc:subject. The implicit interpretation of the :Animal concept hierarchy as a subject is clearly different from what the :Animal concept hierarchy actually represents. However, now consider this example in a new domain with the following changes in relation to the example of Noy (2005)(§Approach 4):

- The :TDC element is populated with the class :DishwasherDetergent instead of :Book.
- The :$Terminology_i$ element is populated with the class :Brand instead of :Animal.
- The :$hasInterpretation_a$ element is populated with the object property :hasBrand instead of dc:subject.
- The :$Interpretation_a$ element is populated with the implicit interpretation that the property :$hasInterpretation_a$ introduces, which in this case is :$Brand$ instead of :$Subject$.

With the changes above, the meaning of :$Interpretation_a$ and :$Terminology_i$ conflate. Both of them stand for the same notion represented in the model by the class :Brand. Therefore, there is no need to differentiate between them. In addition, the meaning of the object property :$hasInterpretation_a$ and a hypothetical object property :$hasTerminology_i$ would also conflate. Figure 3.6 visualizes the example ontology model as a result of the changes mentioned. Consequences of the conflation of :$Interpretation_a$ and :$Terminology_i$ in the figure include:

- The elements :$Interpretation_1$ and :$Terminology_1$ are merged into one. In this case the result is the class :Brand.

- The object property :$hasInterpretation_1$ becomes :$hasTerminology_1$ to align to :$Terminology_1$. In this case :hasBrand.

- The elements of the form :$I_aT_iClass_jTDC$ become :T_iClass_jTDC given that the semantic contribution added by :$Interpretation_a$ is already inherent in :T_iClass_jTDC as a consequence of the semantic conflation of :$Interpretation_a$ and :$Terminology_a$. For example, in the case of Noy (2005)(§Approach 4) the element :$I_1T_1Class_1TDC$ is populated by :BooksAboutLions, which represents the semantic combination of :Subject (:$Interpretation_1$), :Lion (:T_1Class_1) and :Book (:TDC).
In the case of the example in Figure 3.6, a hypothetical element :$I_1T_1Class_1TDC$ would be populated by :BrandCascadeDishDetergent, which would represent the semantic combination of :Brand (:$Interpretation_1$), :Cascade (:T_1Class_1) and :DishwasherDetergent (:TDC).
However, because of the conflation of :$Interpretation_1$ and :$Terminology_1$, the semantic contribution of :$Interpretation_1$ (:Brand) is already inherent in :T_1Class_1 (:Cascade) causing the elements :$I_1T_1Class_1TDC$ (:BrandCascadeDishDetergent) and :T_1Class_1TDC (:CascadeDishDetergent) to conflate as well.
This behaviour is conveyed in the figure using brackets "[...]" around the prefix "I_1" to symbolize the unnecessary semantic contribution of :$Interpretation_1$ in the elements :$[I_1]T_1Class_1TDC$ and :$[I_1]T_1Class_2TDC$. In other words, the prefix "I_a" is not needed.

- The element :$[I_1]Terminology_1TDC$ is not needed because it is equivalent to the whole universe of discourse represented by the class :DishwasherDetergent, given that every dishwashing detergent has some type of brand.
A hypothetical element :$[I_1]Terminology_1TDC$ could be named for example, :Branded-DishDetergent or :DishDetergentWithABrand which, considering that all individual dishwashing detergents are expected to have a brand, is essentially equivalent to the whole domain of discourse represented already by the target domain concept :DishwasherDetergent. Therefore this element does not need to be asserted in the ontology model.

The previous example illustrates the consequences in the Class As Property Value ODP when the interpretation of the terminology to reuse and the terminology itself conflate. To further illustrate these consequences, consider now another example but this time much more similar to the original use of the pattern in Noy (2005)(§Approach 4). Let us use the same elements than in Noy (2005)(§Approach 4) changing only the domain of discourse, the :TDC element (:Book) and the object property dc:subject. Let us use as domain of discourse, the concept of :Zoo, and as object property the relation :hasAnimal. In summary:

- The :TDC element is populated with the class :Zoo instead of :Book.

```
owl:Thing                                          (ODP Generic Structure)
  |-- (I) :Brand (Implicit. Not asserted)          (:Interpretation_1)
    |-- :Brand                                        (:Terminology_1)
      |-- :Cascade                                       (:T_1Class_1)
      |-- :Electrasol                                    (:T_1Class_2)
      |-- (...)                                          (...)
  |-- :DishwasherDetergent                         (:TDC)
    |-- (≡) _:BrandedDishDetergent                    (:[I_1]Terminology_1TDC)
    |-- (≡) :CascadeDishDetergent                     (:[I_1]T_1Class_1TDC)
    |-- (≡) :ElectrasolDishDetergent                  (:[I_1]T_1Class_2TDC)
    |-- (...)                                         (...)

owl:topObjectProperty
  |-- :hasBrand                                    (:hasInterpretation_1) same as
                                                   (:hasTerminology_1)
```

Figure 3.6: Example of conflation of the elements :$Interpretation_a$ and :$Terminology_i$

- The :$Terminology_i$ element does not change and is populated with the class :Animal.
- The :$hasInterpretation_a$ element is populated with the object property :hasAnimal (which naturally applies to a given zoo) instead of dc:subject.
- The :$Interpretation_a$ element is populated with the implicit interpretation that the property :$hasInterpretation_a$ introduces, which in this case is actually :$Animal$ instead of :$Subject$.

Again, with the changes above, the meaning of :$Interpretation_a$ and :$Terminology_i$ conflate. Now, both of them stand for the same notion, represented in the model by the single class :Animal. In addition, the meaning of the object property :$hasInterpretation_a$ (:hasAnimal) and a hypothetical object property :$hasTerminology_i$ also conflate. Figure 3.7 visualizes the example ontology model as a result of the changes mentioned.

In generic terms, the previous example is hinting that the conflation of the interpretation and terminology elements in the pattern simplifies significantly the generic structure of the resulting ontology model. The changes with respect to the initial generic structure proposed can be summarised as follows:

- The implicit class :$Interpretation_a$ and the explicit class :$Terminology_i$ conflate, therefore only the explicit class :$Terminology_i$ is represented in the ontology structure.
- The object property :$hasInterpretation_a$ would conflate with a hypothetical object property :$hasTerminology_i$ and thus, for consistency only the latter is represented in the ontology structure.

```
owl:Thing                                       (ODP Generic Structure)
   |-- (I) :Animal (Implicit. Not asserted)     (:Interpretation_1)
      |-- :Animal                                  (:Terminology_1)
         |-- :Lion                                    (:T_1Class_1)
            |-- :AfricanLion                             (:T_1Class_1Class_1)
   |-- :Zoo                                     (:TargetDomainConcept)
      |-- (≡) :ZooWithAnimals                      (:[I_1]Terminology_1TDC)
      |-- (≡) :ZooWithLions                        (:[I_1]T_1Class_1TDC)
      |-- (≡) :ZooWithAfricanLions                 (:[I_1]T_1Class_1Class_1TDC)
      |-- (◊) :LondonZoo                           (:SpecificTDC_1)
      |-- (◊) :NewYorkZoo                          (:SpecificTDC_2)

owl:topObjectProperty
   |-- :hasAnimal                               (:hasInterpretation_1) same as
                                                (:hasTerminology_i)
```

Figure 3.7: Example of conflation of the elements $:Interpretation_a$ and $:Terminology_i$ derived from the example of Approach 4 by Noy (2005), in relation to the generic structure in Figure 3.2.

- The defined class $:I_aTerminology_iTDC$ becomes equivalent to the target domain concept class $:TDC$ given that the compound semantics of $:Interpretation_a$, $:Terminology_i$ and $:TDC$ that come together into the class name $:I_aTerminology_iTDC$ are already represented by the class $:TDC$.
  This situation was already addressed in the example of Figure 3.6 for the target domain concept :DishwasherDetergent. In the case of the example in Figure 3.7, the interpretation $:Animal$, the terminology :Animal and the target domain :Zoo, would generate a defined class $:I_aTerminology_iTDC$ named such as :ZooWithAnimals or similar. Such a class is trivial and equivalent to the target domain concept :Zoo. Thus, only the class $:TDC$ is represented in the ontology structure.

- The rest of the defined class elements in the original generic structure, $:I_aT_iClass_jTDC$, $:I_aT_iClass_jClass_kTDC$, etc., becomes the defined classes $:T_iClass_jTDC$, $:T_iClass_jClass_kTDC$, etc. The prefix "I_a" is not needed given that $:Interpretation_a$ and $:Terminology_i$ represent now the same thing.

With this changes in mind, the generic structure of the ontology schema behind the Class As Property Value ODP in Approach 4 of Noy (2005), is significantly simplified when the implicit interpretation of the terminology to reuse as property values conflate with explicit interpretation of the original terminology itself. Figure 3.8 introduces the resulting simplified generic structure.

Table 3.2 summarizes the OWL implementation of the elements in the simplified generic structure of the Class As Property Value ODP in Figure 3.8, reflecting the changes upon the conflation of $:Interpretation_a$ and $:Terminology_i$.

```
owl:Thing
    |-- :Terminology_i
        |-- :T_iClass_j
            |-- (... rest of potential subclasses)
    |-- :TargetDomainConcept (or :TDC)
        |-- (≡) :T_iClass_jTDC
        |-- (... rest of defined subclasses of :TDC based on
                 all existing subclasses of :Terminology_i)
        |-- [:SpecificTDC_x | (◊) :SpecificTDC_x]

owl:topObjectProperty
    |-- :hasTerminology_i
```

Figure 3.8: Generic structure of the Class As Prop. Value ODP conflating $:Interpretation_a$ and $:Terminology_i$

Class As Prop. Value	OWL Implementation
$:Terminology_i$	owl:Class (primitive)
$:T_iClass_j$	
$:T_iClass_jClass...$	
$:TDC$	
$:Terminology_iTDC$	owl:Class (defined) (≡)
$:T_iClass_jTDC$	
$:T_iClass_jClass...TDC$	
$:SpecificTDC_x$	owl:Class (primitive)
	owl:NamedIndividual (◊)
$:hasTerminology_i$	owl:ObjectProperty

Table 3.2: OWL Implementation of the Class As Property Value ODP conflating $:Interpretation_a$ and $:Terminology_i$.

This simplified scenario of the Class As Property Value ODP characterized in Figure 3.8 is particularly important for the comparative analysis between this pattern and the Value Partition ODP first, and the Normalisation ODP later, to be presented in the chapters that follow.

3.5 Class versus Individual

A recurrent modelling decision to be made in ontology design is whether a particular concept should be represented as a class or as an individual. For example, it could be argued that the subclasses of "Habitat" in Figure 3.4 or "Brand" in Figure 3.6 should be in fact, individuals instead. Noy and McGuinness (2001)(§ 4.6) discuss basic guidelines to address this decision based on the potential application of the ontology model. The application determines the level of granularity required for the conceptual elements that compose the ontology model, so the optimal representation is not only a conceptual matter but a pragmatic one as well.

Individual instances correspond to the most specific concepts in the ontology specification and cannot be further refined. Classes on the other hand, lie at a broader conceptual level that can be specialized further. In some cases, this limitation of individuals is the key factor to consider. For example, as covered in Chapter 4, this is one the factors that motivated Rector (2005) to provide two parallel implementations of his Value Partition ODP for the same ontological concept, one using classes and another using individuals.

With these considerations in mind, the examples of "Habitat" and "Brand" in this Chapter, are represented as a hierarchy of classes given that: (a) in the context of ODPs, where it is important to propose generic reusable solutions, it is plausible to conceive an application that may require to specialize further some of the classes that are part of the "Habitat" or "Brand" class hierarchy respectively; and (b) the purpose of the CPV pattern requires the use of a hierarchy of classes.

3.6 Conclusions

The generic structure of the ontology schema of the Class As Property Value ODP is introduced based on the example of Approach 4 by Noy (2005).

A prototypical implementation of the generic structure main elements required by the pattern is provided. The generic structure and implementation accommodate the notion of interpretation with the elements $:Interpretation_a$ and $:hasInterpretation_a$ and characterize their relation with the rest of the elements in the pattern.

The notion of interpretation in the pattern is discussed further beyond the analysis in Approach 4 of Noy (2005). Additional scenarios involving multiple interpretations and multiple terminologies are characterized relying on the generic structure and implementation of the pattern presented.

As a result, two versions of the pattern are identified: (a) the most generic version, in which the notion of interpretation $:Interpretation_a$ and terminology $:Terminology_i$ are distinct; and (b) the simplified version, in which the notion of interpretation and terminology conflates.

The conflation of the notion of interpretation and terminology simplifies significantly the generic structure of the pattern, in which an existing terminology class hierarchy is still reused as property values, but the original interpretation of this terminology is not altered.

This characterisation of the notion of interpretation in the Class As Property Value ODP, decouples two versions of the pattern that up to now, have not been explicitly considered, in line with the goals of *Research Question 2* of Section 1.4. More specifically, the ontology schema described by the use of multiple interpretations $:Interpretation_a$

for the *same* terminology :$Terminology_a$, is put forward as a partial solution to *Scenario (b)* of Section 1.1.3.

Chapter 4

Revisiting the Value Partition ODP

This chapter presents a revision of the Value Partition ODP introduced by Rector (2005) and further revisited in Egana-Aranguren (2005)(§ 4.3.2.2), Egana-Aranguren (2009)(§ A.17).

The revision covers two versions of the pattern based on their implementation as discussed in Rector (2005). An abstract general structure of the ontology schema behind the two versions is proposed and a generic implementation in OWL of the elements in that structure is also provided.

This revision will be also used to illustrate the alignments among, initially the Value Partition ODP and the Class As Property Value ODP, and in the subsequent chapter, the Normalisation ODP. These alignments are revealed by a comparative analysis of the generic ontology schema behind these patterns, that is key to the contributions of this research and to be discussed in subsequent chapters.

4.1 Value Partition ODP

Rector (2005) introduces the *value partition* pattern to address a modelling scenario, where a set of elements is used to represent a descriptive feature (also referred to as attribute, modifier or characteristic), of some other entity in the ontology. This feature is constrained by a set of possible values (known as feature space) and Rector proposed two different patterns to represent the feature and its feature space:

- Pattern 1, where the feature is represented as a class and the feature space as an enumeration of individuals that belong to and exhaust the class. Section "Pattern

1: Values as sets of individuals" of Rector (2005). The author also refers to this approach as *value set* and Egana-Aranguren (2009) as *enumeration*.

- Pattern 2, where the feature is represented as a class and the feature space as a set of pairwise disjoint subclasses that together exhaust the parent (feature) class. This type of class structure is also known as a *partition*. Section "Pattern 2: Values as subclasses partitioning a feature" of Rector (2005).

To introduce these two value partition patterns, Rector uses an example in the context of the health condition of a person, where the concept "health" is the feature to represent and the concepts "good", "medium" and "bad" are the feature space. The author addresses this modelling scenario with Pattern 1 and Pattern 2 then, discusses some advantages and drawbacks of the resulting ontology model for each case, including its OWL expressivity, which in both cases is within OWL DL.

The revision of the Value Partition ODP throughout this chapter focuses on Section "Representation using variant 2: Placing an existential restriction on the individual" of Rector (2005) that is one of the two variations of Pattern 2.

Figure 4.1 illustrates the example used by the mentioned Pattern 2–Variant 2. The example depicts the feature class :HealthValue partitioned by the classes :Poor_health_value, :Medium_health_value and :Good_health_value. These three subclasses represent a value partition of the parent class :HealthValue as per the definition stated earlier: the three are mutually exclusive and their union is equivalent to the parent class :HealthValue. The example also depicts the idea of using an anonymous individual, :Johns_Health, that belongs to one of the value partition classes, :Good_health_value, as the value to represent a person's health, in this case the individual :John that belongs to the class :Person.

There are two aspects of Pattern 2–Variant 2 of Rector (2005) based on the proposed implementation of the pattern that is particularly important when compared to the Class As Property Value and Normalisation ODP, namely:

- The expressivity of the ontology model in the pattern is within OWL DL.
- The use of anonymous individuals in the :HealthValue class hierarchy as the value of the object property :has_health_status, for the instances of the class :Person.

4.1.1 The term "Value Partition"

A clarification in relation to the title of this chapter might be in order. Even though Pattern 2–Variant 2 is particularly relevant in the scope of this research, this chapter aims to abstract the underlying structure of the two known implementations of the

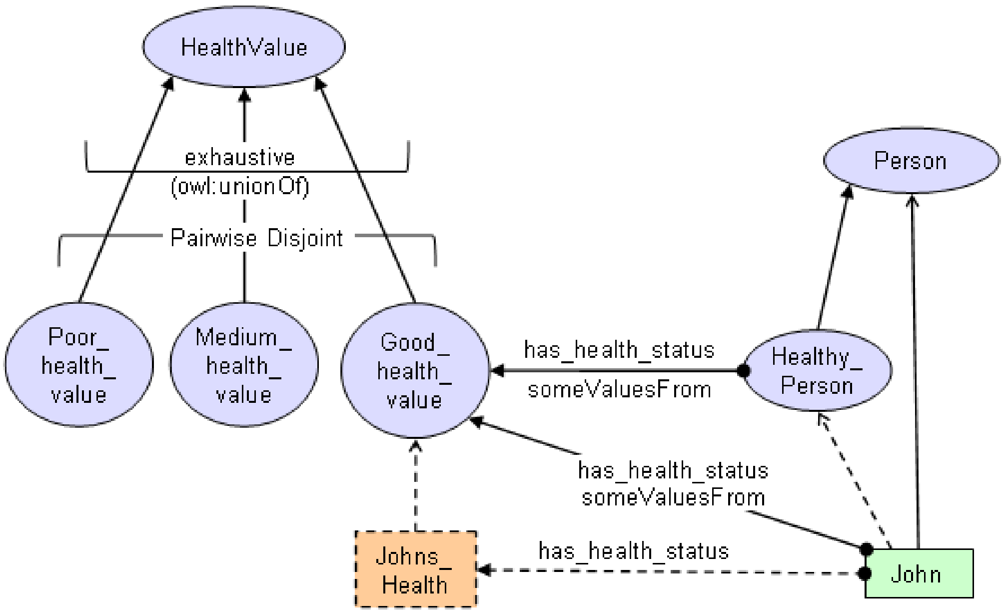


Figure 4.1: *"Pattern 2 variant 2 with an anonymous individual for John's Health"* (Fig. 4 in Rector (2005)).

pattern, that is, using individuals as in Pattern 1 of Rector (2005) (also known as *value set* or *enumeration* pattern), or classes as in Pattern 2 of Rector (2005) (also referred to as *value partition* pattern per se.

At the same time, a term is needed to refer to this pattern as a *generic* concept, meaning, *representing the full set of values that a feature can take*, independently of the chosen implementation. This chapter uses the term "Value Partition" (VP) ODP, for this purpose. In cases, where the specific implementation of the pattern is relevant, this is named explicitly.

The following section abstracts the ontology schema behind Pattern 1 and Pattern 2 of Rector (2005), providing a generic structure that accommodates all the elements that participate in the pattern and a generic implementation that preserves all the characteristics of both versions of the pattern.

4.1.2 Structure and Elements

There are instances of a generic structure for the Value Partition ODP in Egana-Aranguren (2005)(§ 4.3.2.2), Egana-Aranguren (2009)(§ A.17) and even online[1], based on the patterns described by Rector (2005). Figure 4.2 presents the specific version of the generic structure that will be used hereafter to anchor the discussion of the pattern using the notation introduced in Chapter 3–Section 3.1. This structure accommodates

[1]http://odps.sourceforge.net/ (see § Value Partition)

```
owl:Thing
    |-- :Modifier
        |-- (≡)(P) :Value_i
            |-- [:V_iPart_j | (◊) :V_iPart_j]
    |-- :Self_standing_entity
        |-- :TargetDomainConcept (or :TDC)
            |-- (≡) :V_iPart_jTDC
            |-- [:SpecificTDC_x | (◊) :SpecificTDC_x]

owl:topObjectProperty
    |-- :hasValue_i
```

Figure 4.2: Generic structure of the Value Partition ODP in Rector (2005).

```
(Pattern 1)                         (Pattern 2)                           (Generic Structure)
owl:Thing                           owl:Thing
  |-- :Modifier                       |-- :Modifier
    |-- (≡)(P) :Health_Value            |-- (≡)(P) :Health_Value            (:Value_1)
      |-- (◊) :good_health                |-- :Good_health_value            (:V_1Part_1)
      |-- (◊) :medium_health              |-- :Medium_health_value          (:V_1Part_2)
      |-- (◊) :poor_health                |-- :Poor_health_value            (:V_1Part_3)
  |-- :Self_standing_entity           |-- :Self_standing_entity
    |-- :Person                         |-- :Person                       (:TDC)
      |-- (≡) :Healthy_person             |-- (≡) :Healthy_person           (:V_1Part_1TDC)
      |-- (◊) :John                       |-- (◊) :John                     (:SpecificTDC_1)

owl:topObjectProperty               owl:topObjectProperty
  |-- :has_health_status              |-- :has_health_status              (:hasValue_1)
```

Figure 4.3: Alignment of the elements of the two versions of the Value Partition ODP in Rector (2005), in relation to the generic structure in Figure 4.2.

both versions of the pattern presented in Rector (2005) (Pattern 1 and Pattern 2), while preserving the same functionality and semantic expressivity.

The generic structure, elements and notation used in Figure 4.2 to characterize the Value Partition ODP are explained below. Figure 4.3 is also provided to assist with this explanation. Figure 4.3 shows the ontology examples of both versions of the pattern in Rector (2005), (Pattern 1 and Pattern 2), and how they align to the generic ontology schema in Figure 4.2. The ontology models used in Figure 4.3 for the examples in Rector (2005) are provided by the author and available online (Pattern 1[2] and Pattern 2[3]).

- $:Value_i$ denotes the descriptive feature (the attribute, the modifier, the characteristic) of some other entity in the ontology. It is a defined class and it is the class to be partitioned.
- $:V_iPart_j$ denotes one of the specific values in the feature space of the main feature $:Value_i$. The set of all elements $:V_iPart_j$ partitions the main feature represented

[2] http://www.w3.org/TR/swbp-specified-values/values-as-individuals-01.owl
[3] http://www.w3.org/TR/swbp-specified-values/value-partitions-variant-1.owl

by the parent class :$Value_i$. In Pattern 1 of Rector (2005), all elements :V_iPart_j are implemented as an owl:NamedIndividual. In Pattern 2 of Rector (2005), all elements :V_iPart_j are implemented as an owl:Class.

- :$hasValue_i$ denotes an object property that links a specific value :V_iPart_j of the descriptive feature $Value_i$, to some other entity in the ontology. The object property is implemented as functional (owl:FunctionalProperty). Making the :$hasValue_i$ property functional enforces that only a single value :V_iPart_j that partitions the main feature $Value_i$, is assigned to a given element in the ontology. This is consistent with the fact that the set of elements :V_iPart_j that exhaust the main feature and parent class $Value_i$ are intended to be pairwise disjoint.

- :TDC denotes the target domain concept in the pattern. It represents the scope and universe of discourse of the elements to be described using the values of the main feature :$Value_i$.

- :V_iPart_jTDC denotes a defined subclass of the target domain concept :TDC, defined by a relationship to a single feature value :V_iPart_j of the main feature class :$Value_i$, via the property :$hasValue_i$. Note there is a *one-to-one* relationship between a defined class :V_iPart_jTDC and the specific value :V_iPart_j that it derives from.

- :$SpecificTDC_x$ denotes an element from the target domain concept :TDC that is described with one of the specific values :V_iPart_j of the main feature class :$Value_i$, via the property :$hasValue_i$. Note that a given element :$SpecificTDC_x$ can be described with only one specific value :V_iPart_j. An element :$SpecificTDC_x$ can be implemented as an owl:NamedIndividual or as an owl:Class on both versions of the pattern (Pattern 1 and Pattern 2) in Rector (2005). The implementation chosen, depends on the required semantic of the element :$SpecificTDC_x$ in the ontology model.

Table 4.1 summarizes the elements of the generic structure of the Value Partition ODP in Figure 4.2 just introduced and their corresponding OWL implementation.

As stated earlier, Figure 4.3 illustrates the placement of the elements from the two pattern examples in Rector (2005) (Pattern 1 and Pattern 2) in the generic structure of the Value Partition ODP in Figure 4.2. Note from Figure 4.3, that the *one-to-one* relationship between the elements :V_iPart_j and :V_iPart_jTDC exists, although it is implemented only for one element :V_iPart_j, namely: (a) the individual :good_health in Pattern 1; and (b) the class :Good_health_value in Pattern 2. In this case, the elements :V_1Part_2TDC (that would derive from the individual :medium_health or the class :Medium_health_value) and :V_1Part_3TDC (that would derive from :poor_health or :Poor_health_value) from the generic structure are not populated in either of the examples. This is fine. Not *all* defined classes :V_iPart_jTDC may be required in order

Value Partition	**OWL Implementation**
:$Value_i$	owl:Class (defined) (≡)(P)
:V_iPart_j	owl:Class (primitive)
	owl:NamedIndividual (◊)
:TDC	owl:Class (primitive)
:V_iPart_jTDC	owl:Class (defined) (≡)
:$SpecificTDC_x$	owl:Class (primitive)
	owl:NamedIndividual (◊)
:$hasValue_i$	owl:ObjectProperty owl:FunctionalProperty

Table 4.1: OWL Implementation of the Value Partition ODP.

for the pattern to meet its intended purpose. If these two elements, :V_1Part_2TDC and :V_1Part_3TDC, were hypothetically populated, two defined subclasses of :Person, such as for example :Medium_health_person and :Poor_health_person respectively, would be added to the ontology model.

Another scenario of the Value Partition ODP to consider, although not recommended by the author, is that the two versions of the pattern, Pattern 1 and Pattern 2 of Rector (2005), may coexist in the same ontology model for two distinct features :$Value_i$ of the same target domain concept :TDC. A feature class :$Value_1$, whose elements :V_1Part_j are represented as an owl:NamedIndividual and a feature class :$Value_2$, whose elements :V_2Part_j are represented as an owl:Class. The generic structure in Figure 4.2 already accommodates such scenario by populating it accordingly.

4.1.3 Implementation

This section provides a template implementation of the main elements of the Value Partition ODP generic structure in Figure 4.2 introduced earlier. The implementation preserves the semantic of the corresponding element provided in the examples of Rector (2005) although in this case, it is given in generic terms.

Definition 4.1. The implementation of a generic defined class :$Value_i$ varies depending on whether the element :V_iPart_j is represented as an owl:NamedIndividual or as an owl:Class. Both are given below respectively:

```
:Value_i
    rdf:type owl:Class ;
    owl:equivalentClass [ rdf:type owl:Class ;
                          owl:oneOf ( :V_iPart_j
                                      ... and rest of individuals :V_iPart_j
                                          that exhaust the class :Value_i
                                    )
                        ] .
```

Listing 4.1: Implementation of :$Value_i$ with :V_iPart_j as an owl:NamedIndividual

```
:Value_i
    rdf:type owl:Class ;
    owl:equivalentClass [ rdf:type owl:Class ;
                          owl:unionOf ( :V_iPart_j
                                        ... and rest of subclasses :V_iPart_j
                                            that exhaust the class :Value_i
                                      )
                        ] .
```

Listing 4.2: Implementation of :$Value_i$ with :V_iPart_j as an owl:Class

Definition 4.2. The implementation of a generic defined class :V_iPart_jTDC captures the *one-to-one* relationship between :V_iPart_jTDC and the corresponding element :V_iPart_j that it is derived from. Again, the implementation varies on whether :V_iPart_j is represented as an owl:NamedIndividual or as an owl:Class. Both implementation follow below respectively:

```
:V_iPart_jTDC
    rdf:type owl:Class ;
    rdfs:subClassOf :TDC ;
    owl:equivalentClass [ rdf:type owl:Restriction ;
                          owl:onProperty :hasValue_i ;
                          owl:hasValue :V_iPart_j
                        ] .
```

Listing 4.3: Implementation of :V_iPart_jTDC with :V_iPart_j as an owl:NamedIndividual

```
:V_iPart_jTDC
    rdf:type owl:Class ;
    rdfs:subClassOf :TDC ;
    owl:equivalentClass [ rdf:type owl:Restriction ;
                          owl:onProperty :hasValue_i ;
                          owl:someValuesFrom :V_iPart_j
                        ] .
```

Listing 4.4: Implementation of :V_iPart_jTDC with :V_iPart_j as an owl:Class

Definition 4.3. There are four possible implementations of a generic element :$SpecificTDC_x$ according to the generic structure presented for the pattern. They vary depending on whether :$SpecificTDC_x$ is represented as an owl:NamedIndividual or as an owl:Class; and on whether the element :V_iPart_j is represented as an owl:NamedIndividual or an owl:Class as well. The four combinations are given below:

```
:SpecificTDC_x
    rdf:type :TDC ,
             owl:NamedIndividual ,
             [ rdf:type owl:Restriction;
               owl:onProperty :hasValue_i ;
```

```
            owl:hasValue :ViPartj
          ] .
```

Listing 4.5: Implementation of :$SpecificTDC_x$ as an owl:NamedIndividual with :V_iPart_j as an owl:NamedIndividual

```
:SpecificTDCx
    rdf:type owl:Class ;
    rdfs:subClassOf :TDC ,
                    [ rdf:type owl:Restriction ;
                      owl:onProperty :hasValuei ;
                      owl:hasValue :ViPartj
                    ] .
```

Listing 4.6: Implementation of :$SpecificTDC_x$ as an owl:Class with :V_iPart_j as an owl:NamedIndividual

```
:SpecificTDCx
    rdf:type :TDC ,
             owl:NamedIndividual ,
             [ rdf:type owl:Restriction;
               owl:onProperty :hasValuei ;
               owl:someValuesFrom :ViPartj
             ] .
```

Listing 4.7: Implementation of :$SpecificTDC_x$ as an owl:NamedIndividual with :V_iPart_j as an owl:Class

```
:SpecificTDCx
    rdf:type owl:Class ;
    rdfs:subClassOf :TDC ,
                    [ rdf:type owl:Restriction ;
                      owl:onProperty :hasValuei ;
                      owl:someValuesFrom :ViPartj
                    ] .
```

Listing 4.8: Implementation of :$SpecificTDC_x$ as an owl:Class with :V_iPart_j as an owl:Class

The implementation of these elements given, results in an ontology model that maintains the same characteristics as the ontology model in the examples of Pattern 1 and Pattern 2 by Rector (2005), in particular:

- The expressivity of the ontology is within OWL DL.
- In the case of Pattern 2–Variant 2 of Rector (2005), the use of anonymous individuals in the :$Value_i$ feature, as the value of the object property :$hasValue_i$, to describe elements of the target domain concept :TDC.

Note already some of the similarities in the resulting ontology model of the generic implementation of the Value Partition ODP and the counterpart ontology model of the Class As Property Value ODP in the previous chapter.

The following sections expands on the similarities and differences between these two patterns, Value Partition and Class As Property Value, relying on their proposed generic structure to guide the discussion.

4.2 Alignment of the Value Partition and Class As Prop. Value ODPs

An initial comparative analysis between the two examples of the Value Partition ODP and the Class As Property Value ODP, namely Pattern 2–Variant 2 of Rector (2005) and Approach 4 of Noy (2005) respectively, can be found in Rodriguez-Castro and Glaser (2008c) and preliminary work online [4], [5]. The comparison was an initial attempt to identify solutions to the multiple usages that representation of "Fault" has to support in the D&S ontology of ReSIST, as indicated in *Scenario (b)* of Section 1.1.3. However, the analysis in Rodriguez-Castro and Glaser (2008c) is partially complete because it did not factor the implications of having a separate interpretation of the terminology to reuse that can take place in Approach 4 of Noy (2005).

The following sections expand on the comparative analysis between the Value Partition ODP and the Class As Property Value ODP in several ways:

- They distinguish between the two scenarios of the Class As Property Value ODP introduced earlier: (a) the most generic case, where the terminology to reuse and its interpretation are different (Figure 3.2); and (b) the simplified case, where the terminology to reuse and its interpretation conflate (Figure 3.8).
- They use the generic structure of both ODPs to anchor the comparison so that the results can be extrapolated to any specific use case of the patterns.

4.2.1 Separation of Interpretation and Terminology

The comparison between the Value Partition ODP (Figure 4.2) and the most generic case of the Class As Value Property ODP (Figure 3.2) is problematic, because even though from an implementation standpoint there are clear similarities, from an ontological standpoint the similarities are difficult to justify.

[4] http://lists.w3.org/Archives/Public/public-owl-dev/2007OctDec/0078.html
[5] http://lists.w3.org/Archives/Public/public-swbp-wg/2007Oct/0002.html

Value Partition	**Class As Prop. Value**	**OWL Implementation**
$:Value_i$	(Not applicable)	owl:Class (defined) (≡)(P)
(Not applicable)	$:Interpretation_i$	(owl:Class - not asserted) (I)
$:V_iPart_j$	$:Terminology_i$	owl:Class (primitive)
	$:T_iClass_j$	
	(Not applicable)	owl:NamedIndividual (◊)
$:TDC$		owl:Class (primitive)
$:V_iPart_jTDC$	$:I_aTerminology_iTDC$	owl:Class (defined) (≡)
	$:I_aT_iClass_jTDC$	
$:SpecificTDC_x$		owl:Class (primitive)
		owl:NamedIndividual (◊)
$:hasValue_i$	$:hasInterpretation_i$	owl:ObjectProperty owl:FunctionalProperty*

(∗) The Value Partition ODP requires the object property to be functional. The Class As Prop. Value ODP does not.

Table 4.2: Alignment of the elements in the generic structures of the Value Partition and Class As Property Value ODPs.

The main ontological issue has to do with the separate interpretation that the Class As Property Value ODP does of the terminology to be reused. This separate interpretation does not take place in the case of the Value Partition ODP.

In the case under consideration, both patterns are similar from an implementation standpoint in the sense that both use a hierarchy of classes as values for an object property in the ontology model. Class As Prop. Value uses the classes subsumed by $:Terminology_i$ as the value of the property $:hasInterpretation_a$. Value Partition uses the classes subsumed by $:Value_i$ as the value of the property $:hasValue_i$.

However, the patterns are different from an ontological standpoint in the sense that Class As Prop. Value changes the interpretation of the anonymous individuals of the reused class $:Terminology_i$, while Value Partition preserves the same interpretation of the anonymous individuals of the class $:Value_i$.

Table 4.2 attempts to compare side by side the elements that participate on both patterns for the case under consideration, based on their implementation; bringing together:

(a) the elements of the generic structure of the Value Partition ODP introduced in Figure 4.2 and Table 4.1; and

(b) the elements of the generic structure of the Class As Property Value ODP introduced in Figure 3.2 and Table 3.1.

The alignments across both patterns are made based on the semantic equivalences in the prototypical implementation of the elements involved. The generic implementation of the main elements in both patterns was introduced in Definition 4.1, 4.2 and 4.3 for

```
Value Partition                         Class As Property Value
(Pattern 2 - Variant 2)                 (Approach 4)
owl:Thing                               owl:Thing
  |-- (≡)(P) :Health_Value                |-- (I) :Subject (Not Asserted)
    |-- :Good_health_value                  |-- :Animal
    |                                         |-- :Lion
    |                                           |-- :AfricanLion
    |-- :Medium_health_value
    |-- :Poor_health_value
  |-- :Person                             |-- :Book
    |-- (≡) :Healthy_person                 |-- (≡) :BookAboutAnimal
    |                                       |-- (≡) :BookAboutLion
    |                                       |-- (≡) :BookAboutAfricanLion
    |-- (◊) :John                           |-- (◊) :TheAfricanLionBook
                                            |-- (◊) :LionsLifeInThePrideBook

owl:topObjectProperty                   owl:topObjectProperty
  |-- :has_health_status                  |-- dc:subject
```

Figure 4.4: Side by side comparison of the elements in (a) the Value Partition ODP in Rector (2005)(§Pattern 2–Variant 2) and (b) the Class As Property Value ODP in Noy (2005)(§Approach 4); as per Table 4.2.

the Value Partition ODP and in Definition 3.1 and 3.2 for the Class As Value Property ODP.

There are elements in one of the patterns that do not have a counterpart on the other pattern and thus, the "(Not applicable)" label in Table 4.2. This is the case for the element :$Value_i$ in the Value Partition ODP and the elements :$Interpretation_a$ and :$hasInterpretation_i$ (not intended to be a functional property) in the Class As Property Value ODP.

Figure 4.4 attempts to illustrate the comparison of the generic structure of the ontology patterns recapped in Table 4.2 with the specific examples of:

(a) Pattern 2–Variant 2 of the Value Partition ODP in Rector (2005); and

(b) Approach 4 of the Class As Property Value ODP in Noy (2005).

4.2.2 Conflation of Interpretation and Terminology

The alignment between these two patterns becomes more evident when, in the case of the Class As Property Value, the notion of :$Interpretation_a$ and :$Terminology_i$ conflates.

In this case, the elements in the generic structure of the Value Partition ODP (Figure 4.2) and the simplified version of the Class As Prop. Value ODP (Figure 3.8), align from a prototypical implementation standpoint but also from an ontological standpoint.

<table>
<tr><th>Value Partition</th><th>Class As Prop. Value</th><th>OWL Implementation</th></tr>
<tr><td>$:Value_i$</td><td rowspan="2">$:Terminology_i$</td><td>owl:Class (defined) (≡)(P)</td></tr>
<tr><td>(Not applicable)</td><td>owl:Class (primitive)</td></tr>
<tr><td rowspan="2">$:V_iPart_j$</td><td>$:T_iClass_j$</td><td>owl:Class (primitive)</td></tr>
<tr><td>(Not applicable)</td><td>owl:NamedIndividual (◊)</td></tr>
<tr><td colspan="2">$:TDC$</td><td>owl:Class (primitive)</td></tr>
<tr><td>$:V_iPart_jTDC$</td><td>$:T_iClass_jTDC$</td><td>owl:Class (defined) (≡)</td></tr>
<tr><td colspan="2" rowspan="2">$:SpecificTDC_x$</td><td>owl:NamedIndividual (◊)</td></tr>
<tr><td>owl:Class (primitive)</td></tr>
<tr><td>$:hasValue_i$</td><td>$:hasTerminology_i$</td><td>owl:ObjectProperty
owl:FunctionalProperty*</td></tr>
</table>

(∗) The Value Partition ODP requires the object property to be functional. The Class As Prop. Value ODP does not.

Table 4.3: Alignment of the Value Partition and the Class As Property Value (when $:Interpretation_a$ and $:Terminology_i$ conflate) ODPs.

Table 4.3 attempts to compare side by side the elements that participate on both patterns for the case under consideration; bringing together:

(a) the elements of the generic structure of the Value Partition ODP introduced in Figure 4.2 and Table 4.1; and

(b) the elements of the simplified generic structure of the Class As Property Value ODP introduced in Figure 3.8 and Table 3.2.

This alignment is again drawn upon the implementation, the relationship and the intended use of the elements in the specified version of the generic structure of the two patterns. Note the simplification of Table 4.3 with respect to Table 4.2. The disappearance of the $:Interpretation_a$ element, facilitates the alignment of mainly ($:Value_i$, $Terminology_i$) and ($:hasValue_i$, $:hasTerminology_i$), simplifying the mappings of the rest of elements dependent upon these.

To further characterize the alignment in Table 4.3, two additional figures are provided:

- Figure 4.5, which places side by side the specific version of the generic structure of the Value Partition and the simplified Class As Property Value.
- Figure 4.6, which places side by side the Value Partition example in Rector (2005) (§Pattern 2 - Variant 2) from Figure 4.3 and the Class As Property Value example in Noy (2005) (§Approach 4) with the variation in the :Zoo domain from Figure 3.7, so that $:Interpretation_a$ and $:Terminology_i$ conflates.

Note that both figures also illustrate the simplification in the alignment between the two patterns of Table 4.3 with respect to Table 4.2.

```
(Value Partition ODP)
owl:Thing
  |-- :Modifier
    |-- (≡)(P) :Value_i
      |-- [:V_iPart_j | (◊) :V_iPart_j]

  |-- :Self_standing_entity
    |-- :TargetDomainConcept (or :TDC)
      |-- (≡) :V_iPart_jTDC
      |
      |-- [:SpecificTDC_x |
           (◊) :SpecificTDC_x]

owl:topObjectProperty
  |-- :hasValue_i
```

```
(Class As Property Value ODP)
owl:Thing
  |
  |-- :Terminology_i
     |-- :T_iClass_j
       |-- :T_iClass_jClass_... etc.

  |-- :TargetDomainConcept (or :TDC)
     |-- (≡) :T_iClass_jTDC
       |-- :T_iClass_jClass_..TDC etc.
     |-- [:SpecificTDC_x |
          (◊) :SpecificTDC_x]

owl:topObjectProperty
  |-- :hasTerminology_i
```

Figure 4.5: Side by side comparison between the generic structure of the Value Partition ODP and the simplified version of the Class As Property Value ODP.

```
Value Partition
(Pattern 2 - Variant 2)
owl:Thing
  |-- (≡)(P) :Health_Value
    |-- :Good_health_value
    |
    |-- :Medium_health_value
    |-- :Poor_health_value
  |-- :Person
    |-- (≡) :Healthy_person
    |
    |-- (◊) :John

owl:topObjectProperty
  |-- :has_health_status
```

```
Class As Property Value
(Approach 4)
owl:Thing
  |-- :Animal
     |-- :Lion
       |-- :AfricanLion


  |-- :Zoo
     |-- (≡) :ZooWithLion
     |-- (≡) :ZooWithAfricanLion
     |-- (◊) :LondonZoo
     |-- (◊) :NewYorkZoo

owl:topObjectProperty
  |-- :hasAnimal
```

Figure 4.6: Alignment of the elements in (a) the Value Partition ODP in Rector (2005)(§ Pattern 2–Variant 2) and (b) the Class As Property Value ODP in Noy (2005)(§ Approach 4) in the :Zoo domain; with relation to Table 4.3.

4.2.3 Summary

This section summarizes the similarities and differences between the two ODPs under examination based on the information presented so far.

The similarities can be summarized as follows:

- Regarding the *OWL expressivity* of the underlying ontology model, both ODPs keep it within *OWL-DL*.
- Regarding the overall *generic structure*, both ODPs are applied to a target domain concept, a universe of discourse represented by the owl:Class :TDC that delimits

the scope of the patterns. This target domain concept contains two types of elements that represent, perform and implement the same functionality on both ODPs: (a) the defined subclasses of :TDC; and (b) the :$SpecificTDC_x$ elements.

- Regarding the *hierarchy of classes* for the version of the Value Partition ODP in which the values of the feature space (:V_iPart_j) are implemented as classes (i.e. Pattern 2–Variant 2 of Rector (2005)):
  - Both ODPs use a *hierarchy of classes* to provide *anonymous individuals* as *property values* for other concepts in the ontology model.
  - Regarding the *anonymous individuals* for the simplified version of the Class As Property Value in which the notion of interpretation and terminology conflate (i.e. Figure 3.7 derived from Approach 4 of Noy (2005)):
    * In both ODPs, the anonymous individuals are of the *same type* of the other individuals in the class. (i.e. in Pattern 2–Variant 2 of Rector (2005), all individuals of a given :V_iPart_j class represent a good health value or a medium health value or a poor health value and in Figure 3.7 derived from Approach 4 of Noy (2005), all individuals of a given :T_iClass_j represent an actual lion, an actual African lion, etc.).

The differences are recapped below:

- Regarding the *hierarchy of classes*:
  - In the Value Partition ODP the hierarchy of classes conforms to the definition of *value partition* and is used as a representation of features, attributes, or modifiers that describe other concepts in the ontology model.
  - In the Class As Property Value ODP the hierarchy of classes does not have to conform to the restriction of a *value partition* and is used as a terminology or subject index to annotate other domain concepts in the ontology model.
- Regarding the *object property*:
  - In the Value Partition ODP the object property is functional.
  - In the Class As Property Value ODP it does not have to be.
- Regarding the *anonymous individuals*, for the version of the Value Partition ODP in which the values of the feature space (:V_iPart_j) are implemented as classes and for the most generic version of the Class As Property Value in which the notion of interpretation and terminology are distinct:
  - In the Value Partition ODP the anonymous individuals are of the *same type* of the other individuals in the class (i.e. in Pattern 2–Variant 2 of Rector (2005), all individuals of a given :V_iPart_j class represent a good health value or a medium health value or a poor health value).

– In the Class As Property Value ODP the anonymous individuals are of *different type* of the other individuals in the class (i.e. the anonymous individuals in Approach 4 of Noy (2005) are subjects while others represent actual animals: an actual lion, an actual African lion, etc.).

Based on these similarities and differences between the two ODPs, it can be concluded that the version of the Value Partition ODP in which the values of the feature space (:V_iPart_j) are implemented as classes is a refinement or a specialization of the simplified version of the Class As Property Value ODP in which the notion of interpretation and terminology conflate. The specialization is given by: (a) the requirement of value partition to the terminology hierarchy of classes; and (b) the requirement of functional property to the object property featured in the pattern.

In other words, every instance of the version of the Value Partition ODP in which the values of the feature space (:V_iPart_j) are implemented as classes is also an instance of the simplified version of the Class As Property Value ODP in which the notion of interpretation and terminology conflate, although the vice versa is usually not true.

4.3 Conclusions

The generic structure of the ontology schema of the Value Partition ODP is introduced based on the examples of Rector (2005). This schema can accommodate the two versions of the pattern described by Rector (2005).

A prototypical implementation of the main elements (with the possible variations) that conform to the generic schema of the pattern is provided.

The generic structure presented, allows to perform a comparative analysis between the generic ontology schemata of the Value Partition ODP with respect to the Class As Property Value ODP described in Chapter 3.

The comparative analysis reveals a series of alignments between the two patterns that can be divided into two types: (a) the alignment from an implementation standpoint of the Value Partition and the most generic case of the Class As Property Value; and (b) the alignment from an implementation but also ontological standpoint of the Value Partition and the simplified case of the Class As Property Value, when in the latter, the notion of interpretation and terminology conflates.

Based on the differences and similarities exposed between the two patterns, the version of the Value Partition ODP in which the values of the feature space are implemented as classes, (i.e. Pattern 2–Variant 2 of Rector (2005)) can be seen as a refinement or more restrictive case of the simplified version of the Class As Property Value ODP where the notion of interpretation and terminology conflates, (i.e. the example in Figure 3.7 derived from Approach 4 of Noy (2005)).

Thus, an instance of the version of the Value Partition ODP in which the values of the feature space (:$V_i Part_j$) are implemented as classes is also an instance of the simplified version of the Class As Property Value ODP in which the notion of interpretation and terminology conflate, although this is usually not true in the opposite direction.

Chapter 5

Revisiting the Normalization ODP

5.1 Normalization ODP

The Normalization pattern is classified as a "Good Practice" ODP in the catalog of ODPs introduced in Egana-Aranguren (2005) Egana-Aranguren (2009) (available online[1]). It can be applied to any OWL DL ontology that consists of a polyhierarchy where some *semantic axes* can be pointed. Each of those axes will be a *module*. One of their most powerful features, is the ability of logical reasoners to link these independent ontology modules to allow them to be separately maintained, extended, and re-used.

The pattern also establishes a series of requirements that a normalized ontology should meet, some of which are summarized below:

- The essence for the normalization proposal is that the primitive skeleton of the domain ontology should consist of disjoint homogeneous trees (also referred to as *modules*) Rector (2003).
- Each primitive class that is part of the primitive skeleton should only have a primitive parent, and primitive sibling classes should be disjoint, creating the *modules* Egana-Aranguren (2005)(§ 4.3.2.1).
- This implies that for any two primitive concepts either one subsumes the other or they are disjoint. Assertion of multiple inheritance relations among primitive concepts are not allowed Rector (2003).
- Normalization allows exactly one unlabeled flavour of *is-kind-of* link corresponding to the links declared in the primitive skeleton. All others are inferred by the reasoner Rector (2003).

[1] http://odps.sourceforge.net/ (see § Normalization)

```
owl:Thing
   |-- :Module1
      |-- [:M1Elem1 | (◊) :M1Elem1]
      |-- [:M1Elem2 | (◊) :M1Elem2]
      |-- (... rest of elements of :Module1)
   |-- :Module2
      |-- [:M2Elem1 | (◊) :M2Elem1]
      |-- [:M2Elem2 | (◊) :M2Elem2]
      |-- (... rest of elements of :Module2)
   |-- (... rest of modules and their elements)
   |-- :TargetDomainConcept (or :TDC)
      |-- (≡) :M1Class1TDC
      |-- (≡) :M1Class2TDC
      |-- (≡) (... rest of defined classes based on :Module1)
      |-- (≡) :M2Class1TDC
      |-- (≡) :M2Class2TDC
      |-- (≡) (... rest of defined classes based on :Module2)
      |-- (≡) (... rest of defined classes based on subclasses of the rest of modules)
      |-- :SpecificTDC1
      |-- (◊) :SpecificTDC2
      |-- [:SpecificTDC3 | (◊) :SpecificTDC3]
      |-- (... rest of specific items from the :TDC to be represented and classified)

owl:topObjectProperty
   |-- :hasModule1
   |-- :hasModule2
   |-- (... rest of properties based on the rest of modules)
```

(≡) denotes a defined owl:Class.
(◊) denotes an owl:NamedIndividual.
[:a | (◊) :a] denotes :a can be either an owl:Class or an owl:NamedIndividual.

Figure 5.1: Generic structure of the Normalization ODP.

5.1.1 Structure and Elements

There are several examples of the generic structure of the Normalization ODP in the literature Egana-Aranguren (2005)(§ 4.3.2.1), Egana-Aranguren (2009)(§ 6.1.5, § A.13) and online[1]. Figure 5.1 presents the specific version of the generic structure that this paper will refer to hereafter, which preserves the required characteristics of the pattern.

Figure 5.2 depicts a further generalization of the structure in Figure 5.1 and introduces the following notation:

- :$Module_i$ denotes a primitive class that represents one of the modules or semantic axes identified.
- :M_iElem_j denotes a primitive owl:Class or an owl:NamedIndividual that represents a subset or a specific instance respectively, of the module class :$Module_i$.
- :$hasModule_i$ denotes an object property that links the elements :M_iElem_j of a module or semantic axis $Module_i$, to other elements in the ontology model.

```
owl:Thing
   |-- :Module_i
      |-- [:M_iElem_j | (◊) :M_iElem_j]
   |-- :TargetDomainConcept (or :TDC)
      |-- (≡) :M_iClass_jTDC
      |-- [:SpecificTDC_x | (◊) :SpecificTDC_x]

owl:topObjectProperty
   |-- :hasModule_i
```

Figure 5.2: Generic structure of the Normalization ODP.

- :TDC denotes a primitive class representing the domain concept being normalized (the scope and universe of discourse of the pattern). The elements of the class :TDC are involved in the poly-hierarchies that the Normalisation pattern aims to untangle and manage. The property :$hasModule_i$ links the elements of the class :TDC to the elements :M_iElem_j from the various modules or semantic axes :$Module_i$.

- :M_iClass_jTDC denotes a defined subclass of the target domain concept :TDC, defined by a relationship to a single element :M_iElem_j of the module or semantic axis class :$Module_i$, via the property :$hasModule_i$. Note there is a *one-to-one* relationship between a defined class :M_iClass_jTDC and the specific value :M_iElem_j that it derives from.

- :$SpecificTDC_x$ denotes an element from the target domain concept :TDC, that is described with one of the specific elements :M_iElem_j of the module or semantic axis class :$Module_i$, via the property :$hasModule_i$. Note that a given element :$SpecificTDC_x$ may be described with several elements :M_iElem_j from different modules or semantic axes :$Module_i$. An element :$SpecificTDC_x$ can be implemented as an owl:NamedIndividual or as an owl:Class depending on the required semantic in the ontology model.

The ontological relationship that exists between an element :$SpecificTDC_x$ and the multiple elements :M_iElem_j from the different modules :$Module_i$ is the underlying reason for the poly-hierarchies that the Normalisation ODP aims to decouple and manage.

Table 5.1 summarizes the elements of the generic structure of the Normalisation ODP in Figure 5.2 just introduced and their corresponding OWL implementation.

5.1.2 Implementation

One of the main features of the Normalization ODP is to enable a reasoner to maintain the subsumption relations between a class :$SpecificTDC_x$ and the various classes

Class As Prop. Value	**OWL Implementation**
:$Module_i$	owl:Class (primitive)
:M_iElem_j	owl:Class (primitive)
	owl:NamedIndividual (◊)
:TDC	owl:Class (primitive)
:M_iElem_jTDC	owl:Class (defined) (≡)
:$SpecificTDC_x$	owl:Class (primitive)
	owl:NamedIndividual (◊)
:$hasModule_i$	owl:ObjectProperty

Table 5.1: OWL Implementation of the Normalisation ODP.

:M_iClass_jTDC involved in its description. This feature is accomplished encoding the conditions of the subsumption relation as restrictions in the implementation of the classes :M_iClass_jTDC and :$SpecificTDC_x$.

Definition 5.1. The implementation of a generic defined class :M_iClass_jTDC is given as follows:

```
:M_iClass_jTDC
    rdf:type owl:Class ;
    rdfs:subClassOf :TDC ;
    owl:equivalentClass [ rdf:type owl:Restriction ;
                          owl:onProperty :hasModule_i ;
                          owl:someValuesFrom :M_iElem_j
                        ] .
```

Listing 5.1: Implementation of a generic defined class :M_iClass_jTDC

This implementation indicates that:

- A :M_iClass_jTDC class is equivalent to an anonymous class described by an existential property restriction.
- The restriction is on the object property :$hasModule_i$ associated to the module :$Module_i$ that subsumes the class :M_iElem_j.
- The filler of the restriction is the class :M_iElem_j linked to the definition of :M_iClass_jTDC.

Definition 5.2. The implementation of a generic class :$SpecificTDC_x$ is given as follows:

```
:SpecificTDC_x
    rdf:type owl:Class ;
    rdfs:subClassOf :TDC ,
                    [ rdf:type owl:Restriction ;
                      owl:onProperty :hasModule_i ;
```

```
                    owl:someValuesFrom :M_iElem_j
                  ] ,
                  [ ... rest of existential restrictions on property :hasModule_i
                        for every class :M_iElem_j that participates
                        in the description of :SpecificTDC_x
                  ] .
```

Listing 5.2: Implementation of a generic class :$SpecificTDC_x$

This representation indicates the following:

- A class :$SpecificTDC_x$ is subsumed by a variable number of anonymous classes. More specifically, one anonymous class for every class :M_iElem_j of every module :$Module_i$ that is linked to the description of :$SpecificTDC_x$. Every anonymous class is represented by an existential property restriction such as:
  - The restriction is on the object property :$hasModule_i$, associated to the module :$Module_i$ that subsumes the class :M_iElem_j.
  - The filler of the restriction is the class :M_iElem_j, linked to the description of :$SpecificTDC_x$.

This implementation of the classes :M_iClass_jTDC and :$SpecificTDC_x$ respectively, enable a reasoner to infer and maintain the subsumption relations between a given class :$SpecificTDC_x$ and the various classes :M_iClass_jTDC that it is related to.

Specific examples of the Normalization ODP in the literature Egana-Aranguren (2005)(§ 4.3.2.1), Egana-Aranguren (2009)(§ 6.1.5, § A.13) and online[1] demonstrate the features of the pattern in specific use case scenarios.

5.2 Alignment of the Normalization, Value Partition and Class As Prop. Value ODPs

This section analyzes the alignment among the Normalization ODP, the Value Partition ODP, and the simplified version of the Class As Property Value ODP, in which the elements :$Interpretation_a$ and :$Terminology_i$ conflate as described previously in Section 3.4.

The relationship between Normalization and Value Partitions is fairly evident and it is referred to in previous works Egana-Aranguren (2005). Perhaps more subtle it is the relationship to the simplified version of the Class As Property Value ODP. Once again, the generic structure of three ODPs is used to anchor the comparison, so that the results

Normalisation	**Value Partition**	**Class As Value**	**OWL Implementation**
$:Module_i$	$:Value_i$	$:Terminology_i$	owl:Class (defined) (≡)(P)
	(Not applicable)		owl:Class (primitive)
$:M_iElem_j$	$:V_iPart_j$	$:T_iClass_j$	owl:Class (primitive)
		(Not applicable)	owl:NamedIndividual (◊)
$:TDC$			owl:Class (primitive)
$:M_iElem_jTDC$	$:V_iPart_jTDC$	$:T_iClass_jTDC$	owl:Class (defined) (≡)
$:SpecificTDC_x$			owl:NamedIndividual (◊)
			owl:Class (primitive)
$:hasModule_i$	$:hasValue_i$	$:hasTerminology_i$	owl:ObjectProperty owl:FunctionalProperty *

(∗) The Value Partition ODP requires the object property to be functional. The Class As Property Value ODP and the Normalisation ODP does not.

Table 5.2: Alignment of the elements in the generic structure of the ODPs: (a) Normalization, (b) Value Partition; and (c) the simplified version of Class As Property Value where $:Interpretation_a$ and $:Terminology_i$ conflate.

can be extrapolated to any specific use case or instantiation of the patterns. Extending the comparative analysis to the three patterns, reveal further similarities and analogies.

Table 5.2 compares side by side the elements that participate in the three patterns under consideration. Table 5.2 extends the comparison captured in Table 4.3 adding to it the elements of the Normalisation ODP. The comparison is again drawn upon the implementation, the relationship and the intended use of the elements in the specified version of the generic structure of the three patterns, and it brings together the following:

(a) the elements of the generic structure of the Normalisation ODP introduced in Figure 5.2 and Table 5.1,

(b) the elements of the generic structure of the Value Partition ODP introduced in Figure 4.2 and Table 4.1; and

(c) the elements of the simplified generic structure of the Class As Property Value ODP introduced in Figure 3.8 and Table 3.2, in which the elements $:Interpretation_a$ and $:Terminology_i$ conflate.

To illustrate the alignments across the three patterns, two figures are put forward as examples:

- Figure 5.3, which places side by side the elements of the Normalisation ODP and the corresponding elements of the ontology model in the Value Partition ODP example of Pattern 2–Variant 2 by Rector (2005) as shown in Figure 4.3.

```
owl:Thing                                       (Norm ODP Generic Structure)
   |-- :Modifier
      |-- (≡)(P) :Health_Value                   (:Module1)
         |-- :Good_health_value                    (:M1Class1)
         |-- :Medium_health_value                  (:M1Class2)
         |-- :Poor_health_value                    (:M1Class3)
   |-- :Self_standing_entity
      |-- :Person                               (:TargetDomainConcept)
         |-- (≡) :Healthy_person                   (:M1Class1TDC)
         |-- (◊) :John                             (:SpecificTDC1)

owl:topObjectProperty
   |-- :has_health_status                       (:hasModule1)
```

Figure 5.3: Relationship between the Value Partition ODP in Rector (2005)(§Pattern 2–Variant 2) and the Normalization ODP.

```
owl:Thing                                       (Norm ODP Generic Structure)
   |-- :Animal                                  (:Module1)
      |-- :Lion                                    (:M1Class1)
         |-- :AfricanLion                            (:M1Class1Class1)
   |-- :Zoo                                     (:TargetDomainConcept)
      |-- (≡) :ZooWithLions                        (:M1Class1TDC)
         |-- (≡) :ZooWithAfricanLions                (:M1Class1Class1TDC)
      |-- (◊) :LondonZoo                           (:SpecificTDC1)
      |-- (◊) :NewYorkZoo                          (:SpecificTDC2)

owl:topObjectProperty
   |-- :hasAnimal                               (:hasModule1)
```

Figure 5.4: Relationship between the Class As Prop. Value ODP in Noy (2005)(§Approach 4) and the Normalization ODP.

- Figure 5.4, which places side by side the elements of the Normalisation ODP and the corresponding elements of the ontology model in the simplified version of the Class As Property Value ODP example based on Approach 4 by Noy (2005) as shown in Figure 3.7.

5.2.1 Summary

This section summarizes the similarities and differences between the three ODPs under examination based on the information presented so far.

The similarities can be summarized as follows:

- Regarding the *OWL expressivity* of the underlying ontology model, the three ODPs keep it within *OWL-DL*.
- Regarding the overall *generic structure*, the three ODPs are applied to a target domain concept, a universe of discourse represented by the owl:Class :*TDC* that

delimits the scope of the patterns. This target domain concept contains two types of elements that represent, perform and implement the same functionality on all three ODPs: (a) the defined subclasses of $:TDC$; and (b) the $:SpecificTDC_x$ elements.

- Regarding the *hierarchy of classes* for: (a) the version of Normalisation ODP in which the elements ($:M_iElem_j$) of a semantic axis are implemented as classes; and (b) the version of the Value Partition ODP in which the values of the feature space ($:V_iPart_j$) are implemented as classes (i.e. Pattern 2–Variant 2 of Rector (2005)):
  - All three ODPs use a *hierarchy of classes* to provide *anonymous individuals* as *property values* for other concepts in the ontology model.
  - Regarding the *anonymous individuals* for the simplified version of the Class As Property Value in which the notion of interpretation and terminology conflate (i.e. Figure 3.7 derived from Approach 4 of Noy (2005)):
    * In all three ODPs, the anonymous individuals are of the *same type* of the other individuals in the class ($:M_iElem_j$, $:V_iPart_j$, and $:T_iClass_j$ respectively).
- Regarding reasoning, all the benefits provided by the good-practice Normalisation ODP apply to the other two patterns. The subsumption relations that could lead to complex manually encoded poly-hierarchies, are maintained by the reasoner in all three ODPs.

The differences are recapped below:

- Regarding the *hierarchy of classes*, in the of Normalisation ODP, the requirements of the hierarchy of classes subsumed by $:Module_i$ fall between the requirements of the hierarchy of classes subsumed by $:Terminology_i$ and $:Value_i$ in the other two patterns respectively, in the sense that:
  - $:Module_i$ is more restrictive than $:Terminology_i$ because all primitive subclasses of the former are disjoint, while this is not a requirement in the latter.
  - $:Module_i$ is less restrive than $:Value_i$ because the subclasses of the former are not required to exhaust or cover $:Module_i$, while this is required in the latter as per the definition of *value partition*.
- Regarding the *object property*, in the Value Partition ODP the object property is functional, while in the Normalisation and Class As Property Value ODPs this is not mandatory.

5.3 Conclusions

The generic structure of the ontology schema of the Normalisation ODP is introduced based on the characterization of the pattern by Egana-Aranguren (2005), Egana-Aranguren (2009). A prototypical implementation of the generic structure main elements required by the pattern is provided.

The generic structure presented, allows one to perform a comparative analysis between the generic ontology schemata of the Normalisation ODP with respect to the Value Partition ODP described in Chapter 4 and the simplified version of the Class As Property Value ODP, in which the notion of interpretation and terminology conflates, as described in Chapter 3–Section 3.4.

The comparative analysis reveals a series of alignments and differences among the three patterns. The Normalisation ODP can be seen as a superset, one degree higher in the level of abstraction with respect to the other two patterns, where a module or semantic axis :$Module_i$ can be populated by a descriptive feature :$Value_i$ of the Value Partition ODP or a terminology :$Terminology_i$ of the Class As Property Value ODP.

Aligning a :$Module_i$ to a :$Terminology_i$ or to a :$Value_i$, the relationship, functionality and implementation among the rest of the elements in the generic ontology schema of the three patterns is analogous. Thus, an instantiation of the Normalisation ODP may as well be composed of an instantiation of the Value Partition ODP or the Class As Property Value ODP.

In other words:

(a) An instantiation of the VP ODP in which the elements :V_iPart_j are implemented as classes, is also an instantiation of the Normalisation ODP in which the elements :M_iElem_j are implemented as classes. However, the vice versa may not be true.

(b) An instantiation of the Normalisation ODP in which the elements :M_iElem_j are implemented as classes, is also an instantiation of the CPV ODP. However, the vice versa may not be true.

(c) From (a) and (b) it follows that an instantiation of the VP ODP in which the elements :V_iPart_j are implemented as classes, is also an instantiation of the CPV ODP. However, the vice versa may not be true.

The ODP examples presented here, demonstrates the three statements above. This can be easily tested by populating each statement with the applicable ODP examples involved. In essence, three patterns that are aimed at three different modelling scenarios, are ultimately implemented by a similar set of OWL idioms.

Chapter 6

Introducing the Faceted Classification ODP

6.1 Faceted Classification Scheme (FCS)

A basic overview of facet analysis and FCSs is given in Chapter 2–Section 2.4. This chapter focuses on the main features of a FCS involved in the comparative analysis to the Normalization ODP for a given domain of discourse.

Denton (2003)(§ 0), characterized a FCS for a given domain as follows: "a set of mutually exclusive and jointly exhaustive categories, each made by isolating one perspective on the items (a facet), that combine to completely describe all the objects in question, and which users can use, by searching and browsing, to find what they need".

However, in order to develop a FCS it is required to go through the process of Facet Analysis. Vickery describes Facet Analysis as: "The essence of facet analysis is the sorting of terms in a given field of knowledge into homogeneous, mutually exclusive facets, each derived from the parent universe by a single characteristic of division" Denton (2003)(§ 2.3).

The key to Facet Analysis and FCSs is the notion of *facet*. Spiteri (1998) simplified existing canons and principles used in established Universal FCSs in Library Science. One of such principles is defined by the author as follows: "The Principles of Homogeneity and Mutual Exclusivity state respectively that facets must be homogeneous and mutually exclusive, i.e., that the contents of any two facets cannot overlap, and that each facet must represent only one characteristic of division of the parent universe" .

In this sense, each facet can be designed separately and it models the domain of discourse from a distinct aspect. Each facet consists of a terminology, a finite set of terms that exhaust the facet. This set of terms is also referred to as *foci*.

There are numerous types of FCSs that vary in complexity. For example, FCSs that include several subject fields containing multiple facets and subfacets Vickery (2008)(§ 8, Fig. 1). However, the following section characterizes the elements of a simple generic FCS that will be used to anchor the discussion hereafter.

6.1.1 Structure and Elements

Definition 6.1. Elements of a simple generic Faceted Classification Scheme:

- Target Domain Concept (TDC).
- Facets: Facet1, Facet2, ..., rest of facets.
- Terms or foci (organized by facets):
  - Facet1: F1Term1, F1Term2, ..., rest of terms in Facet1.
  - Facet2: F2Term1, F2Term2, ..., rest of terms in Facet2.
  - ... rest of terms by facet.
- Set of items (from the TDC) to classify: Item1, Item2, ..., rest of items.

The following notation is introduced to refer to the elements of a generic FCS in Definition 6.1:

- TDC denotes the domain or universe of discourse. The domain-specific concept targeted by the FCS.
- $Facet_i$ denotes one of the facets of the FCS.
- F_iTerm_j denotes one of the terms of $Facet_i$.
- $Item_x$ denotes one the items from the domain of discourse to be classified.

Example 6.1. *Table 6.1 recaps the final FCS developed for the "Dishwasher Detergent" domain example in Denton (2003)(§ 2.4). The elements of the schema fit into the generic structure presented in Definition 6.1 where:*

- The TDC element is populated with the domain "Dishwasher Detergent".
- $Facet_i$ elements are populated with the facets: "Agent", "Form", "Brand Name", "Scent", "Effect On Agent", and "Special Property".
- F_iTerm_j elements are populated with the terms or foci listed below (grouped by facet):

Dish Detergent FCS	
Facet	**Term**
Agent	dishwasher, person
Form	gel, gelpac, liquid, powder, tablet
Brand Name	Cascade, Electrasol, Ivory, No Name, Palmolive, President's Choice, Sunlight
Scent	green apple, green tea, lavender, lemon, mandarin, ocean breeze, orange blossom, orchard fresh, passion flower, ruby red grapefruit, ylang ylang
Effect on Agent	aroma therapy (subdivisions: invigorating, relaxing)
Special Property	antibacterial

Table 6.1: "Dish Detergent" FCS example by Denton (2003)(§ 2.4).

- Agent: dishwasher, person.
- Form: gel, gelpac, liquid, powder, tablet.
- Brand Name: Cascade, Electrasol, Ivory, No Name, Palmolive, President's Choice, Sunlight.
- Scent: green apple, green tea, lavender, lemon, mandarin, ocean breeze, orange blossom, orchard fresh, passion flower, ruby red grapefruit, ylang ylang.
- Effect on Agent: aroma therapy (subdivisions: invigorating, relaxing).
- Special Property: antibacterial.

- $Item_x$ elements are populated in this case with two example items to classify:
  - "President's Choice Antibacterial Hand Soap and Dishwasher Liquid".
  - "Palmolive Aroma Therapy, Lavender and Ylang Ylang".

6.2 Alignment of a FCS to the Normalization ODP

A comparative analysis between the main characteristics of a FCS and the Normalization ODP presented in previous sections, indicates the existence of key similarities between the elements in the generic structures of both conceptual models.

One such key similarity lies in the notion of *facet* in FCSs and the notion of *module* (or *semantic axis*) in the Normalisation ODP. Both elements represent one perspective of the domain being modeled, a single characteristic of division, a single criterion of classification in their respective paradigms.

Another key similarity is linked to the requirement for facets in a FCS to be homogeneous and mutually exclusive and likewise the requirement of modules in the Normalization

ODP to be comprised of primitive classes arranged in a structure of disjoint homogeneous class trees.

These key similarities prompt to identify a mapping between the elements of both conceptual models that allows one to transform a FCS into a normalized ontology model. In this first approach, the mapping aims to keep the design choices of the resultant normalized ontology as simple and straight-forward as possible, without compromising any of the requirements and features of both FCSs and the normalization mechanism. This approach might not be suitable for converting all possible schemata into a normalized ontology but it is an attempt to provide an initial set of basic design guidelines. These guidelines can be extended hereafter to support more complex cases of FCSs.

The main principle is to represent each *facet* as a independent *module* or *semantic axis*. Following this principle makes the application of the Normalization ODP *almost* straight-forward. Moreover, the resultant ontology includes the representation of the multiple alternative classification criteria that were considered in the original FCS for the target domain concept.

Table 6.2 summarizes the alignment of the elements in the generic structure of both conceptual models. This alignment enables the conversion from a FCS to an OWL DL ontology by applying the Normalization ODP.

- The first column (leftmost), contains the elements of a generic FCS as introduced in Section 6.1–Definition 6.1.
- The second column contains the elements of the Normalization ODP generic structure as introduced in Section 5.1–Figure 5.2.
- The third column represents the selected OWL notation for the elements of a generic FCS in the context of the Normalization ODP generic structure.
- The forth column (rightmost), indicates the OWL implementation chosen for every element. The selection complies with the requirements of the normalization mechanism.

Based on the principle of representing each facet as a module, the underlying ideas behind the mappings in Table 6.2 can be outlined as follows:

- The target domain concept TDC represents the domain of discourse of both a FCS and the Normalization ODP. The primitive class :TDC fulfills that role in the normalized ontology.
- A facet $Facet_i$ from a generic FCS corresponds to a module :$Module_i$ in the Normalization ODP, therefore it becomes a primitive class :$Facet_i$ in the normalized ontology model.

Library Sc.	**Ontology Modeling**		
FCS	**Norm. ODP**	**FCS in Norm.**	**OWL Impl.**
TDC	:TDC		owl:Class (primitive)
$Facet_i$	:$Module_i$	:$Facet_i$	owl:Class (primitive)
	:$hasModule_i$	:$hasFacet_i$	owl:ObjectProperty
F_iTerm_j	:M_iElem_j	:F_iTerm_j	owl:Class (primitive)
			owl:NamedIndividual (◊)
	:M_iClass_jTDC	:F_iTerm_jTDC	owl:Class (defined) (≡)
$Item_x$	:$SpecificTDC_x$		owl:Class (primitive)
			owl:NamedIndividual (◊)

Table 6.2: Alignment of a Faceted Classification Scheme to the Normalization ODP

- A facet $Facet_i$ from a FCS also becomes an object property :$hasFacet_i$ in the normalized ontology, given that for every module :$Module_i$ in the Normalization ODP, there is an object property :$hasModule_i$.

- From the relationship between facet and module, it follows that a facet term F_iTerm_j from a FCS maps to a module element :M_iElem_j from the Normalization ODP. Both elements represent the same notion in their respective conceptual models. A subdivision, a refinement of the facet or module that they complement respectively. Therefore, a facet term F_iTerm_j from a FCS becomes an element :F_iTerm_j in the normalized ontology.

- A facet term F_iTerm_j from a FCS also produces a defined class :F_iTerm_jTDC in the normalized ontology, given that for every element :M_iElem_j in the Normalization ODP, there is a corresponding defined class :M_iClass_jTDC.

- Every item $Item_x$ to be classified in the FCS aligns to an element :$Specific_x$ that is automatically classified by a reasoner in the Normalization ODP. Therefore, every element $Item_x$ is represented as an element :$SpecificTDC_x$ in the normalized ontology.

The rest of this section details the characteristics of the resultant normalized ontology model that is obtained by applying the Normalization ODP to a generic FCS. The application of the pattern is driven by the alignments summarized in Table 6.2. The process is illustrated using the example of the "Dishwasher Detergent" FCS presented in Section 6.1.1–Example 6.1.

6.2.1 Structure and Elements

Figure 6.1 depicts the placement of the elements of a generic FCS into the generic structure of the Normalization ODP based on the structure of the pattern in Figure 5.2 and the corresponding mappings from Table 6.2.

```
owl:Thing
   |-- :Facet_i
      |-- :F_iTerm_j
   |-- :TargetDomainConcept (or :TDC)
      |-- (≡) :F_iTerm_jTDC
      |-- :SpecificTDC_x

owl:topObjectProperty
   |-- :hasFacet_i
```

Figure 6.1: Generic structure of the Faceted Classification Scheme ODP.

Figure 6.2 further illustrates the high-level picture of the main steps involved in the transformation, going from FCS to ontology model via Normalisation ODP as per the identified alignments between the two.

Example 6.2. *Now let us populate the generic ontology structure in Figure 6.1 with the specific elements of the "Dishwasher Detergent" FCS example. Figure 6.3 presents the overall normalized ontology class diagram obtained.*

It is important to note that the structure in Figure 6.3 includes axioms to comply with the requirement already stated of the Normalization ODP. That is, the skeleton of primitive classes consists of disjoint homogeneous tress where each primitive class only has a primitive parent, and primitive sibling classes are disjoint. This class structure create the modules or semantic axes of the Normalisation pattern. Such requirement complies as well with the FCS requirement of facets being homogeneous and mutually exclusive based on the alignments in Table 6.2.

Two additional figures are provided to support Example 6.2:

- Figure 6.4 can be seen as an instantiation of the process summarised by Figure 6.2 for the "Dishwasher Detergent" FCS example. It illustrates the placement of the elements of the "Dishwasher Detergent" FCS into the ontology schema that results from the application of the alignments between a FCS and the Normalisation ODP.

- Figure 6.5 focuses on one aspect of the transformation of the "Dishwasher Detergent" FCS example from Figure 6.4. It focuses on the relationship between the defined classes :F_iElem_jTDC, subsumed by the target domain concept class :TDC; and the corresponding elements :F_iElem_j from the various facet classes :$Facet_i$; via the applicable object property :$hasFacet_i$.

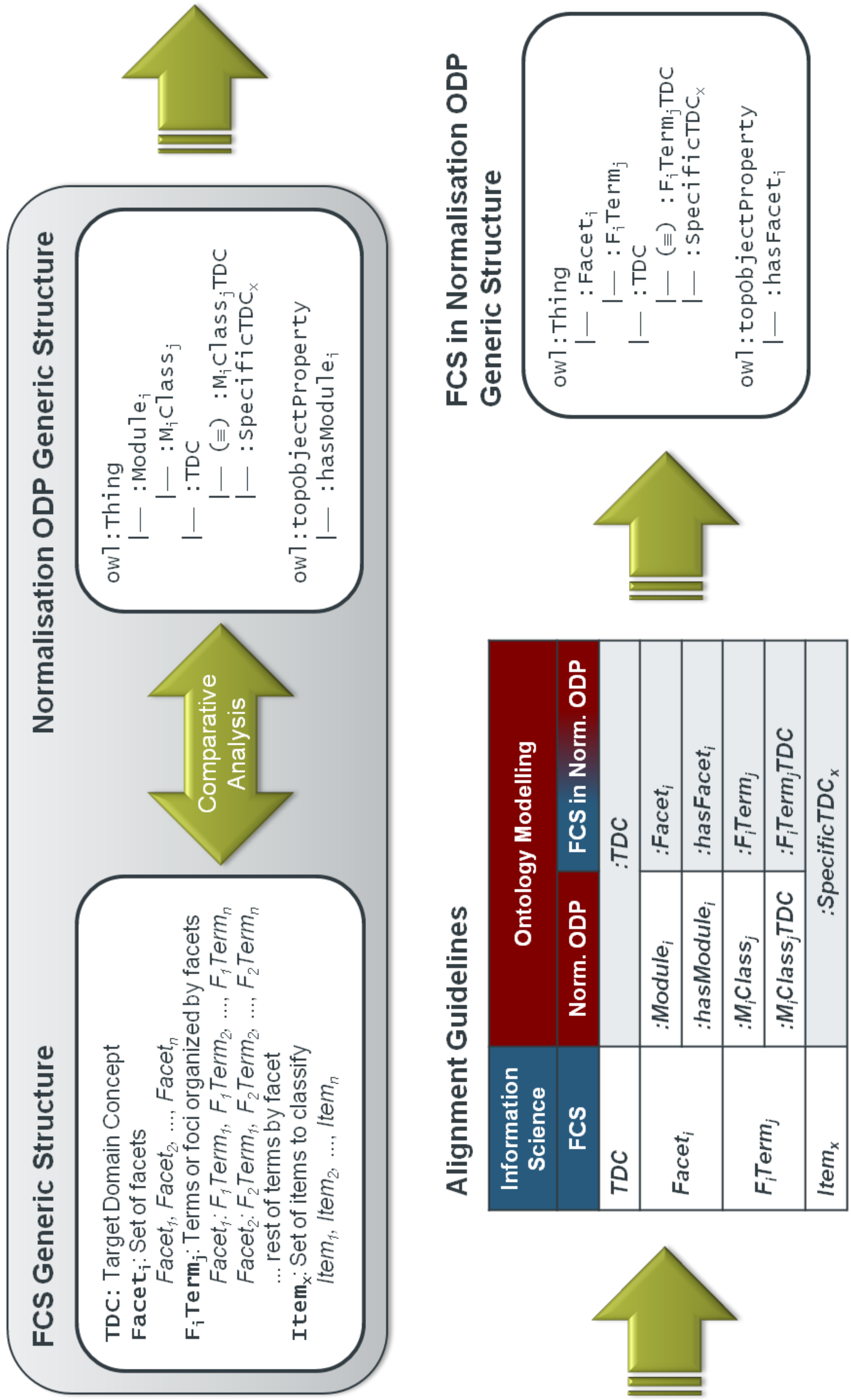


Figure 6.2: Placement of FCS elements into the Normalisation ODP generic structure.

```
owl:Thing
  |-- :Agent
     |-- :Person
     |-- :Dishwasher
  |-- :Form
     |-- :Gel
     |-- :Gelpac
     |-- (... rest of terms in the facet "Form")
  |-- :BrandName
     |-- :Cascade
     |-- :Electrasol
     |-- (... rest of terms in the facet "Brand Name")
  |-- :Scent
     |-- :GreenApple
     |-- :GreenTea
     |-- (... rest of terms in the facet "Scent")
  |-- :EffectOnAgent
     |-- :AromaTherapy
        |-- :Invigorating
        |-- :Relaxing
  |-- :SpecialProperty
     |-- :Antibacterial
  |-- :DishwasherDetergent (:TDC)
     |-- (≡) :ManualDishDetergent
     |-- (≡) :DishwasherDishDetergent
     |-- (≡) :GelDishDetergent
     |-- (≡) :GelpacDishDetergent
     |-- (≡) (... rest of subclasses for each term in the facet "Form")
     |-- (≡) :CascaseDishDetergent
     |-- (≡) :ElectrasolDishDetergent
     |-- (≡) (... rest of subclasses for each
                                        term in the facet "Brand Name")
     |-- (≡) :GreenAppleDishDetergent
     |-- (≡) :GreenTeaDishDetergent
     |-- (≡) (... rest of subclasses for each term in the facet "Scent")
     |-- (≡) :AromaTherapyDishDetergent
        |-- (≡) :InvigoratingDishDetergent
        |-- (≡) :RelaxingDishDetergent
     |-- (≡) :AntibacterialDishDetergent
     |-- :PresidentsPersonLiquidAntibacterial
     |-- :PalmoliveAromaTherapyLavenderYlangYlang
     |-- :SpecificDishDetergent3
     |-- (... rest of specific dish detergent classes
              :SpecificDishDetergent_x to classify)

owl:topObjectProperty
  |-- :hasAgent
  |-- :hasForm
  |-- :hasBrand
  |-- :hasScent
  |-- :hasEffectOnAgent
  |-- :hasSpecialProperty
```

Figure 6.3: Normalized ontology structure of the FCS for the "Dishwasher Detergent" domain concept.

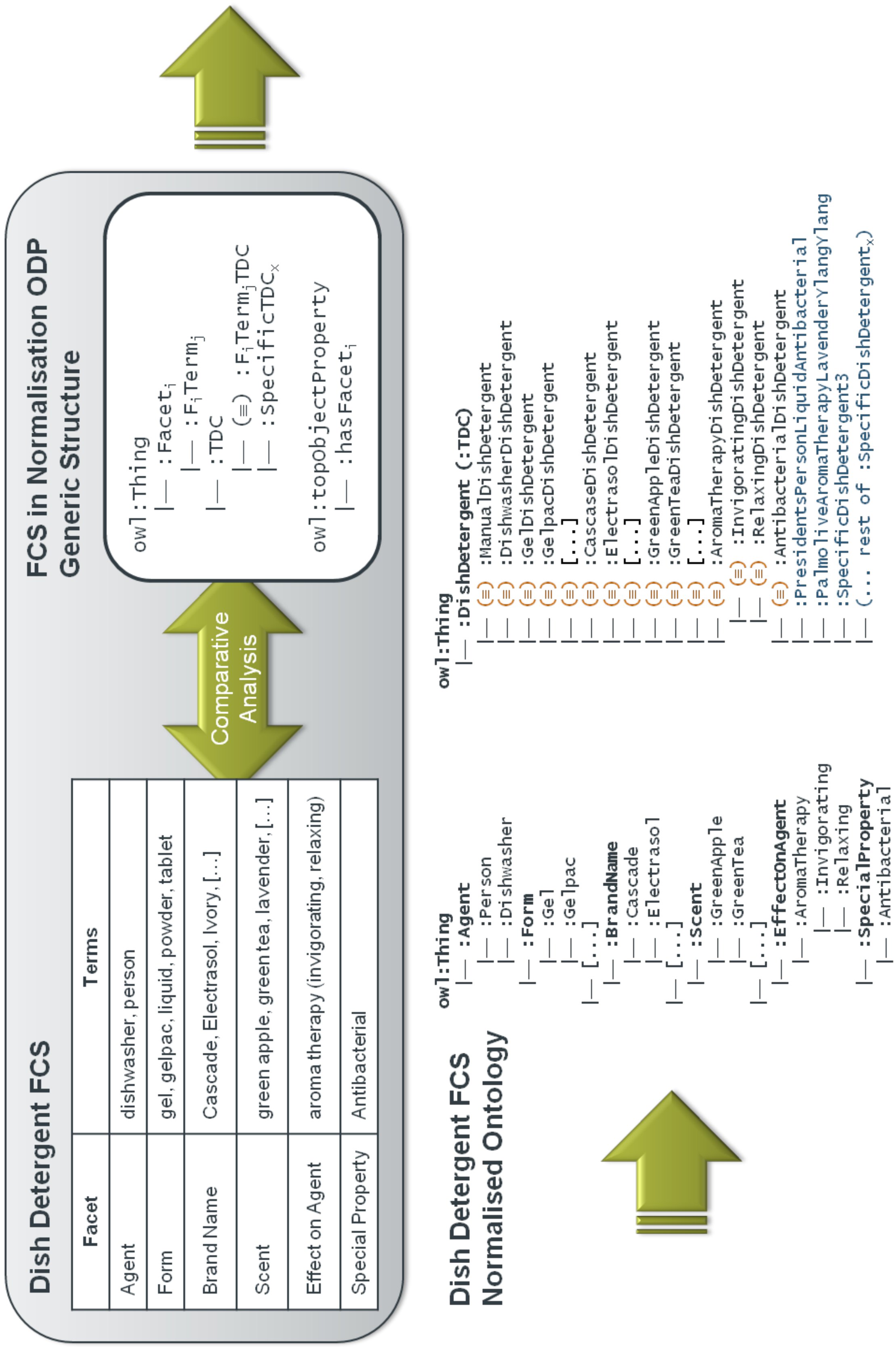


Figure 6.4: Placement of the Dish Detergent FCS elements into the Normalisation ODP generic structure (1 of 2).

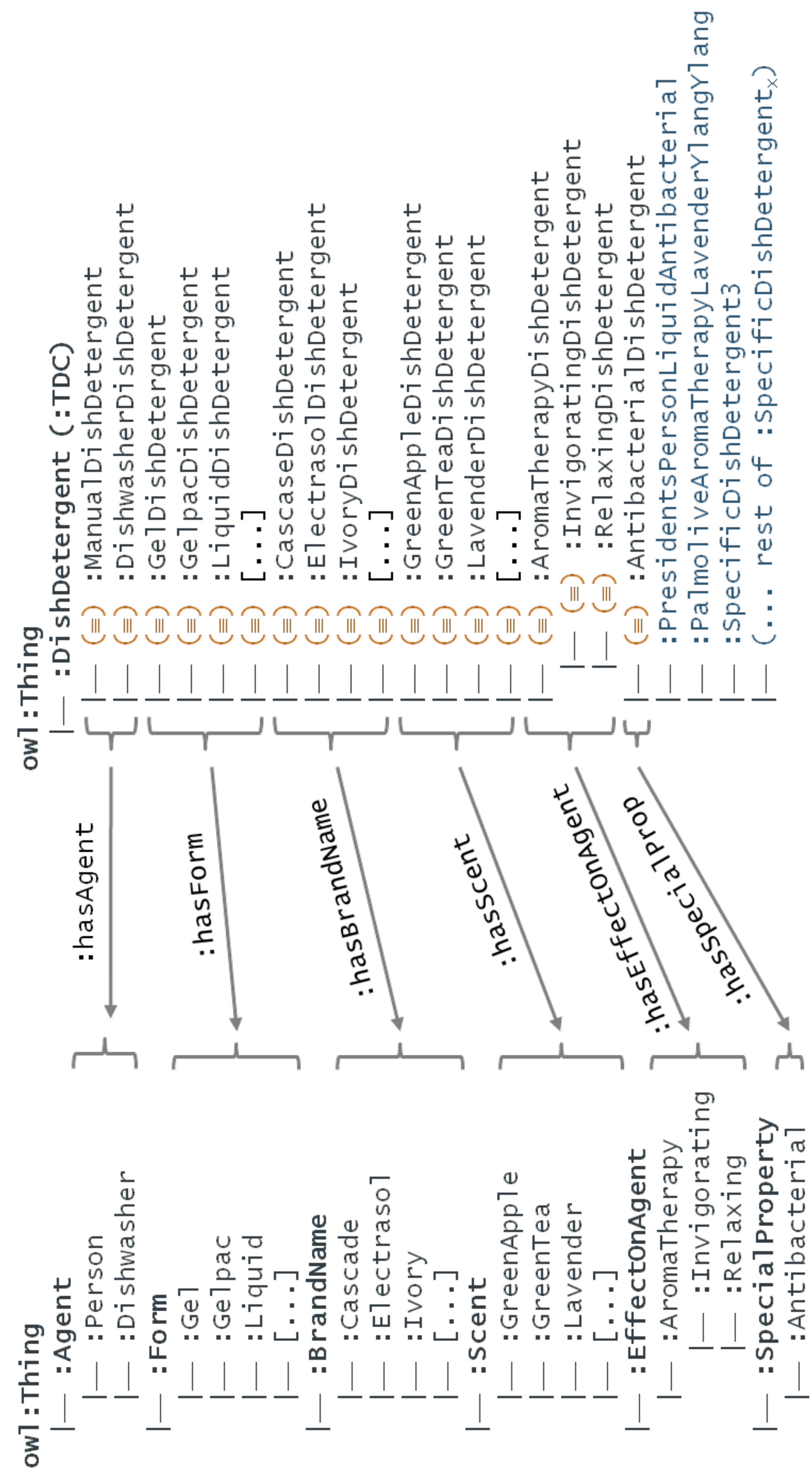


Figure 6.5: Placement of the Dish Detergent FCS elements into the Normalisation ODP generic structure (2 of 2).

6.2.2 Implementation

6.2.2.1 Defined Classes

The generic implementation of a defined class :F_iTerm_jTDC in terms of FCS elements is straight-forward based on the definition of :M_iClass_jTDC from Section 5.1–Definition 5.1 and the corresponding mappings from Table 6.2.

Example 6.3. *Let us illustrate the implementation of a defined class in the "Dishwasher Detergent" FCS example. Consider the facet "Agent" which contains the terms "Person" and "Dishwasher". From Table 6.2, these FCS elements fit into the normalized ontology as follows:*

- :$Facet_i$ is populated with :Agent.
- :$hasFacet_i$ is populated with :hasAgent.
- :F_iTerm_j is populated with :Person and :Dishwasher respectively.
- :F_iTerm_jTDC is populated with :ManualDishDetergent and :DishwasherDishDetergent respectively.

As an example, let us focus on the class :DishwasherDishDetergent. The implementation in the normalized ontology can be stated as follows:

```
:DishwasherDishDetergent
    rdf:type owl:Class ;
    rdfs:subClassOf :DishDetergent .
    owl:equivalentClass [ rdf:type owl:Restriction ;
                          owl:onProperty :hasAgent ;
                          owl:someValuesFrom :Dishwasher
                        ] ;
```

Listing 6.1: Implementation of the defined class :DishwasherDishDetergent

The implementation of the rest defined classes in the "Dishwasher Detergent" FCS shown in Figure 6.3 follows the same rationale.

6.2.2.2 Classification Elements

The generic implementation of a class :$SpecificTDC_x$ in terms of FCS elements is straight-forward following the implementation of :$SpecificTDC_x$ given in Section 5.1–Definition 5.2 and the applicable mappings from Table 6.2.

To illustrate the representation of a specific dishwasher detergent, let us reuse one of the classification examples presented in Denton (2003)(§ 2.4). The item "President's Choice

Antibacterial Hand Soap and Dishwashing Liquid" is classified in the cited reference as follows:

- Agent: person
- Form: liquid
- Brand Name: President's Choice
- Scent: (none)
- Effect on Agent: (none)
- Special Property: antibacterial

From Table 6.2, the description of the example detergent reveals the following mappings:

- :TDC is populated by :DishDetergent.
- :$SpecificTDC_x$ is populated by :PresidentsPersonLiquidAntibacterial.
- There are four existential restrictions. One per facet term involved in the description of the specific detergent at hand ("person", "liquid", "President's Choice", and "antibacterial"). Therefore, for each restriction:
  - :$hasFacet_i$ is populated with :hasAgent, :hasForm, :hasBrandName and :hasSpecialProperty respectively.
  - :F_iTerm_j is populated with :Person, :Liquid, :PresidentsChoice and :Antibacterial respectively.

Example 6.4. *The implementation of this particular detergent in the normalized ontology can be stated as follows:*

```
:PresidentsPersonLiquidAntibacterial
    rdf:type owl:Class ;
    rdfs:subClassOf :DishDetergent ,
                    [ rdf:type owl:Restriction ;
                      owl:onProperty :hasAgent ;
                      owl:someValuesFrom :Person
                    ] ,
                    [ rdf:type owl:Restriction ;
                      owl:onProperty :hasForm ;
                      owl:someValuesFrom :Liquid
                    ] ,
                    [ rdf:type owl:Restriction ;
                      owl:onProperty :hasBrandName ;
                      owl:someValuesFrom :PresidentsChoice
```

```
                    ] ,
                    [ rdf:type owl:Restriction ;
                      owl:onProperty :hasSpecialProperty ;
                      owl:someValuesFrom :Antibacterial
                    ] .
```

Listing 6.2: Implementation of the owl:Class :PresidentsPersonLiquidAntibacterial

This description makes explicit the relationship between the specific detergent class and every term of every facet that participates in the facet classification of the item. Moreover, it enables a reasoner to infer that :PresidentsPersonLiquidAntibacterial is a subclass of the following :$F_i Term_j TDC$ defined classes: :ManualDishDetergent, :LiquidDishDetergent, :PresidentsChoiceDishDetergent and :AntibacterialDishDetergent.

Figure 6.6 presents a portion of the inferred ontology class structure that results after applying a standard OWL reasoner to classify the specific element "President's Choice Antibacterial Hand Soap and Dishwashing Liquid" from Example 6.4, and the additional example element "Palmolive Aroma Therapy, Lavender and Ylang Ylang" also provided in Denton (2003)(§ 2.4).

All subsumption relations between the two specific detergent examples and the defined classes of the :TDC class :DishwasherDetergent, (in orange colour on Figure 6.6, or darker grey if printed in grey-scale), are automatically created and maintained by the OWL reasoner.

Figure 6.6 was generated using the OWLViz[1] tab of the Protege 4.x[2] open source Ontology Editor.

The Faceted Classification ODP was initially drafted in Rodriguez-Castro et al. (2010b). A version of the complete normalized ontology model for the "Dishwasher Detergent" FCS example is available online in RDF/XML format[3].

As already mentioned, in this first approach, the idea is to keep the transformation design guidelines simple, while complying with the requirements of a FCS and the Normalization ODP. Nonetheless, the following subsections discuss certain design choices to consider in the transformation process of a FCS into a normalized ontology.

6.2.3 Self-standing or Partitioning Concepts

The original normalization mechanism recommends to differentiate classes in the ontology model based on whether they represent a *domain* entity (also known as self-standing or independent entity) or a *modifier* entity (also known as refining or dependent entity)

[1] http://protegewiki.stanford.edu/wiki/OWLViz
[2] http://protege.stanford.edu/
[3] http://purl.org/net/project/enakting/ontology/detergent_fcs_norm

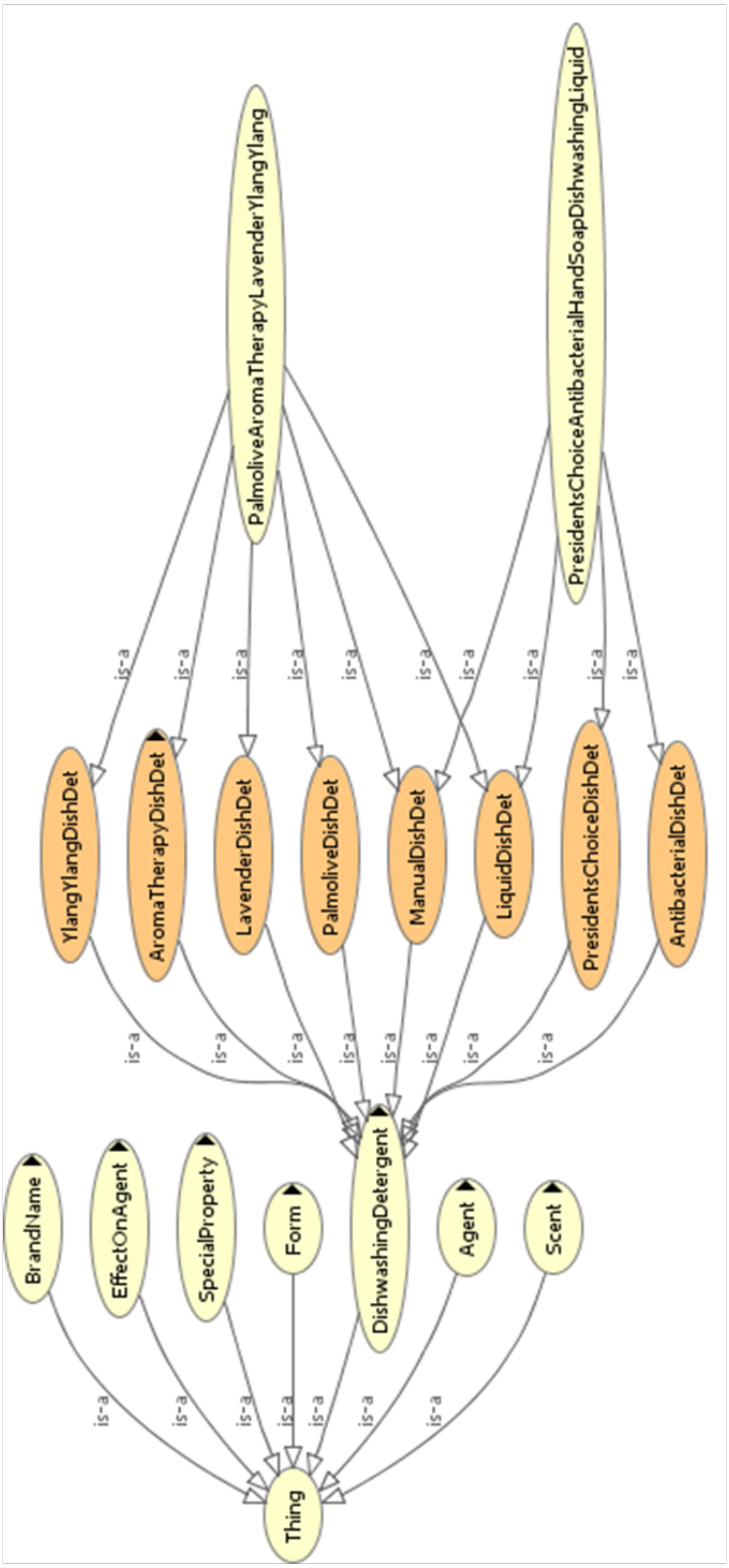


Figure 6.6: President's Choice Antibacterial Hand Soap and Dishwashing Liquid.

Rector (2003). The Normalization ODP derived from the original mechanism does not explicitly request this distinction Egana-Aranguren (2005)(§ 4.3.2.1), Egana-Aranguren (2009)(§ 6.1.5, § A.13). For simplicity, the proposed transformation guidelines considered all entities to be *domain* entities.

6.2.4 Domain and Range of Object Properties

The Normalization ODP does not prescribe the domain and range of object properties in the pattern. It only requires that domain and range property restrictions do not introduce overlap between primitive concepts that are intended to be disjoint. This scenario can take place when the domain or range of a property is set to more than one class which results in the intersection of the classes involved. Based on the definition of the object property :$hasFacet_i$, the natural choice of domain and range would be the target domain concept class :TDC and the corresponding facet class :$Facet_i$ respectively. That is:

```
:hasFacet_i rdf:type owl:ObjectProperty ;
          rdfs:domain :TDC ;
          rdfs:range :Facet_i .
```

6.2.5 Mutual Exclusion of Facets

There is a characteristic of a FCS system that could lead to some confusion. A FCS requires its facets and facet terms to be mutually exclusive in the conceptual model of the scheme. However, it allows to use multiple terms from the same facet to classify an item from its domain of discourse. In library classification, such items are referred to as *compound subjects* and ultimately they are the main motivation for faceted classification schemes.

This characteristic is illustrated in the classification example of the detergent "Palmolive Aroma Therapy, Lavender and Ylang Ylang" given in Denton (2003)(§ 2.4) using two terms of the same facet to classify the item (Scent: lavender, ylang ylang).

The feature of *functional* property provided by OWL allows one to capture this behavior in an ontology model given that existential property restrictions lead to *unsatisfiability* if a functional property is inferred to have two or more disjoint values.

In terms of our ontology model:

(a) The primitive classes :Lavender and :YlangYlang, subclasses of :Scent, are disjoint to comply with FCS and Normalization requirements; and

(b) As per Definition 5.2, the representation of the class :PalmoliveAromaTherapyLavenderYlangYlang includes two existential restrictions on property :hasScent over the classes :Lavender and :YlangYlang respectively.

Under these conditions, if :hasScent is a functional property, the class :PalmoliveAromaTherapyLavenderYlangYlang would be inferred to be unsatisfiable.

6.3 Conclusions

The generic structure of a simple Faceted Classification Scheme is introduced based on a simplified characterization of this library resource for a specific domain by Spiteri (1998) and Denton (2003). The generic structure introduced, allows one to perform a comparative analysis between the generic schemata of a FCS with respect to the generic structure of the Normalisation ODP presented in Chapter 5–Section 5.1.1.

The comparative analysis reveals a series of alignments between these two knowledge resources, that allows the transformation of a given FCS into a normalised OWL DL ontology model, following a consistent and systematic approach. The key alignment of a *facet* in a FCS to a :$Module_i$ (semantic axis) in the Normalisation ODP, determines the mapping, functionality and implementation of the rest of the elements in the generic structure of the two conceptual paradigms. An existing FCS example in the domain of "Dishwasher Detergent" is used to illustrate the main steps of our conversion procedure.

The overall transformation procedure, referred to as the Faceted Classification ODP and summarised in Rodriguez-Castro et al. (2010a), can be classified as a Re-engineering ODP according to the classification of patterns in Presutti et al. (2008)(§ 2.2.2). This is due to the fact that it provides a transformation of a non-ontological resource (a FCS) into an ontological one (an normalised OWL DL ontology). The Faceted Classification ODP is put forward as a partial solution to *Research Question 1* of Section 1.4.

Some of the shortcomings that can be argued with the Faceted Classification ODP approach, deals with support for more sophisticated and complex structures of FCSs and the need of an OWL DL reasoner to fully benefit from the advantages of the underlying Normalisation ODP.

Chapter 7

Evaluating the "Pizza" Domain Concept in the Protege Tutorial

Horridge et al. (2009), authors of the practical guide to the Protege free open source ontology editor, use the domain concept of "Pizza" to familiarise readers with the features of the editor and how they link to the W3C OWL specification.

The tutorial starts with an empty ontology model and as it makes progress, it builds an ontology that represents plenty of different types of pizzas based on some key characteristics such as the country of origin, the toppings used or the type of pizza base.

Looking at this representation of "Pizza" with the modeling problem subject of this research in mind, these key characteristics of pizza could be seen as multiple classification criteria of the concept. The Protege tutorial represents these classification criteria implicitly. Part of the aim of this evaluation section, is to show how bringing the representation of the multiple classification criteria of "Pizza" to the forefront of the ontology building process, can assist such process by limiting the opportunity for ad-hoc decisions.

The "Pizza" ontology built throughout the Protege tutorial, is based on an already existing ontology model available online[1]. This is the model that will be used to carry out this evaluation.

The "Pizza" ontology model makes use of the ODPs revisited throughout this research. The Protege tutorial covers explicitly the Value Partition design pattern Horridge et al. (2009)(§4.14) and it does makes reference in a fairly broad sense, to the Normalisation design pattern as an *ontology normalisation technique* Horridge et al. (2009)(§4.11).

Yet, a further examination of the elements of the "Pizza" ontology, reveals a more concealed use of the Class As Property Value pattern (not explicitly referred to) and at a larger scale, of the Normalisation pattern in the terms covered in Chapter 3 and

[1] `http://www.co-ode.org/ontologies/pizza/pizza.owl`

Chapter 5 respectively. For example, the "Pizza" ontology is built according to one of the key principles of Normalisation, which states that the class hierarchy of the asserted ontology model should form a simple tree. That is, classes do not have more than one parent class (single inheritance).

7.1 Structure of the "Pizza" Ontology

The procedure to examine the "Pizza" ontology model is based on a reverse engineering approach. It looks at the main element in the model, "Pizza", or :TDC in terms of the Normalisation ODP generic structure. From there it identifies the defined classes subsumed by "Pizza" and observes to which other elements in the ontology these defined classes are linked to. At that point is possible to realise if the elements linked to the various defined classes of "Pizza" are organised into separate modules as prescribed by the Normalisation ODP rationale. Up to this point the elements :TDC:, :M_iClass_jTDC, :M_iElem_j, and :$Module_i$ have been identified. A review of the properties in the ontology should also indicate if they align with a particular module as required by a generic :$hasModule_i$ property. Going back to the class "Pizza", the primitive classes subsumed by "Pizza" are examined to asses if they align to the generic element :$SpecificTDC_x$.

The sections below detail the mapping of all the elements in the generic structure of the Normalisation ODP to elements of the example "Pizza" ontology model.

7.1.1 Modules

A careful walk through the class hierarchy of the "Pizza" ontology indicates the existence of several *modules* in the context of the Normalization ODP. In fact, four modules can be identified. Using the notation of the Normalisation ODP generic structure in Figure 5.2, the class names of the four modules (:$Module_1$, :$Module_2$, :$Module_3$, and :$Module_4$) are respectively: :$Country$, :$PizzaBase$, :$PizzaTopping$, and :$Spiciness$. These four modules are shown in Figure 7.1, 7.2, 7.3, and 7.4 respectively.

The four figures depict the class structure of each module in the "Pizza" ontology on the left side and the element of the Normalisation ODP generic structure that corresponds to each element of the class structure on the right. The alignment of the Normalisation ODP to the Value Partition and Class As Property Value ODPs identified in Table 5.2, together with the intended use and implementation of these four modules in the "Pizza" ontology, indicate that:

- :$Module_1$ (:$Country$) does not align to any of the elements of the Value Partition or Class As Property Value ODPs, given that on one hand, the elements involved (the individuals :M_1Elem_j) do not form a partition and on the other, these are

owl:Thing	(*Normalization ODP Generic Structure*)
|-- :DomainConcept	
|-- :Country	$(:Module_1)$
|-- (◊) :America	$(:M_1Elem_1)$
|-- (◊) :England	$(:M_1Elem_2)$
|-- (◊) :France	$(:M_1Elem_3)$
|-- (◊) :Germany	$(:M_1Elem_4)$
|-- (◊) :Italy	$(:M_1Elem_5)$

Figure 7.1: The Country of "Pizza".

owl:Thing	(*Normalization ODP Generic Structure*)
|-- :DomainConcept	
|-- :Food	
|-- :PizzaBase	$(:Module_2)$
|-- :DeepPanBase	$(:M_2Elem_1)$
|-- :ThinkAndCrispyBase	$(:M_2Elem_2)$

Figure 7.2: The Base of "Pizza".

individuals being used as property values, not classes. The individuals of the :*Country* class are used as values for the object property :$hasCountryOfOrigin$.

- :$Module_2$ (:*PizzaBase*) does align to the element :$Terminology_i$ of the Class As Property Value ODP generic structure, given that the subclasses of :*PizzaBase* are used as values for the object property :*hasBase*.
- :$Module_3$ (:*PizzaTopping*) also aligns to the element :$Terminology_i$ of the Class As Property Value ODP, given that the subclasses of :*PizzaTopping* are used as values for the object property :*hasTopping*.
- :$Module_4$ (:*Spiciness*) is explicitly presented as a Value Partition and therefore aligns to the element :$Value_i$ of the Value Partition ODP. The object property used to implement the pattern is :*hasSpiciness*.

These modules correspond to what the normalisation mechanism refers to as *semantic axes* and to what this research has aligned to the notion of *facet* to represent the various *classification criteria* that the domain concept under scrutiny is subject to.

7.1.2 Object Properties

Figure 7.5 summarizes the object properties in the "Pizza" ontology including the domain and range (if any) of each property and again, on the far right column the element of the Normalisation ODP generic structure that the property maps into.

Note how the one-to-one relation in the Normalisation ODP generic structure between the class :$Module_i$ and the object property :$hasModule_i$ holds in the "Pizza" ontology

`owl:Thing`	($Normalization\ ODP\ Generic\ Structure$)
`   |-- :DomainConcept`	
`      |-- :Food`	
`         |-- :PizzaTopping`	$(:Module_3)$
`           |-- :CheeseTopping`	$(:M_3Elem_1)$
`              |-- :FourCheesesTopping`	$(:M_3Elem_1Elem_1)$
`              |-- :GoatsCheeseTopping`	$(:M_3Elem_1Elem_2)$
`              |-- :GorgonzolaTopping`	$(:M_3Elem_1Elem_3)$
`              |-- :MozzarellaTopping`	$(:M_3Elem_1Elem_4)$
`              |-- :ParmesanTopping`	$(:M_3Elem_1Elem_5)$
`           |-- :FishTopping`	$(:M_3Elem_2)$
`              |-- :AnchoviesTopping`	$(:M_3Elem_2Elem_1)$
`              |-- :MixedSeafoodTopping`	$(:M_3Elem_2Elem_2)$
`              |-- :PrawnsTopping`	$(:M_3Elem_2Elem_3)$
`           |-- :FruitTopping`	$(:M_3Elem_3)$
`              |-- :SultanaTopping`	$(:M_3Elem_3Elem_1)$
`           |-- :HerbSpiceTopping`	$(:M_3Elem_4)$
`              |-- :CajunSpiceTopping`	$(:M_3Elem_4Elem_1)$
`              |-- :RosemaryTopping`	$(:M_3Elem_4Elem_1)$
`           |-- :MeatTopping`	$(:M_3Elem_5)$
`              |-- :ChickenTopping`	$(:M_3Elem_5Elem_1)$
`              |-- :HamTopping`	$(:M_3Elem_5Elem_2)$
`                 |-- :ParmaHamTopping`	$(:M_3Elem_5Elem_2Elem_1)$
`              |-- :HotSpicedBeefTopping`	$(:M_3Elem_5Elem_3)$
`              |-- :PeperoniSausageTopping`	$(:M_3Elem_5Elem_4)$
`           |-- :NutTopping`	$(:M_3Elem_6)$
`              |-- :PineKernels`	$(:M_3Elem_6Elem_1)$
`           |-- :SauceTopping`	$(:M_3Elem_7)$
`              |-- :TobascoPepperSauce`	$(:M_3Elem_7Elem_1)$
`           |-- (≡) :SpicyTopping`	$(:M_3Elem_8)$
`           |-- :VegetableTopping`	$(:M_3Elem_9)$
`              |-- :ArtichokeTopping`	$(:M_3Elem_9Elem_1)$
`              |-- :AsparagusTopping`	$(:M_3Elem_9Elem_2)$
`              |-- :CaperTopping`	$(:M_3Elem_9Elem_3)$
`              |-- :CheesyVegetableTopping`	$(:M_3Elem_9Elem_4)$
`              |-- :GarlicTopping`	$(:M_3Elem_9Elem_5)$
`              |-- :LeekTopping`	$(:M_3Elem_9Elem_6)$
`              |-- :MushroomTopping`	$(:M_3Elem_9Elem_7)$
`              |-- :OliveTopping`	$(:M_3Elem_9Elem_8)$
`              |-- :OnionTopping`	$(:M_3Elem_9Elem_9)$
`                 |-- :RedOnionTopping`	$(:M_3Elem_9Elem_9Elem_1)$
`              |-- :PepperTopping`	$(:M_3Elem_9Elem_{10})$
`                 |-- :GreenPepperTopping`	$(:M_3Elem_9Elem_{10}Elem_1)$
`                    |-- :HotGreenPepperTopping`	$(:M_3Cl_9Cl_{10}Cl_1Cl_1)$
`                 |-- :JalapenoPepperTopping`	$(:M_3Elem_9Elem_{10}Elem_2)$
`                 |-- :PeperonataTopping`	$(:M_3Elem_9Elem_{10}Elem_3)$
`                 |-- :SweetPepperTopping`	$(:M_3Elem_9Elem_{10}Elem_4)$
`              |-- :PetitPoisTopping`	$(:M_3Elem_9Elem_{11})$
`              |-- :RocketTopping`	$(:M_3Elem_9Elem_{12})$
`              |-- :SpinachTopping`	$(:M_3Elem_9Elem_{13})$
`              |-- :TomatoTopping`	$(:M_3Elem_9Elem_{14})$
`                 |-- :SlicedTomatoTopping`	$(:M_3Elem_9Elem_{14}Elem_1)$
`                 |-- :SundriedTomatoTopping`	$(:M_3Elem_9Elem_{14}Elem_2)$
`           |-- (≡) :VegetarianTopping`	$(:M_3Elem_{10})$

Figure 7.3: The Topping of "Pizza".

```
owl:Thing                                      (Normalization ODP Generic Structure)
  |-- (≡)(P) :ValuePartition
     |-- (≡) :Spiciness                        (:Module4)
        |-- :Mild                                 (:M4Elem1)
        |-- :Medium                               (:M4Elem2)
        |-- :Hot                                  (:M4Elem3)
```

Figure 7.4: The Spiciness of "Pizza".

```
owl:topObjectProperty        (rdfs : domain)   (rdfs : range)    (Normalization ODP)
   |-- :hasCountryOfOrigin                                       (:hasModule1)
   |-- :hasIngredient        :Food             :Food
      |-- :hasBase           :Pizza            :PizzaBase        (:hasModule2)
      |-- :hasTopping        :Pizza            :PizzaTopping     (:hasModule3)
   |-- :hasSpiciness                           :Spiciness        (:hasModule4)
   |-- :isIngredientOf       :Food             :Food
      |-- :isBaseOf          :PizzaBase        :Pizza
      |-- :isToppingOf       :PizzaTopping     :Pizza
```

Figure 7.5: Properties of the ontology model for "Pizza".

between the classes :*Country*, :*PizzaBase*, :*PizzaTopping*, :*Spiciness* and the object properties :$hasCountryOfOrigin$, :$hasBase$, :$hasTopping$, :$hasSpiciness$ respectively.

7.1.3 Target Domain Concept

Another key element of the Normalisation ODP is the class that represents the domain or universe of discourse being normalised. What in the generic structure of the pattern in Figure 5.2 is labeled as target domain concept, :*TargetDomainConcept* or :TDC. In the "Pizza" ontology the :TDC element is represented by the class :$Pizza$.

Figure 7.6 captures the full class structure of the :$Pizza$ class in the ontology. The right side of Figure 7.6 indicates the element of the Normalisation ODP generic structure associated to each element of the :$Pizza$ class hierarchy. There are clearly two sections of the :$Pizza$ class structure that populates the two elements of the :TDC class structure, : M_iClass_jTDC and :$SpecificTDC_x$. These are respectively, all the defined subclasses of :$Pizza$, and all the primitive subclasses of :$NamedPizza$. More about these two elements in the following sections.

At this point, all the elements in the Normalisation ODP generic structure have been identified in the "Pizza" ontology. Figures 7.1, 7.2, 7.3, 7.4, 7.5 and 7.6 indicates the elements of the "Pizza" ontology that populate each of the elements in the Normalisation ODP generic structure (:$Module_i$, :M_iElem_j, :$hasModule_i$, :TDC, :M_iClass_jTDC, :$SpecificTDC_x$). The alignment between the two models is further supported by the OWL implementation of key elements of the "Pizza" ontology.

```
owl:Thing                                          (Normalization ODP Generic Structure)
   |-- :DomainConcept
      |-- :Food
         |-- :Pizza                                (:TargetDomainConcept)
            |-- (≡) :CheeseyPizza                     (:M_3Class_1TDC)
            |-- (≡) :InterestingPizza                 (related to multiple :M_3Elem_j)
            |-- (≡) :MeatyPizza                       (:M_3Class_5TDC)
            |-- (≡) :NonVegetarianPizza               (related to multiple :M_iElem_j)
            |-- (≡) :RealItalianPizza                 (:M_1Class_5TDC)
            |-- (≡) :SpicyPizza                       (:M_3Class_8TDC)
            |-- (≡) :SpicyPizzaEquivalent             (:M_4Class_3TDC)
            |-- (≡) :ThinAndCrispyPizza               (:M_2Class_2TDC)
            |-- (≡) :VegetarianPizza                  (related to multiple :M_iElem_j)
            |-- (≡) :VegetarianPizzaEquivalent1       (:M_3Class_10TDC)
            |-- (≡) :VegetarianPizzaEquivalent2       (related to multiple :M_iElem_j)
            |-- :NamedPizza
               |-- :American                             (:SpecificTDC_1)
               |-- :AmericanHot                          (:SpecificTDC_2)
               |-- :Cajun                                (:SpecificTDC_3)
               |-- :Capricciosa                          (:SpecificTDC_4)
               |-- :Caprina                              (:SpecificTDC_5)
               |-- :Fiorentina                           (:SpecificTDC_6)
               |-- :FourSeasons                          (:SpecificTDC_7)
               |-- :FruttiDiMare                         (:SpecificTDC_8)
               |-- :Giardiniera                          (:SpecificTDC_9)
               |-- :LaReine                              (:SpecificTDC_10)
               |-- :Margherita                           (:SpecificTDC_11)
               |-- :Mushroom                             (:SpecificTDC_12)
               |-- :Napoletana                           (:SpecificTDC_13)
               |-- :Parmense                             (:SpecificTDC_14)
               |-- :PolloAdAstra                         (:SpecificTDC_15)
               |-- :PrinceCarlo                          (:SpecificTDC_16)
               |-- :QuattroFormaggi                      (:SpecificTDC_17)
               |-- :Rosa                                 (:SpecificTDC_18)
               |-- :Siciliana                            (:SpecificTDC_19)
               |-- :SloppyGiuseppe                       (:SpecificTDC_20)
               |-- :Soho                                 (:SpecificTDC_21)
               |-- :UnclosedPizza                        (:SpecificTDC_22)
               |-- :Veneziana                            (:SpecificTDC_23)
```

Figure 7.6: The “Pizza” domain concept.

7.1.3.1 Defined Classes

In the process of exploring the existing alignment of the “Pizza” ontology model to the Normalization ODP generic structure, two groups of defined classes subsumed by the target domain concept :Pizza can be distinguished. One group includes the defined classes that conform to the implementation of a generic element :M_iClass_jTDC in the Normalisation pattern given in Definition 5.1 and the other group includes the defined classes that do not conform to such implementation. The former will be referred to as “Single Defined Classes” and the latter as “Compound Defined Classes”. Both groups are presented in the sections below.

```
:CheeseyPizza

    rdf:type owl:Class ;

    rdfs:label "PizzaComQueijo"@pt ;

    owl:equivalentClass [ rdf:type owl:Class ;
                          owl:intersectionOf ( :Pizza
                                               [ rdf:type owl:Restriction ;
                                                 owl:onProperty :hasTopping ;
                                                 owl:someValuesFrom :CheeseTopping
                                               ]
                                             )
                        ] ;

    rdfs:comment "Any pizza that has at least 1 cheese topping."@en .
```

Listing 7.1: Implementation of the defined class :$CheeseyPizza$

***Single* Defined Classes** The implementation of some defined subclasses of :$Pizza$ mirror the implementation of the generic defined class :M_iClass_jTDC in the Normalisation ODP.

For example, the implementation of :$CheesyPizza$ (see Listing 7.1), and its use of the property :$hasTopping$ and class :$CheeseTopping$, mirrors the implementation of a class :M_iClass_jTDC and its use of the property :$hasModule_i$ and class :M_iElem_j respectively given in Definition 5.1. The one-to-one relationship between :$CheesyPizza$ and :$CheeseTopping$ via the property :$hasTopping$ holds as in the case of :M_iClass_jTDC, :M_iElem_j and :$hasModule_i$.

The rest of defined subclasses of :$Pizza$ whose implementation fit into the implementation of a normalised :M_iClass_jTDC class can be seen in Figure 7.6 with the specific :M_iClass_jTDC class that they align with on their right side.

The defined classes :SpicyPizza and :SpicyPizzaEquivalent deserves a special remark. As their names suggest, they provide two implementations to represent the same concept. The two implementations fit that of a defined class in the Normalisation ODP given in Definition 5.1. They differ on the class :M_iElem_j, and object property :$hasModule_i$ (and therefore on the module :$Module_i$ as well), that each one is associated with.

- :SpicyPizza is defined in terms of :SpicyTopping via the object property :hasTopping (part of :$Module_3$ or :PizzaTopping in Figure 7.3).
- :SpicyPizzaEquivalent is defined in terms of :Hot via the object property :hasSpiciness (part of :$Module_4$ or :Spiciness in Figure 7.4).

From Figure 7.6, it can be observed as well, that not all elements :M_iElem_j from the four modules :$Module_i$ in the "Pizza" ontology, contribute with a corresponding defined subclass :M_iClass_jTDC to the :$Pizza$ target domain concept as detailed in the

```
:InterestingPizza

    rdf:type owl:Class ;

    rdfs:label "PizzaInteressante"@pt ;

    owl:equivalentClass
        [ rdf:type owl:Class ;
          owl:intersectionOf ( :Pizza
                               [ rdf:type owl:Restriction ;
                                 owl:onProperty :hasTopping ;
                                 owl:minCardinality "3"^^xsd:nonNegativeInteger
                               ]
                             )
        ] ;

    rdfs:comment "Any pizza that has at least 3 toppings.
                  Note that this is a cardinality constraint on the hasTopping property
                  and NOT a qualified cardinality constraint (QCR).
                  A QCR would specify from which class the members in this relationship
                  must be. eg has at least 3 toppings from PizzaTopping.
                  This is currently not supported in OWL."@en .
```

Listing 7.2: Implementation of the defined class $:InterestingPizza$

Normalisation ODP generic structure. In general, such population of defined classes of the target domain concept class $:TDC$ is fine but also optional, driven by the ontology requirements and the relevance to the ontology model of the define class in question.

***Compound* Defined Classes** Conversely, there are some $:Pizza$ defined subclasses whose implementation deviates from that of a defined class $:M_iClass_jTDC$ in the Normalisation ODP, however, from a functionality standpoint, they still fulfil the same role. They allow a reasoner to automatically manage the subsumption relations between them and the elements in the ontology to be classified (the elements $:SpecificTDC_x$ that are part of the $:TDC$ class structure). Note the elements $:SpecificTDC_x$ in Figure 5.2 of the Normalisation ODP generic structure and in Figure 7.6 of the "Pizza" ontology example).

For example, the implementation of the class :InterestingPizza (see Listing 7.2), does not follow the implementation of a generic $:M_iClass_jTDC$ class given that it does not correspond to any class $:M_iElem_j$ in any of the modules in the model. Yet, a reasoner would automatically infer that any element of the ontology that has at least three toppings is a subclass of or an individual of type :InterestingPizza. That is exactly what happens to all pizzas subsumed by :NamedPizza in Figure 7.6 except for :QuattroFormaggi and :UnclosedPizza, which do not meet the cardinality constraint of containing 3 or more toppings.

There rest of defined classes subsumed by $:Pizza$ that present a behaviour similar to :InterestingPizza (they do not follow the normalisation implementation, yet they provide the normalisation functionality) are :NonVegeterianPizza, :VegeterianPizza, and :VegeterianPizzaEquivalent2. This is indicated in Figure 7.6 with the annotation "*(no direct mapping to a :$M_i Elem_j$)*" on the right side of the class.

Sources of Defined Classes In the case of the "Pizza" ontology, all existing modules in the example (:Country, :PizzaBase, :PizzaTopping, and :Spiciness) contribute at least with one element to the set of defined subclasses of :Pizza.

- The module with more representation is :PizzaTopping, for which there are four defined classes associated to it via the property :hasTopping (:CheeseyPizza, :MeatyPizza, :SpicyPizza, and :VegetarianPizzaEquivalent1). The rest of modules contribute with one element to the set of the defined :Pizza subclasses.
- The module :Country is represented by the class :RealItalianPizza via the property :hasCountryOfOrigin.
- The module :PizzaBase is represented by the class :ThinAndCrispyPizza via the property :hasBase.
- Finally the module :Spiciness is represented by the class :SpicyPizza through the property :hasSpiciness.

7.1.3.2 Classification Elements

The classification elements (classes or individuals) in the Normalisation ODP refer to the elements :$SpecificTDC_x$ of the generic structure. Figure 7.6 lists the classes that align to a :$SpecificTDC_x$ element in the "Pizza" ontology. They are the subclasses of the class :NamedPizza and represent all the specific types of pizzas that a reasoner should automatically classify under the appropriate defined class(es) of :Pizza. The implementation of the subclasses of :NamedPizza as per the normalisation mechanism, enables a reasoner to infer all the subsumption relations of these subclasses.

For example, consider the implementation of the class :Napoletana, a subclass of :NamedPizza shown in Listing 7.3.

- The relationship between the class :Napoletana and the class :MozarellaTopping via the object property :hasTopping enables a reasoner to infer that :Napoletana is a subclass of the defined class :CheeseyPizza.
- The relationship between the class :Napoletana and the individual :Italy via the object property :hasCountryOfOrigin enables a reasoner to infer that :Napoletana is a subclass of the defined class :RealItalianPizza.

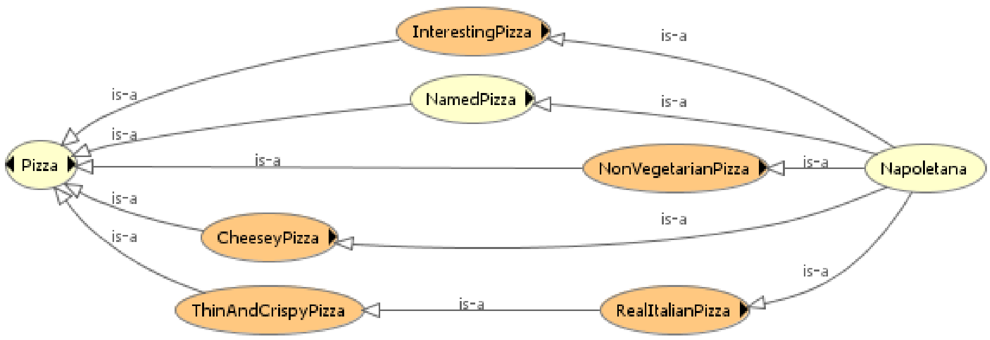


Figure 7.7: Inferred Napoletana Pizza.

- The rest of existential restrictions on the object property :hasTopping of the :Napoletana class, enables a reasoner to infer that :Napoletana is a subclass of the defined classes :InterestingPizza and :NonvegetarianPizza.

Figure 7.6 presented the asserted class hierarchy of the "Pizza" ontology, including the class :Napoletana. On the other hand, Figure 7.7 illustrates all the subsumption inferences that a reasoner automatically calculates for the :Napoletana class based on the implmentation as per the Normalisation ODP of the ontology elements involved.

The same analysis applies to the full list of specific pizzas subsumed by :NamedPizza (from :American to :Veneziana) in Figure 7.6 of this "Pizza" ontology model example.

7.2 The Faceted Classification Scheme of "Pizza"

The previous sections have approached the "Pizza" ontology model example as an instantiation of the Normalisation ODP in the terms presented throughout Chapter 5, detailing the existing alignment between the two.

This section is an attempt to project this alignment one step further and try to present a hypothetical FCS that would result in the "Pizza" ontology model example under evaluation, based on the established alignments between a FCS and the Normalisation ODP described in Chapter 6.

This exercise can be seen as a test of the bidirectionality of the transformation guidelines of a FCS to a normalised ontology model.

With the transformation information summarised in Table 6.2 in mind, an initial trivial attempt to produce the hypothetical FCS that would correspond to the "Pizza" ontology example is captured in Table 7.1.

In that sense, Table 7.1 is populating the elements of the a generic FCS in the terms given in Definition 6.1 as follows:

```
:Napoletana rdf:type owl:Class ;

            rdfs:label "Napoletana"@pt ;

            rdfs:subClassOf :NamedPizza ,
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasTopping ;
                              owl:someValuesFrom :OliveTopping
                            ] ,
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasCountryOfOrigin ;
                              owl:hasValue :Italy
                            ] ,
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasTopping ;
                              owl:someValuesFrom :AnchoviesTopping
                            ] ,
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasTopping ;
                              owl:someValuesFrom :CaperTopping
                            ] ,
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasTopping ;
                              owl:someValuesFrom :TomatoTopping
                            ] ,
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasTopping ;
                              owl:allValuesFrom [ rdf:type owl:Class ;
                                                  owl:unionOf ( :AnchoviesTopping
                                                                :CaperTopping
                                                                :MozzarellaTopping
                                                                :OliveTopping
                                                                :TomatoTopping
                                                              )
                                                ]
                            ] ,
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasTopping ;
                              owl:someValuesFrom :MozzarellaTopping
                            ] ;

            owl:disjointWith :Parmense ,
                             :PolloAdAstra ,
                             :PrinceCarlo ,
                             :QuattroFormaggi ,
                             :Rosa ,
                             :Siciliana ,
                             :SloppyGiuseppe ,
                             :Soho ,
                             :UnclosedPizza ,
                             :Veneziana .
```

Listing 7.3: Implementation of the primitive class :Napoletana

Pizza Faceted Classification Scheme		
Facet		**Term**
Country		America, England, France, Germany, Italy
Base		Deep pan, Thin and crispy
Topping	Cheese	Four cheeses, Goat, Gorgonzolla, Mozarella, Parmesan
	Fish	Anchovy, Mixed seafood, Prawn
	Fruit	Sultana raisin
	Herb and spice	Cajun spice, Rosemary
	Meat	Chicken, Ham, Hot and spiced beef, Pepperoni sausage
	Nut	Pine
	Sauce	Tabasco pepper
	Vegetable	Artichoke, Asparagus, Caper, Garlic, Leek, Mushroom, Olive, Onion: {Red, White}, Pepper: {Green: {Hot}, Jalapeno, Pepperonata, Sweet}, Petit pois, Rocket, Spinach, Tomato: {Sliced, Sundried}
Spiciness		Hot, Medium, Mild

Table 7.1: Hypothetical “Pizza” FCS based on the “Pizza” ontology model example.

- The target domain concept (TDC) is obviously derived from the class :Pizza and is the “Pizza” domain.
- The facets ($Facet_i$) correspond to the four modules, semantic axes or principles of division that are represented implicitly in the “Pizza” ontology example. In the table: “Country”, “Base”, “Topping” and “Spiciness”.
- The terms of the facets (F_iTerm_j) are derived from the elements :M_iElem_j of the modules.
- The items to classify ($Item_x$) correspond to the elements :$SpecificTDC_x$ of the target domain concept :Pizza, which in this case are the subclasses of :NamedPizza (:American, AmericanHot, Cajun, etc.).

7.2.1 The Facet of “Topping”

There is an aspect of the hypothetical Pizza FCS in Table 7.1 that presents a challenge to the transformation guidelines given so far from Chapter 6, which is the facet or semantic axis of “Topping”.

As Figure 7.3 shows, the concept of “Topping” represented in the “Pizza” ontology example is made of a hierarchy of classes of up to four levels of depth. To accommodate such hierarchy structure in a FCS may require the use of *subfacets* (Vickery (2008)). The transformation guidelines for a FCS identified in Chapter 6 focus on a simplified domain FCS structure and initially do not consider the case of subfacets.

The characterisation of the facet "Topping" in Table 7.1 is the result of a naive and intuitive approach, where: (a) the leaf nodes of the class hierarchy structure of the module :Topping are mapped to an element F_iTerm_j in the FCS structure; and (b) the internal (non-leaf) nodes are represented as a *subfacet* of the facet "Topping".

For simplicity, only one sub-level of the facet "Topping" is represented in the column "Facet" of Table 7.1. The rest are indicated in the column "Term", using curly-brackets such as: "Onion {Red, White}", etc.

Again, this approach is merely intuitional and the support and transformation of a *subfacet* in the context of this research requires further investigation.

7.3 Conclusions

The example ontology model in the domain of "Pizza" by Horridge et al. (2009) is evaluated with the focus set on the modelling of multiple classification criteria.

The elements that conform the "Pizza" ontology are dissected and analyzed using a reverse engineering approach, with the goal of identifying alignments to the elements of the generic structure of the three ODPs revisited throughout this research.

The analysis reveals that the "Pizza" ontology model can be seen as an instantiation of the Normalisation ODP, where the class :Pizza is the :TDC element of the pattern and there are four modules or semantic axes :$Module_i$, represented by the classes: :Country, :PizzaBase, :PizzaTopping and :Spiciness.

In fact, the "Pizza" ontology model meets one of the main conditions of normalisation, and it does not present any multiple inheritance relations in the asserted ontology model. That is, there are no classes with more than one superclass manually asserted.

This instantiation of the Normalisation ODP in the "Pizza" ontology is composed of the instantiation of other ODPs such as:

- An instantiation of the Value Partition ODP, where the element :TDC of the pattern is also the class :Pizza, and the element :$Value_i$ is represented by the class :Spiciness.
- An instantiation of the simplified version of the Class As Property Value ODP, (in which the notion of interpretation and terminology conflate), where the element :TDC of the pattern is also the class :Pizza, and two elements :$Terminology_i$ are populated by the class :PizzaBase and :PizzaTopping respectively.

Not all defined classes :M_iElem_jTDC of the target domain concept in the Normalisation ODP, has to follow the prototypical implementation in Definition 5.1 and being derived from a single element :M_iElem_j, (what has been referred to as *single defined classes*). They can become fairly sophisticated and combine various elements :M_iElem_j to create interesting subsets of the target domain concept, (what has been called *compound defined classes*).

It can also happen that an instantiation of the Normalisation ODP includes another instantiation of the pattern on a smaller scale. This can be observed in the “Pizza” ontology example as well, where another instance of the Normalisation ODP can be seen with the elements:

- The class :PizzaTopping as the target domain concept :TDC.
- The class :Spiciness as a module or semantic axis again :$Module_i$. In this case, the domain of “Pizza Topping” classified by its spiciness level.
- The class :SpicyTopping as an element :M_iElem_jTDC, a defined subclass of :PizzaTopping.
- The various subclasses of :PizzaTopping as the classification elements :$SpecificTDC_x$.

This instance of the Normalisation ODP at the :PizzaTopping class scope, (taking place within the overall instantiation at the :Pizza class scope), allows an OWL DL reasoner to automatically classify the classes :CajunSpiceTopping, :HotGreenPepperTopping, :HotSpicedBeefTopping, :JalapenoPepperTopping and :TobascoPepperSauce as subclasses of the defined class :SpicyTopping.

The instantiations of the Normalisation ODP identified throughout this analysis of the “Pizza” ontology example brings to the forefront the classification criteria implicitly considered in the ontology model, namely “Pizza” viewed by the “country of origin”, the “type of base”, the “type of toppings” and the “level of spiciness”. With these classification criteria, these semantic axes or principles of division, and the generic structure of the Faceted Classification ODP from Chapter 6, a hypothetical FCS in the domain of “Pizza” is proposed.

The creation of the “Pizza” FCS from the “Pizza” ontology model, suggests areas of the transformation guidelines between the two knowledge resources that require further investigation such as the inclusion of *subfacets*.

Chapter 8

Evaluating the "Wine" Domain Concept in the OWL Language Guide

The next evaluation ontology model example focuses on the domain concept of "Wine". As mentioned in the introduction to this chapter, an early version of this "Wine" ontology was used to provide basic guidelines on how to create your first ontology [Noy and McGuinness (2001)]. The authors of the first guide to the OWL Web ontology language Welty et al. (2004), created a new version of the "Wine" ontology to introduce and navigate readers throughout the elements and features of the the first version of the W3C OWL specification, also referred to as OWL 1.0.

In the example ontology, wines are described based on their type of grape, region of production, color, body, flavor, and even sugar levels. Once again, with the modeling problem that motivated this research, those various characteristics that determine a particular type of wine, can be regarded as multiple classification criteria for the concept of wine.

As an introductory guide, the referred document does not delve into topics pertaining to ODPs. There are no references to any of the ODPs discussed so far. Yet, the following evaluation illustrates that the "Wine" example ontology model implicitly exhibits instances of the Value Partition ODP and partly even of the Normalisation ODP.

The "Wine" ontology employed throughout Welty et al. (2004), is based on an already existing ontology model available online[1] and is the model that will be used to carry out this evaluation.

[1] http://www.w3.org/TR/owl-guide/wine.rdf

8.1 Structure of the “Wine” Ontology

The heuristic procedure to examine the “Wine” ontology example is the same as in the case of the “Pizza” ontology example, and it is based on a reverse engineering approach. The sections that follow, discuss the alignment of the “Wine” ontology example to the elements of the generic structure of the Normalisation ODP (:$Module_i$, :M_iElem_j, :$hasModule_i$, :TDC, :M_iClass_jTDC, and :$SpecificTDC_x$).

8.1.1 Modules

- :$Module_1$ (vin:WineGrape) does not align to any of the elements of the Value Partition or Class As Property Value ODPs, given that on one hand, the elements involved (the individuals :M_1Elem_j) do not form a partition and on the other, these are individuals being used as property values, not classes. The individuals of the vin:WineGrape class are used as values for the object property vin:madeFromGrape.
- :$Module_2$ (vin:Region) does not align to any of the elements of the Value Partition or Class As Property Value ODPs, given that on one hand, the elements involved (the individuals :M_2Elem_j) do not form a partition and on the other, these are individuals being used as property values, not classes. The individuals of the vin:Region class are used as values for the object property vin:locatedIn.
- :$Module_3$ (vin:WineColor) is explicitly presented as a Value Partition and therefore aligns to the element :$Value_i$ of the Value Partition ODP. The object property used to implement the pattern is vin:hasColor.
- :$Module_4$ (vin:WineBody) is explicitly presented as a Value Partition and therefore aligns to the element :$Value_i$ of the Value Partition ODP. The object property used to implement the pattern is vin:hasBody.
- :$Module_5$ (vin:WineFlavor) is explicitly presented as a Value Partition and therefore aligns to the element :$Value_i$ of the Value Partition ODP. The object property used to implement the pattern is vin:hasFlavor.
- :$Module_6$ (vin:WineSugar) is explicitly presented as a Value Partition and therefore aligns to the element :$Value_i$ of the Value Partition ODP. The object property used to implement the pattern is vin:hasSugar.

8.1.2 Object Properties

Figure 8.4 summarizes the object properties in the “Wine” ontology including the domain and range (if any) of each property and again, on the far right column the element

```
owl:Thing                                                  (Normalization ODP
  |-- food:ConsumableThing                                  Generic Structure)
     |-- food:EdibleThing
        |-- food:SweetFruit
           |-- food:Grape
              |-- vin:WineGrape                            (:Module1)
                 |-- (◊) vin:CabernetFrancGrape               (:M1Elem1)
                 |-- (◊) vin:CabernetSauvignonGrape           (:M1Elem2)
                 |-- (◊) vin:ChardonnayGrape                  (:M1Elem3)
                 |-- (◊) vin:CheninBlancGrape                 (:M1Elem4)
                 |-- (◊) vin:GamayGrape                       (:M1Elem5)
                 |-- (◊) vin:MalbecGrape                      (:M1Elem6)
                 |-- (◊) vin:MerlotGrape                      (:M1Elem7)
                 |-- (◊) vin:PetiteSyrahGrape                 (:M1Elem8)
                 |-- (◊) vin:PetiteVerdotGrape                (:M1Elem9)
                 |-- (◊) vin:PinotBlancGrape                  (:M1Elem10)
                 |-- (◊) vin:PinotNoirGrape                   (:M1Elem11)
                 |-- (◊) vin:RieslingGrape                    (:M1Elem12)
                 |-- (◊) vin:SangioveseGrape                  (:M1Elem13)†
                 |-- (◊) vin:SauvignonBlancGrape              (:M1Elem14)
                 |-- (◊) vin:SemillonGrape                    (:M1Elem15)
                 |-- (◊) vin:ZinfandelGrape                   (:M1Elem16)
```

(†) denotes elements that do not participate in the definition of a defined class subsumed by vin:Wine (see Section 8.1.3.1(§ Defined Classes)).

Figure 8.1: The Grape of "Wine".

of the Normalisation ODP generic structure that the property maps into. Note how the one-to-one relation in the Normalisation ODP generic structure between the class $:Module_i$ and the object property $:hasModule_i$ holds in the "Wine" ontology between the classes vin:WineGrape, vin:Region, vin:WineColor, vin:WineBody, vin:WineFlavor, vin:WineSugar and the object properties vin:madeFromGrape, vin:locatedIn, vin:hasColor, vin:hasBody, vin:hasFlavor, vin:hasSugar respectively.

8.1.3 Target Domain Concept

In terms of the notation used to characterize the generic structure of any of the ODPs covered in previous chapters (i.e. Figure 5.2), the element of the "Wine" ontology that corresponds to the target domain concept or the $:TDC$ element, is the class food:Wine or the class vin:Wine (which in the ontology are equivalent).

The main OWL elements for the purpose of this evaluation, that are part of the vin:Wine class can be organized into two groups. One group is formed by all the classes (in their majority *defined classes*), subsumed by the $:TDC$ class vin:Wine, and another group is formed by all the individuals that are of type vin:Wine. Figure 8.5 and Figure 8.6 present these two groups respectively.

Observing the structure of the "Wine" ontology and the generic structure of the Normalisation ODP given in Figure 5.2, a correlation can be drawn between the elements in Figure 8.5 and Figure 8.6:

`owl:Thing`	($Normalization\ ODP\ Generic\ Structure$)
`|-- vin:Region`	(:$Module_2$)
`|-- (◊) vin:AlsaceRegion`	(:M_2Elem_1)
`|-- (◊) vin:AnjouRegion`	(:M_2Elem_2)
`|-- (◊) vin:ArroyoGrandeRegion`	(:M_2Elem_3)*
`|-- (◊) vin:AustralianRegion`	(:M_2Elem_4)*
`|-- (◊) vin:BeaujolaisRegion`	(:M_2Elem_5)
`|-- (◊) vin:BordeauxRegion`	(:M_2Elem_6)
`|-- (◊) vin:BourgogneRegion`	(:M_2Elem_7)
`|-- (◊) vin:CaliforniaRegion`	(:M_2Elem_8)
`|-- (◊) vin:CentralCoastRegion`	(:M_2Elem_9)*
`|-- (◊) vin:CentralTexasRegion`	(:M_2Elem_{10})*
`|-- (◊) vin:ChiantiRegion`	(:M_2Elem_{11})†
`|-- (◊) vin:CotesDOrRegion`	(:M_2Elem_{12})
`|-- (◊) vin:EdnaValleyRegion`	(:M_2Elem_{13})*
`|-- (◊) vin:FrenchRegion`	(:M_2Elem_{14})
`|-- (◊) vin:GermanyRegion`	(:M_2Elem_{15})
`|-- (◊) vin:ItalianRegion`	(:M_2Elem_{16})
`|-- (◊) vin:LoireRegion`	(:M_2Elem_{17})
`|-- (◊) vin:MargauxRegion`	(:M_2Elem_{18})
`|-- (◊) vin:MedocRegion`	(:M_2Elem_{19})
`|-- (◊) vin:MendocinoRegion`	(:M_2Elem_{20})*
`|-- (◊) vin:MeursaultRegion`	(:M_2Elem_{21})
`|-- (◊) vin:MuscadetRegion`	(:M_2Elem_{22})
`|-- (◊) vin:NapaRegion`	(:M_2Elem_{23})*
`|-- (◊) vin:NewZealandRegion`	(:M_2Elem_{24})*
`|-- (◊) vin:PauillacRegion`	(:M_2Elem_{25})
`|-- (◊) vin:PortugalRegion`	(:M_2Elem_{26})†
`|-- (◊) vin:SancerreRegion`	(:M_2Elem_{27})
`|-- (◊) vin:SantaBarbaraRegion`	(:M_2Elem_{28})*
`|-- (◊) vin:SantaCruzMountainsRegion`	(:M_2Elem_{29})*
`|-- (◊) vin:SauterneRegion`	(:M_2Elem_{30})†
`|-- (◊) vin:SonomaRegion`	(:M_2Elem_{31})*
`|-- (◊) vin:SouthAustraliaRegion`	(:M_2Elem_{32})*
`|-- (◊) vin:StEmilionRegion`	(:M_2Elem_{33})
`|-- (◊) vin:TexasRegion`	(:M_2Elem_{34})
`|-- (◊) vin:ToursRegion`	(:M_2Elem_{35})
`|-- (◊) vin:USRegion`	(:M_2Elem_{36})

($*, \dagger$) denote elements that do not participate in the definition of a defined class subsumed by vin:Wine (see Section 8.1.3.1(§ Defined Classes)).

Figure 8.2: The Region of “Wine”.

- Many of the classes in Figure 8.5 align to a generic class :M_iClass_jTDC in the context of the Normalisation ODP generic structure.

- The individuals in Figure 8.6 corresponds to a generic element :$SpecificTDC_x$ in the generic structure of the same ODP.

To highlight this correlation, Figure 8.5 includes the specific class :M_iClass_jTDC on the right-hand side of the relevant subclass of the class vin:Wine, while Figure 8.6 includes the element :$SpecificTDC_i$ also on the right-hand side of the individual of the class vin:Wine.

```
owl:Thing                                          (Normalization ODP Generic Structure)
  |-- (≡) vin:WineDescriptor
     |-- (≡)(P) vin:WineColor                      (:Module_3)
        |-- (◊) vin:Red                               (:M_3Elem_1)
        |-- (◊) vin:Rose                              (:M_3Elem_2)
        |-- (◊) vin:White                             (:M_3Elem_3)
     |-- :WineTaste
        |-- (≡)(P) vin:WineBody                    (:Module_4)
           |-- (◊) vin:Full                           (:M_4Elem_1)
           |-- (◊) vin:Light                          (:M_4Elem_2)†
           |-- (◊) vin:Medium                         (:M_4Elem_3)†
        |-- (≡)(P) vin:WineFlavor                  (:Module_5)
           |-- (◊) vin:Delicate                       (:M_5Elem_1)†
           |-- (◊) vin:Moderate                       (:M_5Elem_2)†
           |-- (◊) vin:Strong                         (:M_5Elem_3)†
        |-- (≡)(P) vin:WineSugar                   (:Module_6)
           |-- (◊) vin:Dry                            (:M_6Elem_1)
           |-- (◊) vin:OffDry                         (:M_6Elem_2)
           |-- (◊) vin:Sweet                          (:M_6Elem_3)
```

(†) denote elements that do not participate in the definition of a defined class subsumed by vin:Wine (see Section 8.1.3.1(§ Defined Classes)).

Figure 8.3: The Color, Body, Flavor and Sugar of “Wine”.

```
owl:topObjectProperty          (rdfs : domain)        (rdfs : range)        (Norm. ODP)
   |-- vin:hasMaker
   |-- vin:hasWineDescriptor   vin:Wine               vin:WineDescriptor
      |-- vin:hasBody                                 vin:WineBody          (:hasModule_4)
      |-- vin:hasColor         vin:Wine               vin:WineColor         (:hasModule_3)
      |-- vin:hasFlavor                               vin:WineFlavor        (:hasModule_5)
      |-- vin:hasSugar                                vin:WineSugar         (:hasModule_6)
   |-- vin:locatedIn           owl:Thing              vin:Region            (:hasModule_2)
   |-- food:madeFromFruit      food:ConsumableThing   food:Fruit
      |-- vin:madeFromGrape    vin:Wine               vin:WineGrape         (:hasModule_1)
```

Figure 8.4: Properties of the ontology model for “Wine”.

8.1.3.1 Defined Classes

Figure 8.5 presents all the defined classes subsumed by the vin:Wine class in the “Wine” ontology. Similar to the case of the “Pizza” ontology model in Chapter 7, the total number of defined classes can be further subdivided into two different groups based on their implementation in the “Wine” ontology. These will be referred to as “Single Defined Classes” and “Compound Defined Classes”; and both groups are presented in the sections below.

***Single* Defined Classes** The first group is formed by the defined classes whose implementation conforms to the implementation of a generic element $:M_iClass_jTDC$ in the Normalisation ODP generic structure presented in Definition 5.1. The main characteristic of this implementation lies in the *one-to-one* relationship between the defined class $:M_iClass_jTDC$ and the element $:M_iElem_j$ that the defined class is derived

```
owl:Thing                                   ($Normalization\ ODP\ Generic\ Structure$)
  |-- food:Wine (≡) vin:Wine                  (:$TargetDomainConcept$)
     |-- (≡) vin:AlsatianWine                   (:$M_2Class_1TDC$)
     |-- (≡) vin:AmericanWine                   (:$M_2Class_{36}TDC$)
     |-- (≡) vin:Beaujolais                     (:$M_2Class_5TDC$)
     |-- (≡) vin:Bordeaux                       (:$M_2Class_6TDC$)
        |-- (≡) vin:Medoc                          (:$M_2Class_{19}TDC$)
           |-- (≡) vin:Margaux                        (:$M_2Class_{18}TDC$)
           |-- (≡) vin:Pauillac                       (:$M_2Class_{25}TDC$)
        |-- (≡) vin:RedBordeaux                    (related to :$M_2Elem_6$, :$M_3Elem_1$)
        |-- vin:Sauternes
        |-- (≡) vin:StEmilion                      (:$M_2Class_{33}TDC$)
        |-- (≡) vin:WhiteBordeaux                  (related to :$M_2Elem_6$, :$M_3Elem_3$)
     |-- (≡) vin:Burgundy                       (:$M_2Class_7TDC$)
        |-- (≡) vin:RedBurgundy                    (related to :$M_2Elem_7$, :$M_3Elem_1$)
           |-- (≡) vin:CotesDOr                       (:$M_2Class_{12}TDC$)
        |-- (≡) vin:WhiteBurgundy                  (related to :$M_2Elem_7$, :$M_3Elem_3$)
           |-- (≡) vin:Meursault                      (:$M_2Class_{21}TDC$)
     |-- (≡) vin:CabernetFranc                  (:$M_1Class_1TDC$)
     |-- (≡) vin:CabernetSauvignon              (:$M_1Class_2TDC$)
     |-- (≡) vin:CaliforniaWine                 (:$M_2Class_8TDC$)
     |-- (≡) vin:Chardonnay                     (:$M_1Class_3TDC$)
     |-- (≡) vin:CheninBlanc                    (:$M_1Class_4TDC$)
     |-- vin:DessertWine
        |-- (≡) vin:IceWine                        (related to :$M_3Elem_3$, :$M_5Elem_2$, :$M_5Elem_3$, :$M_6Elem_3$)
        |-- (≡) vin:SweetRiesling                  (related to :$M_1Elem_{12}$, :$M_6Elem_3$)
     |-- (≡) vin:DryWine                        (:$M_6Class_1TDC$)
        |-- (≡) vin:DryRedWine                     (related to :$M_6Elem_1$, :$M_3Elem_1$)
        |-- (≡) vin:DryWhiteWine                   (related to :$M_6Elem_1$, :$M_3Elem_3$)
     |--  vin:EarlyHarvest
     |-- (≡) vin:FrenchWine                     (:$M_2Class_{14}TDC$)
     |-- (≡) vin:FullBodiedWine                 (:$M_4Class_1TDC$)
     |-- (≡) vin:Gamay                          (:$M_1Class_5TDC$)
     |-- (≡) vin:GermanWine                     (:$M_2Class_{15}TDC$)
     |-- (≡) vin:ItalianWine                    (:$M_2Class_{16}TDC$)
        |-- vin:Chianti
     |-- vin:LateHarvest
        |-- (≡) vin:IceWine                        (related to :$M_3Elem_3$, :$M_5Elem_2$, :$M_5Elem_3$, :$M_6Elem_3$)
        |-- vin:Sauternes
     |-- (≡) vin:Loire                          (:$M_2Class_{17}TDC$)
        |-- (≡) vin:Anjou                          (:$M_2Class_2TDC$)
        |-- (≡) vin:Muscadet                       (:$M_2Class_{22}TDC$)
        |-- (≡) vin:Sancerre                       (:$M_2Class_{27}TDC$)
        |-- (≡) vin:Tours                          (:$M_2Class_{35}TDC$)
        |-- (≡) vin:WhiteLoire                     (related to :$M_2Elem_{17}$, :$M_3Elem_3$)
     |-- (≡) vin:Meritage                       (related to :$M_1Elem_1$, :$M_1Elem_2$, :$M_1Elem_6$, :$M_1Elem_7$, :$M_1Elem_9$)
     |-- (≡) vin:Merlot                         (:$M_1Class_7TDC$)
     |-- (≡) vin:PetiteSyrah                    (:$M_1Class_8TDC$)
     |-- (≡) vin:PinotBlanc                     (:$M_1Class_{10}TDC$)
     |-- (≡) vin:PinotNoir                      (:$M_1Class_{11}TDC$)
     |-- (≡) vin:RedWine                        (:$M_3Class_1TDC$)
        |-- (≡) vin:DryRedWine                     (related to :$M_6Elem_1$, :$M_3Elem_1$)
        |-- vin:Port
        |-- (≡) vin:RedBordeaux                    (related to :$M_2Elem_6$, :$M_3Elem_1$)
        |-- (≡) vin:RedBurgundy                    (related to :$M_2Elem_7$, :$M_3Elem_1$)
           |-- (≡) vin:CotesDOr                       (:$M_2Class_{12}TDC$)
     |-- (≡) vin:Riesling                       (:$M_1Class_{12}TDC$)
        |-- (≡) vin:DryRiesling                    (related to :$M_1Elem_{12}$, :$M_6Elem_1$)
     |-- (≡) vin:RoseWine                       (:$M_3Class_2TDC$)
     |-- (≡) vin:SemillonOrSauvignonBlanc       (related to :$M_1Elem_{14}$, :$M_1Elem_{15}$)
        |-- (≡) vin:SauvignonBlanc                 (:$M_1Class_{14}TDC$)
        |-- (≡) vin:Semillon                       (:$M_1Class_{15}TDC$)
     |-- (≡) vin:SweetWine                      (:$M_6Class_3TDC$)
     |-- (≡) vin:TableWine                      (:$M_6Class_1TDC$)
        |-- (≡) vin:RedTableWine                   (related to :$M_6Elem_1$, :$M_3Elem_1$)
        |-- (≡) vin:WhiteTableWine                 (related to :$M_6Elem_1$, :$M_3Elem_3$)
     |-- (≡) vin:TexasWine                      (:$M_2Class_{34}TDC$)
     |-- (≡) vin:WhiteWine                      (:$M_3Class_3TDC$)
        |-- (≡) vin:DryWhiteWine                   (related to :$M_6Elem_1$, :$M_3Elem_3$)
        |-- (≡) vin:WhiteBordeaux                  (related to :$M_2Elem_6$, :$M_3Elem_3$)
        |-- (≡) vin:WhiteBurgundy                  (related to :$M_2Elem_7$, :$M_3Elem_3$)
           |-- (≡) vin:Meursault                      (:$M_2Class_{21}TDC$)
        |-- (≡) vin:WhiteLoire                     (related to :$M_2Elem_{17}$, :$M_3Elem_3$)
        |-- (≡) vin:WhiteNonSweetWine              (related to :$M_6Elem_1$, :$M_6Elem_2$, :$M_3Elem_3$)
     |-- (≡) vin:Zinfandel                      (:$M_1Class_{16}TDC$)
```

Figure 8.5: The “Wine” domain concept (part 1). The owl:Class elements.

```
:ItalianWine rdf:type owl:Class ;

             owl:equivalentClass [ rdf:type owl:Class ;
                                   owl:intersectionOf ( :Wine
                                                        [ rdf:type owl:Restriction ;
                                                          owl:onProperty :locatedIn ;
                                                          owl:hasValue :ItalianRegion
                                                        ]
                                                      )
                                 ] .
```

Listing 8.1: Implementation of the defined class :ItalianWine

from, via the property :$hasModule_i$. Figure 8.5 indicates the defined classes that belong to this group specifying on the right-hand side of the figure, the *single* :M_iClass_jTDC class associated to them.

For example, consider the implementation of the defined class :ItalianWine in Listing 8.1 extracted verbatim from the “Wine” ontology. The one-to-one relationship between the classes :ItalianWine and :ItalianRegion (from the module :Region), via the property :locatedIn, holds similarly as it does between the classes :M_iClass_jTDC and :M_iElem_j (in module :$Module_i$), via the property :$hasModule_i$ in Definition 5.1.

For the purpose of this evaluation, the class :ItalianRegion in the “Wine” ontology, is aligned to the class :M_2Elem_{16} in :$Module_2$ (see Figure 8.2). Along the same line, the class :ItalianWine is aligned to class :$M_2Class_{16}TDC$ (see Figure 8.5) and the property :locatedIn aligns to the property :$hasModule_2$ (see Figure 8.4).

As it can be observed from Figure 8.5, roughly two thirds of the defined classes subsumed by vin:Wine, belong to this group (they are associated to a single :M_iClass_jTDC element) and their implementation is similar to that of :ItalianWine in Listing 8.1. Classes such as vin:AmericanWine, vin:Bordeaux, vin:Burgundy, vin:DryWine, vin:FullBodiedWine, vin:RedWine, vin:Rose, vin:SweetWine, vin:TableWine, vin:WhiteWine, etc. are additional examples that belong to this group.

***Compound* Defined Classes** The second group is formed by the defined classes whose implementation does *not* conform to the implementation of a generic element :M_iClass_jTDC in the Normalisation ODP generic structure presented in Definition 5.1.

The implementation of the defined classes in this group exhibits a *one-to-many* relationship between the defined class :M_iClass_jTDC and various elements :M_iElem_j that the defined class is derived from. The various elements :M_iElem_j might belong to different modules :$Module_i$ or to the same. For each different module :$Module_i$ involved in the implementation of the defined class, there will be a different property :$hasModule_i$ involved respectively.

```
:DryRiesling rdf:type owl:Class ;

    owl:equivalentClass [ rdf:type owl:Class ;
                          owl:intersectionOf ( :Riesling
                                               [ rdf:type owl:Restriction ;
                                                 owl:onProperty :hasSugar ;
                                                 owl:hasValue :Dry
                                               ]
                                             )
                        ] ;

    rdfs:subClassOf [ ... subclass restrictions omitted
                    ] .
```

Listing 8.2: Implementation of the defined class :DryRiesling

```
:Riesling rdf:type owl:Class ;

    owl:equivalentClass [ rdf:type owl:Class ;
                          owl:intersectionOf
                              ( :Wine
                                [ rdf:type owl:Restriction ;
                                  owl:onProperty :madeFromGrape ;
                                  owl:hasValue :RieslingGrape
                                ]
                                [ rdf:type owl:Restriction ;
                                  owl:onProperty :madeFromGrape ;
                                  owl:maxCardinality "1"^^xsd:nonNegativeInteger
                                ]
                              )
                        ] ;

    rdfs:subClassOf [ ... subclass restrictions omitted
                    ] .
```

Listing 8.3: Implementation of the defined class :Riesling

Figure 8.5 indicates the defined classes that belong to this group specifying on the right-hand side of the figure, the list of the different elements :M_iElem_j involved in their implementation.

For example, consider the implementation of the defined class vin:DryRiesling in Listing 8.2 (subsumed by the also defined class vin:Riesling in Listing 8.3), extracted verbatim from the “Wine” ontology. In the case of vin:DryRiesling, there is more than *one* single element :M_iElem_j involved in the definition (owl:equivalentClass axiom) of the class implementation, which deviates from the one-to-one relationship prescribed in Definition 5.1 of the Normalisation ODP generic structure for a defined class :M_iClass_jTDC. vin:DryRiesling is related to the class vin:Dry via the property vin:hasSugar but also, as a subclass of vin:Riesling, to the class vin:RieslingGrape via the property vin:madeFromGrape (see Listings 8.2 and 8.3).

For the purpose of this evaluation, the elements in the “Wine” ontology mentioned in the previous example are mapped as follows:

- The defined class vin:DryRiesling has not been mapped to any element of the Normalisation ODP generic structure to highlight the fact that its implementation does not align to any of the elements in the pattern.
- The defined class vin:Riesling maps to the class :$M_1Class_{12}TDC$ (Figure 8.5).
- The individual vin:RieslingGrape maps to the element :M_1Elem_{12} in module vin:WineGrape (:$Module_1$) (Figure 8.1).
- The individual vin:Dry maps to the element :M_6Elem_1 in module vin:WineSugar (:$Module_6$) (Figure 8.3).
- The property vin:hasSugar maps to the property :$hasModule_6$ (Figure 8.4).
- The property vin:madeFromGrape maps to the property :$hasModule_1$ (Figure 8.4).

As it can be observed from Figure 8.5, roughly one third of the defined classes subsumed by vin:Wine, belong to this group (they are associated to a multiple :M_iElem_j elements) and their implementation is similar to that of :DryRiesling in Listing 8.2. Classes such as vin:RedBordeaux, vin:WhiteBordeaux, vin:RedBurgundy, vin:WhiteBurgundy, vin:DryRedWine, vin:DryWhiteWine, etc. are additional examples that belong to this group.

In the case of the implementation of this second group of defined classes, multiple inheritance among classes is being asserted manually, which in clear opposition to the design criteria of the Normalisation ODP. The pattern advocates for this multiple classification to be delegated to the reasoner instead of done manually.

Sources of Defined Classes There is an interesting metric that can be obtained from Figure 8.5 and the total number of defined classes that belong to the two groups described earlier. With this information, it can be determined the number of defined classes that each one of the modules :$Module_i$ contribute with, to the :TDC class vin:Wine. In order for a module to contribute to a defined class of vin:Wine, it is required that an element :M_iElem_j of that module, participates in the definition (owl:equivalentClass axiom) of the defined class implementation. In summary, the number of elements :M_iElem_j that meet that criteria for each module are given below:

- vin:WineGrape (:$Module_1$) contributes 15 out of 16 :M_1Elem_j elements to vin:Wine defined classes.
- vin:WineRegion (:$Module_2$) contributes 23 out of 36 :M_2Elem_j elements to vin:Wine defined classes.
- vin:WineColor (:$Module_3$) contributes 3 out of 3 :M_3Elem_j elements to vin:Wine defined classes.

```
:GaryFarrellMerlot rdf:type owl:NamedIndividual ,
                            :Merlot ;

                   :hasSugar :Dry ;

                   :hasMaker :GaryFarrell ;

                   :hasBody :Medium ;

                   :hasFlavor :Moderate ;

                   :locatedIn :SonomaRegion .
```

Listing 8.4: Implementation of the individual :GaryFarrellMerlot

- vin:WineBody (:$Module_4$) contributes 1 out of 3 :M_4Elem_j elements to vin:Wine defined classes.
- vin:WineFlavor (:$Module_5$) contributes 0 out of 3 :M_5Elem_j elements to vin:Wine defined classes.
- vin:WineSugar (:$Module_6$) contributes 3 out of 3 :M_6Elem_j elements to vin:Wine defined classes.

Conversely, the elements :M_iElem_j from the the various modules (:$Module_1$ to :$Module_6$), that do *not* have a defined class :M_iClass_jTDC associated to them in the vin:Wine subclass hierarchy, are marked either with the symbol (*) or (†) in Figures 8.1, 8.2 and 8.3.

The (†) symbol is used to indicate :M_iElem_j elements that do *not* participate in the owl:equivalentClass axiom of a defined class subsumed by vin:Wine, but *do* participate in a rdfs:subClassOf axiom of a subclass of vin:Wine.

The (*) symbol denotes :M_iElem_j elements that do *not* participate in any of the axioms of a defined subclass of vin:Wine (whether it is owl:equivalentClass, or rdfs:subClassOf).

8.1.3.2 Classification Elements

Figure 8.6 lists all the specific types of wines represented in the “Wine” ontology example. They are owl:NamedIndividuals, members of the class :Wine and they align the element :$SpecificTDC_x$ in the generic structure of the Normalisation ODP.

The implementation of the individuals and defined classes of :Wine as per the normalisation mechanism, enables a reasoner to automatically infer all the class memberships and classify these individuals under the corresponding defined class of :Wine.

For example, consider the implementation of the individual :GaryFarrellMerlot, an individual of :Wine shown in Listing 8.4:

```
owl:Thing                                                    (Normalization ODP Generic Structure)
   |-- food:Wine = vin:Wine                                      (:TargetDomainConcept)
      |-- (◊) vin:BancroftChardonnay                             (:SpecificTDC_1)
      |-- (◊) vin:ChateauChevalBlancStEmilion                    (:SpecificTDC_2)
      |-- (◊) vin:ChateauDeMeursaultMeursault                    (:SpecificTDC_3)
      |-- (◊) vin:ChateauDYchemSauterne                          (:SpecificTDC_4)
      |-- (◊) vin:ChateauLafiteRothschildPauillac                (:SpecificTDC_5)
      |-- (◊) vin:ChateauMargaux                                       ...
      |-- (◊) vin:ChateauMorgonBeaujolais                              ...
      |-- (◊) vin:ChiantiClassico                                      ...
      |-- (◊) vin:ClosDeLaPoussieSancerre                              ...
      |-- (◊) vin:ClosDeVougeotCotesDOr                          (:SpecificTDC_10)
      |-- (◊) vin:CongressSpringsSemillon                              ...
      |-- (◊) vin:CorbansDryWhiteRiesling                              ...
      |-- (◊) vin:CorbansPrivateBinSauvignonBlanc                      ...
      |-- (◊) vin:CorbansSauvignonBlanc                                ...
      |-- (◊) vin:CortonMontrachetWhiteBurgundy                  (:SpecificTDC_15)
      |-- (◊) vin:CotturiZinfandel                                     ...
      |-- (◊) vin:ElyseZinfandel                                       ...
      |-- (◊) vin:FormanCabernetSauvignon                              ...
      |-- (◊) vin:FormanChardonnay                                     ...
      |-- (◊) vin:FoxenCheninBlanc                               (:SpecificTDC_20)
      |-- (◊) vin:GaryFarrellMerlot                                    ...
      |-- (◊) vin:KalinCellarsSemillon                                 ...
      |-- (◊) vin:KathrynKennedyLateral                                ...
      |-- (◊) vin:LaneTannerPinotNoir                                  ...
      |-- (◊) vin:LongridgeMerlot                                (:SpecificTDC_25)
      |-- (◊) vin:MariettaCabernetSauvignon                            ...
      |-- (◊) vin:MariettaOldVinesRed                                  ...
      |-- (◊) vin:MariettaPetiteSyrah                                  ...
      |-- (◊) vin:MariettaZinfandel                                    ...
      |-- (◊) vin:MountadamChardonnay                            (:SpecificTDC_30)
      |-- (◊) vin:MountadamPinotNoir                                   ...
      |-- (◊) vin:MountadamRiesling                                    ...
      |-- (◊) vin:MountEdenVineyardEdnaValleyChardonnay                ...
      |-- (◊) vin:MountEdenVineyardEstatePinotNoir                     ...
      |-- (◊) vin:PageMillWineryCabernetSauvignon                (:SpecificTDC_35)
      |-- (◊) vin:PeterMccoyChardonnay                                 ...
      |-- (◊) vin:PulignyMontrachetWhiteBurgundy                       ...
      |-- (◊) vin:RoseDAnjou                                           ...
      |-- (◊) vin:SantaCruzMountainVineyardCabernetSauvignon           ...
      |-- (◊) vin:SaucelitoCanyonZinfandel                       (:SpecificTDC_40)
      |-- (◊) vin:SaucelitoCanyonZinfandel1998                         ...
      |-- (◊) vin:SchlossRothermelTrochenbierenausleseRiesling         ...
      |-- (◊) vin:SchlossVolradTrochenbierenausleseRiesling            ...
      |-- (◊) vin:SeanThackreySiriusPetiteSyrah                        ...
      |-- (◊) vin:SelaksIceWine                                  (:SpecificTDC_45)
      |-- (◊) vin:SelaksSauvignonBlanc                                 ...
      |-- (◊) vin:SevreEtMaineMuscadet                                 ...
      |-- (◊) vin:StGenevieveTexasWhite                                ...
      |-- (◊) vin:StonleighSauvignonBlanc                              ...
      |-- (◊) vin:TaylorPort                                     (:SpecificTDC_50)
      |-- (◊) vin:VentanaCheninBlanc                             (:SpecificTDC_51)
      |-- (◊) vin:WhitehallLaneCabernetFranc                     (:SpecificTDC_52)
      |-- (◊) vin:WhitehallLanePrimavera                         (:SpecificTDC_53)
```

Figure 8.6: The "Wine" domain concept (part 2). The owl:NamedIndividual elements.

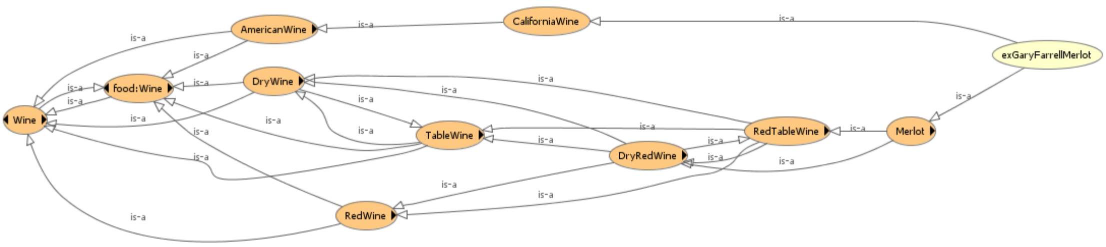


Figure 8.7: Inferred Example of a Specific Wine.

- The relationship between the individual :GaryFarrellMerlot and the individual :Dry via the object property :hasSugar, enables a reasoner to infer that :GaryFarrellMerlot is a member of the defined class :DryWine and its equivalent defined class :TableWine.
- The relationship between the individual :GaryFarrellMerlot and the individual :SonomaRegion via the object property :locatedIn, enables a reasoner to infer that :GaryFarrellMerlot is a member of the defined class :CaliforniaWine.
- The membership of the individual :GaryFarrellMerlot to the defined class :Merlot via the property rdf:type, enables a reasoner to infer that :GaryFarrellMerlot is a member of the defined class :RedWine.

Figure 8.7 illustrates all the class membership inferences that a reasoner automatically calculates for the :GaryFarrellMerlot individual based on the implementation as per Listing 8.4.

The same analysis applies to the full list of 53 specific wines of type :Wine (from vin:BancroftChardonnay to vin:WhitehallLanePrimavera) in Figure 8.6 of this “Wine” ontology model example.

8.2 The Faceted Classification Scheme of “Wine”

The goal of this section is analogous to that of Section 7.2 for the case of the “Pizza” example in Chapter 7.

Once again, this section attempts to present a hypothetical FCS that would result in the “Wine” ontology model example under evaluation, based on the alignments between a FCS and the Normalisation ODP described in Chapter 6. It is another use case that tests the bidirectionality of the transformation guidelines between a FCS and a normalised ontology model.

Table 8.1 presents an initial trivial attempt to produce the hypothetical FCS that would correspond to the “Wine” ontology example, based on the alignments from Table 6.2.

Wine Faceted Classification Scheme	
Facet	**Term**
Grape	Cabernet Franc, Cabernet Sauvignon, Chardonnay, Chenin Blanc, Gamay, Malbec, Merlot, Petite Syrah, Petite Verdot, Pinot Blanc, Pinot Noir, Riesling, Sangiovese, Sauvignon Blanc, Semillon, Zinfandel
Region	Alsace, Anjou, Arroyo Grande, Australia, Beaujolais, Bordeaux, Bourgogne, California, Central Coast, Central Texas, Chianti, Cotes D'Or, Edna Valley, France, Germany, Italy, Loire, Margaux, Medoc, Mendocino, Meursault, Muscadet, Napa, New Zealand, Pauillac, Portugal, Sancerre, Santa Barbara, Santa Cruz Mountains, Sauterne, Sonoma, South Australia, St. Emilion, Texas, Tours, USA
Color	Red, Rose, White
Body	Full, Light, Medium
Flavor	Delicate, Moderate, Strong
Sugar	Dry, Offdry, Sweet

Table 8.1: Hypothetical "Wine" FCS based on the "Wine" ontology model example.

Table 8.1 is derived populating the elements of a generic FCS (TDC, $Facet_i$, F_iTerm_j) with the corresponding elements of the "Wine" ontology example, based on the alignments to the generic structure of the Normalisation ODP discussed throughout the previous sections (:TDC, :$Module_i$, :M_iElem_j).

Table 8.1 does not include the classification element of the FCS ($Item_x$). The element $Item_x$ is populated by the element :$SpecificTDC_x$, which in the case of the "Wine" ontology are listed in Figure 8.6.

8.2.1 The Facet of "Region"

Nonetheless, it can be argued that the arrangements of terms in the "Wine" FCS of Table 8.1 is not optimal. In fact, as it stands, it clearly violates "the Principles of Homogeneity and Mutual Exclusivity" stated in Spiteri (1998) as it can be observed in the facet "Region". There are several terms of the facet "Region" that overlap, such as "France" and "Bordeaux"; "Italy" and "Chianti"; or "California" and "Santa Barbara" for example, to name a few.

In that regard, Table 8.2 presents a reviewed version of the hypothetical FCS that underlies the "Wine" ontology model example. Table 8.2 rearranges the facet "Region" to meet "the Principles of Homogeneity and Mutual Exclusivity" using once again *subfacets* in a naive and intuitive fashion, (as in the case of the facet "Topping" in the "Pizza" FCS example), to remove the overlap among facet terms.

For simplicity, (similar to the notation used for the facet "Topping" in the "Pizza" FCS

Wine Faceted Classification Scheme		
Facet		**Term**
Grape		Cabernet Franc, Cabernet Sauvignon, Chardonnay, Chenin Blanc, Gamay, Malbec, Merlot, Petite Syrah, Petite Verdot, Pinot Blanc, Pinot Noir, Riesling, Sangiovese, Sauvignon Blanc, Semillon, Zinfandel
Region	Australia	South Australia
	France	Alsace, Beaujolais, Bordeaux: {Medoc: {Pauillac, Margaux, St. Emilion}, Sauterne}, Bourgogne: {Cotes D'Or, Meursault}, Loire: {Anjou, Muscadet, Sancerre, Tours}
	Germany	
	Italy	Chianti
	New Zealand	
	Portugal	
	USA	California: {Arroyo Grande, Central Coast, Edna Valley, Mendocino, Napa, Santa Barbara, Santa Cruz Mountains, Sonoma}, Texas: {Central Texas}
Color		Red, Rose, White
Body		Full, Light, Medium
Flavor		Delicate, Moderate, Strong
Sugar		Dry, Offdry, Sweet

Table 8.2: Reviewed "Wine" FCS based on the "Wine" ontology model example.

example), only one sub-level the facet "Region" is represented in the column "Facet" of the "Wine" FCS in Table 8.2. The additional sub-levels are indicated in the column "Term", using curly-brackets such as: "Bourgogne: {Cotes D'Or, Meursault}"; or "Texas: {Central Texas}".

Reiterating the discussion in Section 7.2.1 for the facet "Topping" of the "Pizza" FCS example, in order to focus the scope of this research, the support for *subfacets* has not been considered in the current transformation guidelines between FCS and ontology put forward.

8.2.1.1 The Subsumption of "Region"

A notion that have been discussed in the Ontology Engineering community is the suitability regarding the subsumption relationship between enclosing geographical regions.

At the root of the discussion lies the principle that meronymy is not taxonomy. Meronymy denotes a "part-of" relationship between two entities, while taxonomy denotes a "kind-of" relation that in the case of "Region" from Table 8.2 is populated with subsumption. As Guarino and Welty (2002, 2009) indicate in their OntoClean methodology, a methodology to evaluate taxonomies built upon the rationale of philosophical ontology, is often difficult in ontological analysis to distinguish between the "part-of" and the "is-a" relation, but ultimately, *sumsumption is not part.*

Regarding the "Region" of "Wine", it could be argued that for example, that "Alsace" is a *part of* "France" rather than being *subsumed* by "France". This type of relation requires special consideration even by the OntoClean methodology of Guarino and Welty (2009). The authors indicate that much of the confusion is due to coupling the view of a location as both a *geographical region* and a *geopolitical entity.* Based on the notions of *rigidity*, *identity* and *unity* defined as part of the OntoClean methodoloy, the concept "Geographical Region" is subsumed by "Location", while the concept "Country" (as another name for geopolitical entity) is subsumed by "Social Entity", with "Location" and "Social Entity" being disjoint.

Therefore, in our example, the subsumption relation such as "Alsace" as a subclass of "France", is ontologically valid when both are regarded as *regions.* Subsumption may not be appropriate if the concepts of "Alsace" and "France" stood for the geopolitical entity or country they constitute. Coincidentally, Noy and McGuinness (2001)(§ 4.6) reach the same conclusion for the same example in an early version of the "Wine" ontology when discussing basic guidelines to model a given concept as either a class or an individual.

Holi and Hyvonen (2005) put forward another compelling example, the representation of *partial* conceptual overlap, where geographical regions are modeled using a subsumption classification instead of a "part-of" classification (or *partonomy*). An example of partial conceptual overlap takes place when representing the relation among the regions of Lapland, Sweden, Norway, Finland and Russia, given that the Lapland region spans throughout various parts of the other regions without covering any one of them completely.

8.3 Conclusions

The example ontology model in the domain of "Wine" used by Welty et al. (2004), is evaluated from the point of view of how it models multiple classification criteria.

The elements that conform the "Wine" ontology are examined and analyzed using a reverse engineering approach, with the goal of identifying alignments to the elements of the generic structure of the three ODPs revisited throughout this research.

The analysis reveals that the "Wine" ontology model can be seen as an instantiation of the Normalisation ODP, where the class :Wine is the :TDC element of the pattern and there are six modules or semantic axes :$Module_i$, represented by the classes: vin:WineGrape, vin:Region, vin:WineColor, vin:WineBody, vin:WineFlavor and vin:WineSugar.

However, the "Wine" ontology model deviates slightly from one of the main conditions of normalisation, because the ontology class hierarchy is not constructed entirely as a simple tree. There is one class, :Sauternes, with more than one superclass manually asserted: :Bordeaux and :LateHarvest.

This instantiation of the Normalisation ODP in the “Wine” ontology is composed of an instantiation of the Value Partition ODP, in which: (a) the feature value elements :V_iPart_j are implemented as an owl:NamedIndividual, (b) the element :TDC of the pattern is the class :Wine; and (c) there are *four* elements :$Value_i$ represented by the classes vin:WinColor, vin:WineBody, vin:WineFlavor, and vin:WineSugar.

As it occurred in the “Pizza” ontology example, not all defined classes :M_iElem_jTDC of the target domain concept of :Wine, has followed the prototypical implementation given in Definition 5.1 of the Normalisation ODP. Instead of being derived from a single element :M_iElem_j, (*single* defined classes), they combine various elements :M_iElem_j, (*compound* defined classes), to create interesting subsets of different types of wines.

Once again, this evaluation driven by the identification of classification criteria in the “Wine” ontology example, brings to the forefront the semantic axes or principles of division implicitly considered in the ontology model. With this classification criteria, a hypothetical FCS in the domain of “Wine” is proposed. This “Wine” FCS corresponds to a FCS that would result into the “Wine” ontology example if the Faceted Classification ODP from Chapter 6 was to be applied.

The creation of the “Wine” FCS from the “Wine” ontology model, suggests once more, the need to consider the support of *subfacets*, as an opportunity to improve the transformation guidelines between a FCS and an ontology model.

Chapter 9

Modeling the "Fault" Domain Concept in the ReSIST Project

This section walks through the creation from scratch of the ontological representation of the concept of "Fault" that will be part of one of the ontology models featured in the knowledge base developed for the ReSIST project. The walk through highlights how the Faceted Classification Scheme ODP introduced in Section 6 is used in this particular modeling scenario.

The examples of the "Pizza" and "Wine" concepts analysed how the respective ontologies aligned to the Normalisation ODP and thus, to a FCS. The alignment brings forward explicitly the acknowledgment of the multiple classification criteria that govern the representation of these two concepts. They also illustrate how the gap between the multiple classification criteria of a concept, the Normalisation ODP and a corresponding FCS can be bridged.

The rationale in these examples is particularly useful for the representation of "Fault". The "Fault" concept remained a daunting modeling task for weeks in terms of how to combine the different OWL elements to represent it as required by the ReSIST project. However, with the help of the design guidelines regarding multiple classification criteria presented throughout this work, the complexity of the task is notably clarified and delimited.

9.1 The Faceted Classification Scheme of "Fault"

The background information of the "Fault" concept to be considered for the ReSIST Knowledge Base is captured in Avizienis et al. (2004). Figures 1.1, 1.2 and 1.3 extracted from the cited reference and presented in Section 1.1, provide an informative and complete graphical summary of the various terms in the various classification criteria that form the concept of “Fault”, for which an ontological representation is to be developed.

As Section 1.1 explains, the background information considered, suggests multiple classification criteria for the concept of “Fault”. From Figure 1.2 alone, faults can be classified according to the 8 basic fault viewpoints that they are made of, or according to certain known categories of fault examples, or even according to three known major partially overlapping groups. An additional criterion includes 31 type of faults, corresponding the 31 likely combinations of the 8 basic fault viewpoints mentioned earlier. As detailed in Avizienis et al. (2004), the 2 mutual exclusive values in each of the 8 basic fault viewpoints could lead to a 256 different combinations of fault types, however only the 31 featured in Figure 1.2 are likely or feasible to occur in the real world.

Nonetheless, there is an aspect of the classification criteria of “Fault” that does not occur in the case of the “Pizza”, “Wine”, or “Dish Detergent” examples. In the case of “Fault”, the elements that form the 8 basic fault viewpoints, can also be used to determine the elements in the rest of the classification criteria identified. For example, consider “Fault Type 1”, one of the elements in the “31 Likely Combinations of Faults” classification criterion at the bottom left of Figure 1.2. “Fault Type 1” is determined by a specific combination of elements in the “8 Basic Fault Viewpoint” criterion, namely: {Development, Internal, Human-made, Software, Non-malicious, Non-deliberate, Accidental, Permanent}.

A similar situation takes place in the case of the elements that form the “Known Examples of Fault” classification criterion. Consider for instance, the category “Software Flaw” at the bottom left of Figure 1.2. “Software Flaw” is the combination of four of the 31 likely fault types, more specifically: {Fault Type 1, Fault Type 2, Fault Type 3, Fault Type 4}. In turn, each of these 4 types can be expressed in terms of the 8 basic viewpoint classification criteria. Thus, the “Software Flaw” concept can be expressed as the following combination of elements from the “8 Basic Fault Viewpoint” criterion by means of expanding the four fault types that it is made of: {Development, Internal, Human-made, Software, Non-malicious, (Non-deliberate or Deliberate), (Accidental or Incompetence), Permanent}.

This dependency of all elements in the rest of classification criteria with respect to the “8 Basic Fault Viewpoint” criterion, indicates that the latter suffice to represent the entire universe of discourse of the “Fault” modeling scenario at hand. The pair of mutually exclusive terms in each viewpoint, exhaust all the instances of faults to be considered for the ReSIST Knowledge Base. This characteristic of each viewpoint aligns to the various definitions of a Faceted Classification Scheme recapped in Section 6 from Denton (2003) and Spiteri (1998). In essence, a facet in a given domain, corresponds to a single principle of division of the parent universe. The facet is formed by a set of mutually exclusive and jointly exhaustive categories. Facets combine to completely describe all objects in

Fault FCS	
Facet	**Term**
Phase of Creation	Development, Operational
System Boundary	Internal, External
Phenomenological Cause	Natural, Human-made
Dimension	Hardware, Software
Objective	Malicious, Non-malicious
Intent	Deliberate, Non-deliberate
Capability	Accidental, Incompetence
Persistence	Permanent, Transient

Table 9.1: “Fault” FCS.

the domain of discourse.

Based on this definition of facet, each one of the viewpoints of “Fault” can be considered a facet of a prospective Faceted Classification Scheme of the “Fault” domain in the ReSIST Knowledge Base. Using the principles of faceted classification outlined in Denton (2003) and Spiteri (1998), the following FCS of “Fault” is presented in Table 9.1.

Note the correlation between the terms presented in Figure 1.1 in Section 1.1 (the elementary fault classes) and the proposed “Fault” FCS Table 9.1. Each elementary fault viewpoint in Figure 1.1 becomes a facet in Table 9.1, and the elements of each viewpoint become the terms of each facet.

The sections that follow, go through the construction of the ontology model of “Fault” for the ReSIST Knowledge Base, applying the proposed guidelines introduced in Section 6, to convert an existing FCS into a normalised OWL DL ontology model.

9.2 Structure of the “Fault” Ontology

Provided the Faceted Classification Scheme of “Fault” presented in Table 9.1, the rationale to build the corresponding ontology model is sustained on the guidelines proposed throughout Chapter 6. More specifically:

- Section 6.1 detailed the generic structure of a FCS (Definition 6.1) and how a given example in the domain of “Dish Detergent” fits into that structure (Example 6.1, Table 6.1).
- Section 6.2 detailed in Table 6.2 the alignment between a generic FCS and the Normalisation ODP, producing the generic structure of the Faceted Classification ODP introduced in Figure 6.1. The complete process is recapped in Figure 6.2.

- Section 6.2 also illustrates the FCS to Normalisation alignment with the example FCS of "Dish Detergent" (Figures 6.3, 6.4 and 6.5). The illustration includes examples of the representation of some elements from the "Dish Detergent" domain of discourse to be classified (Figure 6.6).

In the case of the "Fault" concept, the ontology model to be built, is going to be determined by following an analogue procedure to that in Chapter 6 using the "Fault" Faceted Classification Scheme presented in Table 9.1. The initial step involves populating the generic elements of the Faceted Classification ODP (:$Facet$, :F_iTerm_j, :TDC, :F_iTerm_jTDC, :$SpecificTDC_x$, :$hasFacet_i$), with the appropriate elements of the "Fault" FCS in Table 9.1.

9.2.1 Facets (Modules) and Facet Terms

The element :$Facet_i$ in the FCS ODP generic structure is populated as an owl:Class with each of the facets identified for the "Fault" FCS in Table 9.1. In addition, the elements :F_iTerm_j are populated with the corresponding terms identified for each facet :$Facet_i$ element from the same "Fault" FCS table.

There is another aspect regarding the elements :$Facet_i$ and :F_iTerm_j to consider in the design of the "Fault" ontology. That is the fact that the two terms :F_iTerm_j of each facet :$Facet_i$ in the "Fault" FCS represent a value partition for the facet, aligning to the generic structure of the Value Partition ODP. In line with the pattern, the elements :F_iTerm_j can be represented as an owl:Class or as an owl:NamedIndividual. Both options are fit-for-purpose in the case of "Fault", however, as Rector (2005) highlights in the overview of both versions, using the owl:Class representation allows for further refinements of the partition classes if necessary. In other words, if the elements :F_iTerm_j that partition a given :$Facet_i$ are represented as an owl:Class, new classes can be added at a later point in time as subclasses of :F_iTerm_j, refining the partition further. Otherwise, if the elements :F_iTerm_j are represented as owl:NamedIndividual, this refinement is not possible.

For the purpose of this example, the elements :F_iTerm_j are represented as an owl:Class to provide the "Fault" ontology model with this maintenance flexibility. Following the requirements of the Value Partition ODP, the element :$Facet_i$ is implemented as a defined owl:Class equivalent to the union of the mutually disjoint classes {:F_iTerm_1, :F_iTerm_2}. As an example, consider Listing 9.1, which implements the "Fault" facet "Dimension" and its two terms "Hardware" and "Software" applying the Value Partition ODP.

Figure 9.1 illustrates the ontology structure that results after all these previous assignments are in place. Note from the figure the inclusion of the class :FaultViewpoint,

```
:Dimension rdf:type owl:Class ;
           owl:equivalentClass [ rdf:type owl:Class ;
                                 owl:unionOf ( :Hardware
                                               :Software
                                             )
                               ] ;
           rdfs:subClassOf :FaultViewpoint .
```

Listing 9.1: Implementation of the “Fault” facet “Dimension”

```
owl:Thing                                          (FCS ODP Generic Structure)
  |-- :FaultViewpoint
     |-- (≡)(P) :PhaseOfCreation                   (:Facet_1)
        |-- :Development                              (:F_1Term_1)
        |-- :Operational                              (:F_1Term_2)
     |-- (≡)(P) :SystemBoundary                    (:Facet_2)
        |-- :Internal                                 (:F_2Term_1)
        |-- :External                                 (:F_2Term_2)
     |-- (≡)(P) :PhenomenologicalCause             (:Facet_3)
        |-- :Natural                                  (:F_3Term_1)
        |-- :HumanMade                                (:F_3Term_2)
     |-- (≡)(P) :Dimension                         (:Facet_4)
        |-- :Hardware                                 (:F_4Term_1)
        |-- :Software                                 (:F_4Term_2)
     |-- (≡)(P) :Objective                         (:Facet_5)
        |-- :Malicious                                (:F_5Term_1)
        |-- :NonMalicious                             (:F_5Term_2)
     |-- (≡)(P) :Intent                            (:Facet_6)
        |-- :Deliberate                               (:F_6Term_1)
        |-- :NonDeliberate                            (:F_6Term_2)
     |-- (≡)(P) :Capability                        (:Facet_7)
        |-- :Accidental                               (:F_7Term_1)
        |-- :Incompetence                             (:F_7Term_2)
     |-- (≡)(P) :Persistence                       (:Facet_8)
        |-- :Permanent                                (:F_8Term_1)
        |-- :Transient                                (:F_8Term_2)
```

Figure 9.1: Representation of the facets and facet terms in the “Fault” ontology model for the “Fault” FCS.

that is not present in the initial “Fault” FCS. According to the Faceted Classification Scheme ODP design guidelines, the :$Facet_i$ elements are subsumed by owl:Thing by default. However in this case, the class :FaultViewpoint is introduced to limit the semantic range of class names so broad in meaning such as “Dimension”, “Objective”, “Intent”, etc. (the facets). The class :FaultViewpoint allows to keep the meaning of the facets that it subsumes in the domain intended for “Fault”.

9.2.2 Object Properties

The element :$hasFacet_i$ is populated with an object property for each facet in the “Fault” FCS. In other words, for each :$Facet_i$ class. Figure 9.2 illustrates the object properties required to apply the Faceted Classification Scheme ODP to “Fault”.

```
owl:topObjectProperty                   (rdfs:domain)  (rdfs:range)             (FCS ODP)
  |-- :hasFaultViewPoint                :Fault         :FaultViewPoint
     |-- :hasPhaseOfCreation            :Fault         :PhaseOfCreation         (:hasFacet1)
     |-- :hasSystemBoundary             :Fault         :SystemBoundary          (:hasFacet2)
     |-- :hasPhenomenologicalC...       :Fault         :PhenomenologicalC...    (:hasFacet3)
     |-- :hasDimension                  :Fault         :Dimension               (:hasFacet4)
     |-- :hasObjective                  :Fault         :Objective               (:hasFacet5)
     |-- :hasIntent                     :Fault         :Intent                  (:hasFacet6)
     |-- :hasCapability                 :Fault         :Capability              (:hasFacet7)
     |-- :hasPersistence                :Fault         :Persistence             (:hasFacet8)
```

Figure 9.2: Object Properties in the "Fault" ontology model for the "Fault" FCS.

The object property :hasFaultViewpoint is introduced to subsume all the :$hasFacet_i$ object properties for the analogous motivation to its owl:Class counterpart :FaultViewpoint, and to be consistent with the :$Facet_i$ class hierarchy.

9.2.3 Target Domain Concept

This section documents how the target domain concept element :TDC, and the two elements :F_iTerm_jTDC and :$SpecificTDC_x$ that it subsumes, are populated for the "Fault" ontology model from the "Fault" FCS in Table 9.1.

The element :TDC is obviously populated by the owl:Class :Fault. Let us explore the defined classes of :Fault (:F_iTerm_jTDC) and the classification elements of :Fault (:$SpecificTDC_x$).

9.2.3.1 Defined Classes

***Single* Defined Classes** The defined classes in this group are associated to the main 8 classification criteria of the "Fault" concept, which in turn, correspond to the 8 facets identified in the "Fault" FCS.

The representation of the defined classes presented in Figure 9.1 is straight-forward following: (a) the alignment outlined by the Faceted Classification Scheme ODP between the generic elements :F_iTerm_jTDC and :M_iElem_jTDC in Section 6.2.2.1; and (b) the generic implementation of :M_iElem_jTDC given in Definition 5.1.

For example, the element :F_1Term_1 populated with the class :Development (Figure 9.1) is associated to element :F_1Term_1TDC that would be populated with the defined class :DevelopmentFault. The element :F_1Term_2 populated with the class :Operational (Figure 9.1) is associated to element :F_1Term_2TDC that would be populated with the defined class :OperationalFault. The process repeats for every element :F_iTerm_j in Figure 9.1.

```
:HardwareFault rdf:type owl:Class ;
               owl:equivalentClass [ rdf:type owl:Restriction ;
                                     owl:onProperty :hasDimension ;
                                     owl:someValuesFrom :Hardware
                                   ] ;
               rdfs:subClassOf :ElementalFault .
```

Listing 9.2: Implementation of the “Fault” defined class :HardwareFault

Figure 9.3 illustrates all the defined classes $:F_iTerm_jTDC$ subsumed by the target domain concept in the “Fault” ontology model. Note the introduction of the owl:Class :ElementalFault in Figure 9.3. The class :ElementalFault is introduced to distinguish the defined classes that are associated to a single element $:F_iTerm_j$ of the “8 Basic Fault Viewpoint” criterion, from other types of defined classes contained in the ontology that will be presented below.

As an implementation example, consider the representation of the defined class :HardwareFault ($:F_4Term_1TDC$) associated to the class :Hardware ($:F_4Term_1$) of the facet :Dimension ($:Facet_4$), given in Listing 9.2.

***Compound* Defined Classes** The main difference between the defined classes supplied by the “8 Basic Fault Viewpoints” classification criteria (the classes subsumed by :ElementalFault in Figure 9.3), and the defined classes supplied by the “31 Likely Combinations of Fault” and the “Known Examples of Fault” criteria is that each defined class from the first criterion is associated to a single facet term element ($:F_iTerm_j$), while each defined class from the second and third criteria is associated to many facet term elements ($:F_iTerm_j$).

The Faceted Classification ODP introduced in Chapter 6 only address the creation of defined classes ($:F_iTerm_jTDC$) directly related to a single facet term element in the FCS ($:F_iTerm_j$). These are the defined classes covered in the previous section “Single Defined Classes”. However as the examples of “Pizza” and “Wine” have showed, many defined classes subsumed by the target domain concept :Pizza and :Wine respectively, are related to multiple elements from the various modules in the ontology model.

This aspect of the ontology modeling can be extrapolated to the field of Faceted Classification. The combination of multiple facet terms to classify a particular element is referred to as a “compound”. Compound subjects are in essence the main motivation for using a Faceted Classification Scheme. The defined classes presented in the “Compound Defined Classes” section of the “Pizza” and “Wine” examples (Sections 7.1.3.1 and 8.1.3.1 respectively), can be seen as the representation of a “compound” in terms of a FCS in an OWL ontology model.

In the case of “Fault”, there are two classification criteria, namely the “Known Examples of Fault” and the “31 Likely Combinations of Fault”, whose elements depend on a series

owl:Thing (*FCS ODP Generic Structure*)
  |-- :Fault (:TDC)
    |-- :ElementalFault
      |-- (≡) :DevelopmentFault (:F_1Term_1TDC)
      |-- (≡) :OperationalFault (:F_1Term_2TDC)
      |-- (≡) :InternalFault (:F_2Term_1TDC)
      |-- (≡) :ExternalFault (:F_2Term_2TDC)
      |-- (≡) :NaturalFault (:F_3Term_1TDC)
      |-- (≡) :HumanMadeFault (:F_3Term_2TDC)
      |-- (≡) :HardwareFault (:F_4Term_1TDC)
      |-- (≡) :SoftwareFault (:F_4Term_2TDC)
      |-- (≡) :MaliciousFault (:F_5Term_1TDC)
      |-- (≡) :NonMaliciousFault (:F_5Term_2TDC)
      |-- (≡) :DeliberateFault (:F_6Term_1TDC)
      |-- (≡) :NonDeliberateFault (:F_6Term_2TDC)
      |-- (≡) :AccidentalFault (:F_7Term_1TDC)
      |-- (≡) :IncompetenceFault (:F_7Term_2TDC)
      |-- (≡) :PermanentFault (:F_8Term_1TDC)
      |-- (≡) :TransientFault (:F_8Term_2TDC)
    |-- :ExampleOfFault
      |-- (≡) :SoftwareFlawFault (related to multiple :F_iTerm_j)
      |-- (≡) :LogicBombFault (related to multiple :F_iTerm_j)
      |-- (≡) :HardwareErrataFault (related to multiple :F_iTerm_j)
      |-- (≡) :ProductionDefectFault (related to multiple :F_iTerm_j)
      |-- (≡) :PhysicalDeteriorationFault (related to multiple :F_iTerm_j)
      |-- (≡) :PhysicalInterferenceFault (related to multiple :F_iTerm_j)
      |-- (≡) :IntrusionAttemptFault (related to multiple :F_iTerm_j)
      |-- (≡) :VirusesAndWormFault (related to multiple :F_iTerm_j)
      |-- (≡) :InputMistakeFault (related to multiple :F_iTerm_j)
    |-- :CombinedFault
      |-- (≡) :CombinedFault1 (related to multiple :F_iTerm_j)
      |-- (≡) :CombinedFault2 (related to multiple :F_iTerm_j)
      |-- (... rest of likely Combined Fault
            Type defined classes)
      |-- (≡) :CombinedFault31 (related to multiple :F_iTerm_j)

Figure 9.3: Representation of the defined classes in the "Fault" ontology model for the "Fault" FCS.

of combinations of the main 8 principles of divisions. These two classification criteria will contribute their own set of defined classes, or in other words, the "Compound Defined Classes" of the target domain concept :Fault.

The 31 Combinations of Fault As covered in Avizienis et al. (2004) and depicted in Figure 1.2, each one of the 31 likely combinations of faults is made of a unique selection of elements from each one of the 8 basic viewpoints of "Fault". Let us name these 31 defined classes such as: :CombinedFault1, :CombinedFault2, ..., :CombinedFault31. The implementation of :CombinedFault1 for example, is presented in Listing 9.3.

The implementation of :CombinedFault1 reflects that this defined class is associated to multiple elements :F_iTerm_j, more specifically to the set: {:Accidental, :Software, :NonDeliberate, :NonMalicious, :Permanent, :Development, :HumanMade, :Internal}.

The implementation of the rest of combined faults is analogous to that of :Combined-

```
:CombinedFault1 rdf:type owl:Class ;
          rdfs:subClassOf :CombinedFault ;
          owl:equivalentClass [ rdf:type owl:Class ;
                                owl:intersectionOf
                                    ( [ rdf:type owl:Restriction ;
                                        owl:onProperty :hasCapability ;
                                        owl:someValuesFrom :Accidental
                                      ]
                                      [ rdf:type owl:Restriction ;
                                        owl:onProperty :hasDimension ;
                                        owl:someValuesFrom :Software
                                      ]
                                      [ rdf:type owl:Restriction ;
                                        owl:onProperty :hasIntent ;
                                        owl:someValuesFrom :NonDeliberate
                                      ]
                                      [ rdf:type owl:Restriction ;
                                        owl:onProperty :hasObjective ;
                                        owl:someValuesFrom :NonMalicious
                                      ]
                                      [ rdf:type owl:Restriction ;
                                        owl:onProperty :hasPersistence ;
                                        owl:someValuesFrom :Permanent
                                      ]
                                      [ rdf:type owl:Restriction ;
                                        owl:onProperty :hasPhaseOfCreation ;
                                        owl:someValuesFrom :Development
                                      ]
                                      [ rdf:type owl:Restriction ;
                                        owl:onProperty :hasPhenomenologicalCause ;
                                        owl:someValuesFrom :HumanMade
                                      ]
                                      [ rdf:type owl:Restriction ;
                                        owl:onProperty :hasSystemBoundary ;
                                        owl:someValuesFrom :Internal
                                      ]
                                    )
                              ] .
```

Listing 9.3: Implementation of the “Fault” defined class :CombinedFault1

Fault1 in Listing 9.3 applying the corresponding unique combination of elements :F_iTerm_j that can be drawn from Figure 1.2.

Figure 9.3 also illustrates the defined classes from the “31 Combinations of Fault” classification criterion as they will be represented in the “Fault” ontology model. Note once again that another intermediate class, :CombinedFault, is introduced to distinguish the defined classes from this criterion from the rest of the defined classes of :Fault rooted into other criteria.

The Known Examples of Fault The “Known Examples of Fault” classification criterion is comprised of the 9 fault categories identified on the bottom of Figure 1.2 and it is the source of 9 additional defined classes subsumed by the “Fault” target domain concept. At the same time, each known example of fault is formed by a group

```
:SoftwareFlawFault rdf:type owl:Class ;
                   rdfs:subClassOf :ExemplaryFault ;
                   owl:equivalentClass [ rdf:type owl:Class ;
                                         owl:unionOf ( :CombinedFault1
                                                       :CombinedFault2
                                                       :CombinedFault3
                                                       :CombinedFault4
                                                     )
                                       ] .
```

Listing 9.4: Implementation of the "Fault" defined class :SoftwareFlawFault (option 1)

of combined faults from the "31 Combination of Fault" classification criterion. For example, as Figure 1.2 shows, the category "Software Flaws" is formed by the union of the combined faults "Fault 1", "Fault 2", "Fault 3", and "Fault 4".

This relationship between both classification criteria provides two equivalent options to implement an element from the "Known Examples of Fault" criterion: (a) in terms of elements from the "31 Combination of Fault" and (b) in terms of elements from the "8 Basic Fault Viewpoints".

Using "Software Flaw" as an example again, Listing 9.4 presents the implementation of the defined class :SoftwareFlawFault in terms of elements from the "31 Combination of Fault", while Listing 9.5 presents the same definition implemented in terms of elements from the '8 Basic Fault Viewpoints".

The defined classes from the "Known Examples of Fault" classification criterion are also displayed in Figure 9.3 as they will appear in the "Fault" ontology model. Similar to the previous cases, the intermediate class :ExampleOfFault, is introduced to separate the defined classes from this criterion from the rest of defined classes subsumed by :Fault.

The implementation of the rest of the 9 known examples of fault is analogous to that of :SoftwareFlawFault in Listing 9.3 applying: (a) the corresponding combinations of elements from the "31 Combination of Fault"; or (b) the corresponding unique combination of elements :F_iTerm_j from the "8 Basic Fault Viewpoints". Either option can be drawn from Figure 1.2.

Sources of Defined Classes In the case of "Fault", the contribution of defined classes (single and compound) from the different facets (or modules in normalisation terms), is fairly equally distributed across the 8 facets represented in the "Fault" ontology model. This can be easily observed from the links between the elements :F_iTerm_j associated to the "8 Basic Fault Viewpoints" indicated in Figure 9.1 and (a) the elements :F_iTerm_jTDC associated to the single defined classes indicated in Figure 9.3; and (b) how the compound defined classes also in Figure 9.3, are determined by the various combinations of these :F_iTerm_j elements.

```
:SoftwareFlawFault rdf:type owl:Class ;
                   rdfs:subClassOf :ExemplaryFault ;
                   owl:equivalentClass
                       [ rdf:type owl:Class ;
                         owl:intersectionOf
                             ( [ rdf:type owl:Restriction ;
                                 owl:onProperty :hasCapability ;
                                 owl:someValuesFrom :Capability
                               ]
                               [ rdf:type owl:Restriction ;
                                 owl:onProperty :hasDimension ;
                                 owl:someValuesFrom :Software
                               ]
                               [ rdf:type owl:Restriction ;
                                 owl:onProperty :hasIntent ;
                                 owl:someValuesFrom :Intent
                               ]
                               [ rdf:type owl:Restriction ;
                                 owl:onProperty :hasObjective ;
                                 owl:someValuesFrom :NonMalicious
                               ]
                               [ rdf:type owl:Restriction ;
                                 owl:onProperty :hasPersistence ;
                                 owl:someValuesFrom :Permanent
                               ]
                               [ rdf:type owl:Restriction ;
                                 owl:onProperty :hasPhaseOfCreation ;
                                 owl:someValuesFrom :Development
                               ]
                               [ rdf:type owl:Restriction ;
                                 owl:onProperty :hasPhenomenologicalCause ;
                                 owl:someValuesFrom :HumanMade
                               ]
                               [ rdf:type owl:Restriction ;
                                 owl:onProperty :hasSystemBoundary ;
                                 owl:someValuesFrom :Internal
                               ]
                             )
                       ] .
```

Listing 9.5: Implementation of the "Fault" defined class :SoftwareFlawFault (option 2)

9.3 "Fault" As Property Value

As Section 1.1.3 indicated, the ReSIST project intended several uses of the "Fault" ontology model including: (a) the representation and classification of instances of faults in real world systems; and (b) as a terminology or keyword index for publications, projects, research interests and the resilient mechanisms of computer systems.

The "Fault" ontology model developed in the previous sections addressed the needs expressed in *Scenario (a)*, by applying the Faceted Classification ODP. The needs described in *Scenario (b)* however, suggest the use of the :Fault class hierarchy in the 'Fault" ontology model to annotate other entities as part of a larger ontology model such as: "Publication", "Project", "Research Interest" (for people or institutions for example), or "Resilient Mechanism".

Scenario (b) is very similar to the example in the Class As Property Value ODP reviewed in Chapter 3, where the role of the :Animal class hierarchy is similar to the role of :Fault in (b) and the role of the :Book target domain concept is similar to that of the concepts such as "Publication", "Project", "Research Interest", etc. In essence, *Scenario (b)* is calling for multiple interpretations of the :Fault class hierarchy used as a property value.

There is another distinction that it is important to note between these two scenarios. In *Scenario (a)*, as discussed in the previous sections, "Fault" performs the function of the :TDC element in the generic structure of the Faceted Classification ODP (Figure 6.1), while in *Scenario (b)* as the next section will discuss, "Fault" performs the function of the :$Terminology_i$ element in the generic structure of the Class As Property Value ODP (Figure 3.2).

9.3.1 Multiple Interpretation of "Fault" in ReSIST

To illustrate an example of the multiple interpretations of the :Fault class hierarchy, consider the generic structure of the Class As Property Value ODP in Figure 3.2.

The element :$Interpretation_a$ can be populated with: (a) the class :Subject as :$Interpretation_1$ to represent the interpretation of :Fault as the subject of a publication; (b) the class :Interest as :$Interpretation_2$ to represent the interpretation of :Fault as the research interest of people or projects; and (c) the class :Resilience as :$Interpretation_3$ to represent the interpretation of :Fault as the target of the resilience mechanism of a computer system.

According to the generic structure of the pattern, there is an object property :$hasInterpretation_a$ for each :$Interpretation_a$ element, therefore the following object properties will be created: (a) :hasSubject as :$hasInterpretation_1$ (b) :hasInterest as :$hasInterpretation_2$; and (c) :hasResilience as :$hasInterpretation_3$.

There is a different :TDC element for each :$Interpretation_a$, namely (a) :Publication as :TDC_1 for the :Subject interpretation of :Fault; (b) :Person as :TDC_2 for the :Interest interpretation; and (c) :ComputerSystem as :TDC_3 for the :Resilience interpretation.

Figure 9.4 portrays a partial ontology model illustrating the multiple interpretations of "Fault". The figure includes the following elements:

- The :Fault class hierarchy developed throughout this Chapter as presented in Figure 9.3. Only a few classes of the total hierarchy (those needed for the example) are included. Note that in this case, the class :Fault aligns to the :$Terminology_i$ element of the Class As Property Value pattern, while in Figure 9.3 aligns to the :TDC element of the Faceted Classification ODP.

- The 3 interpretations of the :Fault class hierarchy (:$Interpretation_a$), namely :Subject, :Interest and :Resilience. The 3 interpretations are *not* explicitly asserted in the model (denoted by the symbol "(I)" and *italics* in the figure), although if they were, they all would subsume the class :Fault.

- The 3 target domain concepts (:TDC_i) that require the multiple interpretation of the :Fault class hierarchy, namely :Publication, :Person, and :ComputerSystem.

- A sample of defined classes for each target domain concept, (the generic element :$I_aT_iClass_j...TDC$ in the Class As Property Value ODP). The definition of the defined classes is based on the classes that belong to the :Fault class hierarchy.

- A sample of individuals for each target domain concept, (the generic element :$SpecificTDC_x$ in the Class As Property Value ODP). More specifically: (a) :Avizienis2004 to represent the specific publication Avizienis et al. (2004); (b) :AvizienisA and :LaprieJC to represent two specific authors of the publication Avizienis et al. (2004); and (c) :RKB Explorer to represent a specific computer system.

- The 3 object properties that correspond to the 3 interpretations respectively: :hasSubject, :hasInterest and :hasResilience. Note the rdfs:domain and rdfs:range of these properties. The rdfs:domain of the object property is the applicable :TDC, while the rdfs:range for all 3 of them is the class :Fault.

Th extended ontology model of "Fault" that Figure 9.4 partially exhibits, presents several interesting characteristic:

- The "Fault" ontology model is normalised in terms of the Normalisation ODP with respect to the :Fault target domain concept.

- The overall structure of the "Fault" ontology model in Figure 9.1, Figure 9.2 and Figure 9.3, represent one instantiation of the Normalisation ODP with respect to the :Fault target domain concept that includes several instantiations of the Value Partition ODP (one per facet represented).

- The extended ontology in Figure 9.4 is normalised in terms of the Normalisation ODP with respect to the :Publication, :Person and :ComputerSystem target domain concepts.

- The structure of the extended ontology model in Figure 9.4 represent one instantiation of the Normalisation ODP for each one of the target domain concepts represented (:Publication, :Person, :ComputerSystem) that includes one instantiation of the Class As Property Value ODP with respect to the :Fault class hierarchy.

`owl:Thing`	$(Class\ Prop.\ Value\ ODP)$
`|-- (I) :`*Subject* `or`	$(:Interpretation_1$ or
`    (I) :`*Interest* `or`	$:Interpretation_2$ or
`    (I) :`*Resilience*	$:Interpretation_3)$
`  |-- :Fault`	$(:Terminology_1)$
`    |-- :ElementalFault`	$(:T_1Class_1)$
`      |-- (≡) :XxxElementalFault`	$(:T_1Class_1Class_k)$
`      |-- (≡) :HardwareFault`	$(:T_1Class_1Class_7)$
`    |-- :ExampleOfFault`	$(:T_1Class_2)$
`      |-- (≡) :SoftwareFlawFault`	$(:T_1Class_2Class_1)$
`      |-- (≡) :XxxExampleOfFault`	$(:T_1Class_2Class_k)$
`|-- :Publication`	$(:TDC_1)$
`  |-- (≡) :PubAboutFault`	$(:Int_1Terminology_1TDC)$
`  |-- (≡) :PubAboutElementalFault`	$(:Int_1T_1Class_1TDC)$
`  |-- (≡) :PubAboutHardwareFault`	$(:Int_1T_1Class_1Class_7TDC)$
`  |-- (≡) :PubAboutExampleOfFault`	$(:Int_1T_1Class_2TDC)$
`  |-- (≡) :PubAboutSoftwareFlawFault`	$(:Int_1T_1Class_2Class_1TDC)$
`  |-- (≡) :PubAboutXxxFault`	$(:Int_1T_1Class_jClass_kTDC)$
`  |-- (◊) :Avizienis2004`	$(:SpecificTDC_1$ of $TDC_1)$
`|-- :Person`	$(:TDC_2)$
`  |-- (≡) :PersonInterestFault`	$(:Int_2Terminology_1TDC)$
`  |-- (≡) :PersonInterestElementalFault`	$(:Int_2T_1Class_1TDC)$
`  |-- (≡) :PersonInterestHardwareFault`	$(:Int_2T_1Class_1Class_7TDC)$
`  |-- (≡) :PersonInterestExampleOfFault`	$(:Int_2T_1Class_2TDC)$
`  |-- (≡) :PersonInterestSoftwareFlawFault`	$(:Int_2T_1Class_2Class_1TDC)$
`  |-- (≡) :PersonInterestXxxFault`	$(:Int_2T_1Class_jClass_kTDC)$
`  |-- (◊) :AvizienisA`	$(:SpecificTDC_1$ of $TDC_2)$
`  |-- (◊) :LaprieJC`	$(:SpecificTDC_2$ of $TDC_2)$
`|-- :ComputerSystem`	$(:TDC_3)$
`  |-- (≡) :CompSysResiliantFault`	$(:Int_3Terminology_1TDC)$
`  |-- (≡) :CompSysResiliantElementalFault`	$(:Int_3T_1Class_1TDC)$
`  |-- (≡) :CompSysResiliantHardwareFault`	$(:Int_3T_1Class_1Class_7TDC)$
`  |-- (≡) :CompSysResiliantSoftwareFlawFault`	$(:Int_3T_1Class_2Class_1TDC)$
`  |-- (≡) :CompSysResiliantExampleOfFault`	$(:Int_3T_1Class_2TDC)$
`  |-- (≡) :CompSysResiliantXxxFault`	$(:Int_3T_1Class_jClass_kTDC)$
`  |-- (◊) :RKBExplorer`	$(:SpecificTDC_1$ of $TDC_3)$

`owl:topObjectProperty`	$(rdfs{:}domain)$	$(rdfs{:}range)$	$(Class\ Prop.\ Value\ ODP)$
`|-- :hasSubject`	`:Publication`	`:Fault`	$(:hasInterpretation_1)$
`|-- :hasInterest`	`:Person`	`:Fault`	$(:hasInterpretation_2)$
`|-- :hasResilience`	`:ComputerSystem`	`:Fault`	$(:hasInterpretation_3)$

Figure 9.4: Placement of the multiple interpretations of Fault in the Class As Property Value ODP generic structure in Figure 3.2.

This composition of several instantiations of the Normalisation ODP using the Value Partition and the Class As Property Value ODPs could be seen as a *nested normalisation* mechanism.

9.4 Conclusions

This Chapter has illustrated the application from scratch of the Faceted Classification ODP introduced in Chapter 6, to the modelling of the "Fault" domain concept for the D&S ontology of the ReSIST project, as per the requirements summarised in Section 1.1.3.

The background knowledge of the concept of "Fault" from Avizienis et al. (2004) characterizes "Fault" as a concept subject to multiple classification criteria. Based on the classification criteria of "Fault" a FCS is built using the simplified methodology of facet analysis and faceted classification of Spiteri (1998) and Denton (2003). The proposed "Fault" FCS is transformed into a normalised OWL DL ontology model applying the transformation guidelines outlined in the reengineering Faceted Classification ODP.

The normalised OWL DL "Fault" ontology model delivered, features *single* defined classes derived from a single facet term of the source "Fault" FCS, but also *compound* defined classes derived from the combination of multiple facet terms to represent additional relevant classification criteria.

The "Fault" ontology model built from this endeavor is available online[1] in N3 Turtle format and provides an answer to *Research Question 1* in Section 1.4, corresponding to the first of the intended uses of "Fault" for ReSIST: *Scenario (a)* of Section 1.1.3.

Unfortunately, due to various research eventualities beyond anyone's control, the final version of the "Fault" ontology model as developed by this research, did not ultimately come to fruition by the delivery deadline that the ReSIST project was subject to. Therefore, it is not featured as part of the ontology collection that conforms the ReSIST RKB Explorer[2] semantic web portal application.

There is a second contribution that this Chapter has illustrated regarding the modelling of "Fault". The application of the most generic version of the Class As Property Value ODP from Chapter 3, in which there is a separate notion of interpretation and terminology, provides an answer to *Research Question 2* in Section 1.4, corresponding to the second of the intended uses of "Fault" for ReSIST: *Scenario (b)* of Section 1.1.3.

[1] `http://purl.oclc.org/ecs.soton.ac.uk/project/resist/ontology/fault_fcs_norm`
[2] `http://www.rkbexplorer.com/`

Chapter 10

Conclusions and Future Work

10.1 Conclusions

10.1.1 Once Upon a Time There Was a Challenge

The work of this thesis focused on the practical modeling of multiple classification criteria of ontology domain concepts in the context of the Semantic Web (Section 1.2). The introduction and motivation of this research have stressed how recurrent it is to find domain concepts that are naturally represented according to multiple classification criteria and the more than likely relationship to multiple inheritance. Examples include concepts as common as "Pizza" (Chapter 7), "Wine" (Chapter 8), "Dishwashing Detergent" (Chapter 6), or a "Fault" in a computer system (Chapter 9) to name a few. The task that originally motivated this research problem, required building an ontological representation from scratch of one of such concepts, the concept of "Fault", meeting the use case scenarios required by the RKB Explorer application of the ReSIST project (*Scenario (a)* and *(b)* in Section 1.1.3).

A review of existing practices relevant to the modeling of multiple classification criteria has been conducted in order to identify consistent guidelines to build an ontology model from scratch for this particular recurrent modeling scenario. The review includes the fields of Ontology Engineering, more specifically Ontology Design Patterns, an analysis of multiple inheritance in Object-Oriented Design and lastly, Faceted Classification in Library and Information Science (Chapter 2). The outcome indicates a lack of explicit guidelines in the ontology development literature for the scenario described, leaving ample room for ad-hoc practices that can lead to unexpected or undesired results in ontology artifacts.

In fact, a constant ongoing effort in Ontology Engineering is to harness the field with sound practices to mitigate the opportunity for harmful ad-hoc practices. To assist Ontology Engineering with this effort, a series of intermediate contributions (Chapter 3, 4, and 5) have been put forward together with a simple systematic and consistent ontology construction guideline (Chapter 6), to provide a partial solution to the problem under consideration in the context of *Research Question 1* and *2* in Section 1.4.

10.1.2 Patterns vs. Patterns

In Object-Oriented Design, the modelling scenario of nested generalisation is reviewed. Nested generalisation is one the possible materialisations of multiple classification criteria in the domain being modeled, and it leads to solutions involving the use of multiple inheritance. Two object-oriented design patterns to address nested generalisation are discussed, the Bridge Pattern and View Inheritance, and they are also examples of what it is referred to as faceted-oriented design (Section 2.3).

The analysis of multiple inheritance in Object-Oriented Design, the modeling scenario of nested generalisation, the generic structure of the Bridge Pattern and View Inheritance; and the notion of faceted-oriented design, proved to be valuable contributions in terms of what to look for in the other two areas being explored: Ontology Engineering and Faceted Classification.

In Ontology Engineering the Class As Property Value, the Value Partition and the Normalisation ODPs are revisited. These three patterns are shortlisted based on their applicability to the research problem in question, from a joint repository of ODPs formed by the two known ODP catalogues publicly available. A graphical notation is introduced (Section 3.1) that allows one to compare graphically different templates and instantiations of these ODPs. Additionally, a generic structure for each ODP is put forward using this visual notation (Figure 3.2, 4.2, and 5.2), that can accommodate various versions (or implementations) of the ontology schema of the three patterns. The graphical notation and generic structure of the ODPs revisited caters to the needs of *Research Question 2* of Section 1.4.

Two versions of the Class As Property Value are identified (Chapter3): (a) the most generic version, in which the meaning of a class used as a property value is modified (or re-interpreted) (Figure 3.2); and (b) the simplified version, in which the meaning of a class used as a property value is preserved (Figure 3.8). The characterisation of this subtle, yet important variant of the CPV ODP, decouples two versions of the pattern that up to now, have not been explicitly considered. Various applicability scenarios involving modelling multiple cases of version (a) and (b) of the CPV ODP are examined, raising awareness regarding the implications of the applicability of the pattern in a given scenario. Specifically, it is shown that it is possible to use the CPV ODP, as in version (b), without having to modify (or re-interpret) the meaning of the classes being re-used as property values. These findings align to *Research Question 2* of Section 1.4.

Two versions of the Value Partition ODP are revisited: (a) a version in which the values of the feature space are implemented as individuals; and (b) a version in which the values of the feature space are implemented as classes (Chapter 4). The generic structure of the VP ODP is put forward in terms of the same graphical notation used by the CPV ODP generic structure, which allows one to perform a comparative analysis between the two patterns (Table 4.2, and 4.3). One of the outcomes of this analysis reveals that an instantiation of version (b) of the VP ODP is in fact, an instantiation *in disguise* of version (b) of the CPV ODP. That is, the generic structure of the VP ODP, in which the values of the feature space are implemented as classes, is actually a refinement (due to additional restrictions), of the generic structure of the simplified CPV ODP, in which the the meaning of a class used as a property value is preserved. Once again, these findings brings to the forefront aspects regarding the applicability and usage of these two distinct patterns that have not been previously considered, and align to *Research Question 2* of Section 1.4.

A similar rationale is followed to revisit the the Normalisation ODP (Chapter 5). A generic structure for the pattern in terms of the same graphical notation employed by the CPV and the VP ODPs, is presented, which is capable of accommodating multiple modules or semantic axes. The generic structure of the three patterns allows a comparative analysis among the three (Table 5.2). The structural comparison reveals that an instantiation of the Normalisation ODP, in which a module (or semantic axis) is represented using a class subsumption hierarchy, is in fact: (a) an instantiation of the simplified CPV ODP, in which the meaning of a class used as a property value is preserved; or (b) it may be an instantiation of the VP ODP, in which the values of the feature space are implemented as classes. Once again, the inter-dependencies and the existing structural and semantic alignments among these three patterns, has not been discussed before. In essence, three patterns that are aimed at three different modelling scenarios, are ultimately implemented by a similar set of OWL idioms. This raises the awareness regarding their intended usage in line with *Research Question 2* of Section 1.4.

In Library and Information Science, a simplified methodology of facet analysis to develop a Faceted Classification Scheme (FCS) is examined, containing the conceptualization of various classification criteria (facets) of a specific target domain concept (Chapter 6). A series of mappings between the elements of a generic FCS and the Normalization ODP have been identified that allow to convert a given FCS into an OWL DL ontology model following a consistent and systematic approach (Table 6.2). The resultant ontology model includes the representation of the various classification criteria of the domain concept considered in the original FCS. An existing FCS example in the domain of "Dishwashing Detergent" is used to illustrate the main steps of the conversion procedure.

This transformation guidelines of a non-ontological resource, a FCS, into an ontological one, an OWL DL ontology model, have been packaged into a re-engineering pattern referred to as Faceted Classification ODP (Figure 6.1). The Faceted Classification ODP does not cover yet, all existing types of generic structures of FCSs, and does not eliminate all opportunities for potentially hazardous ad-hoc decisions in the development process. However, it is a consistent, systematic and fit-for-purpose approach that allows to significantly reduced them. It provides evidence that Facet Analysis and Faceted Classification can indeed have an important role in Ontology Engineering when the modelling of multiple classification criteria of domain concepts is involved. Arriving to this conclusion was one of the key questions set out by *Research Question 1* of Section 1.4, for which the Faceted Classification ODP is put forward as a partial solution.

10.1.3 Pizza and Wine Will Never Be the Same

To put to the test the various aspects of the research presented, two well-known ontology model examples in the ontology development literature in the context of the W3C standard OWL language, are examined from a reverse engineering standpoint. They are the ontology models of "Pizza" and "Wine", and the topics studied include the following:

- The existing multiple classification criteria being implicitly conceptualised, rather than explicitly.
- The applicability of the generic structure of the three ODPs presented.
- The validity of the alignments identified among the various versions of the three ODPs.
- The composition of patterns that can take place in the instantiation of the Normalisation ODP with respect to the Class As Property Value and Value Partition ODPs.
- The bidirectionality of the transformation guidelines inherent in the Faceted Classification ODP.

The examination shows for the case of the "Pizza" example (Chapter 7), that the ontology schema aligns to an instantiation of the Normalisation ODP, where four modules (or semantic axes) of "Pizza" are considered: country of origin, type of base, type of toppings, and level of spiciness. Each one of these four semantic axes corresponds to a distinct classification criterion of "Pizza" and is a candidate to become a facet in a hypothetical application of the Faceted Classification Schema ODP as per the alignments characterized in Chapter 6.

The examination of the "Pizza" ontology model reveals furthermore that the instantiation of the Normalisation ODP with multiple modules, includes instantiations of the Value Partition and the Class As Property Value ODPs as per the alignments characterized in Chapter 5. Specifically, an instantiation of the VP in which the values of the feature space (spiciness) are represented as classes, and two instantiations of the simplified version of the CPV in which the meaning of the classes used as property values (those that represent a pizza base and a pizza topping respectively) is preserved.

Lastly, our examination shows another interesting aspect in the structural composition of the Normalisation ODP, which is the possibility of having *nested instantiations*. It can be observed that the main instantiation of the Normalisation ODP for the "Pizza" domain concept, includes another instantiation of the Normalisation pattern at a lower level of scope for the concept of "Pizza Topping".

A similar evaluation procedure was followed for the case of the "Wine" ontology model example (Chapter 8). The analysis shows that the "Wine" ontology also aligns to an instantiation of the Normalisation ODP. In this case, the pattern consists of six different modules (or semantic axes), namely: the type of grape, the region of origin, the color, the body, the flavor, and the level of sugar. Each one of these semantic axes can be seen as a classification criterion of "Wine", which means that it can constitute a facet in a hypothetical application of the Faceted Classification Schema ODP as per the alignments characterized in Chapter 6.

In terms of instantiations of the VP or the CPV patterns as part of the Normalisation ODP, there are four occurrences of the Value Partition ODP only, in which the values of the feature space (wine color, body, flavor and sugar level respectively) are represented as individuals. There are no instantiations of the CPV pattern as part of the Normalisation ODP, given that the conceptual elements used to represent the other two semantic axes (type of grape and region of origin) are individuals and not classes.

In both cases, "Pizza" and "Wine", the alignments identified with respect to the Normalisation ODP, bring to the forefront the implicit multiple classification criteria that are part of the conceptualisation of these two domain concepts. This information fits within the goals of *Research Question 2* of Section 1.4 by characterizing further the structural and semantic implications and applicability of the patterns considered in the context of modelling multiple classification criteria.

All alignments and instantiations identified of these three patterns, in conjunction with the transformation guidelines inherent in the Faceted Classification ODP, allows one to propose a hypothetical FCS for "Pizza" and "Wine" that in theory, would produce respectively the ontology models being considered initially (Figure 7.1, and 8.2). Nonetheless, the FCS reconstruction process in both cases, unveils aspects of the transformation process between a FCS and a ontology schema that require further research, suggesting the need to support more complex structures of FCSs, such as those including *subfacets* for example.

At the same time, the construction of a hypothetical traditional FCS (in the Library Information Science sense) from scratch for the "Pizza" and "Wine" domain concepts, using the outcome of the evaluation process in each case, reinforces one of the main ideas captured by *Research Question 1* of Section 1.4: that Facet Analysis and Faceted Classification can increase the consistency and systematic guidelines for the representation of multiple classification criteria of domain concepts in ontology models suitable for deployment in the Semantic Web.

10.1.4 So Whose "Fault" Was It in the End?

Lastly, and going back full circle, the most interesting evaluation of the work produced, consists on the creation from scratch of the ontology model to represent the multiple classification criteria intrinsic to the characterisation of the "Fault" domain concept in the context of the ReSIST Project.

In the case of "Fault", the Faceted Classification ODP from Chapter 6, is applied in its entirety throughout Chapter 9. This use case represents an ideal example of the systematic and consistent guidelines that this research aims to propose to mitigate the opportunity for ad-hoc practices, when building an ontology model from scratch of a domain concept that is naturally conceptualise in terms of multiple classification criteria.

Based on the classification criteria of the "Fault" concept found in Avizienis et al. (2004) and summarized in Section 1.1, a simple FCS is readily created that cover the whole scope of the relevant domain of discourse (Table 9.1). Applying the Faceted Classification ODP, the "Fault" FCS is practically automatically transformed into a corresponding OWL DL ontology model, by virtue of the identified alignments to the Normalisation ODP (Figure 9.1, 9.2, and 9.3). This systematic transformation from a non-ontological FCS of "Fault" into an OWL DL ontology model, is put forward as a partial answer to *Research Question 1* from Section 1.4. Moreover, it addresses directly the requirements set out in *Scenario (a)* of the "Fault" ontology for ReSIST described in Section 1.1.3.

Secondly, the Class As Property Value ODP is applied to the ontological representation of "Fault" in the context of the overall ontology model used by the RKB Explorer application of the ReSIST project at large. More specifically, it is the most generic version of the CPV ODP that is applied, in which the meaning of the classes used as property values is *re-interpreted*. In this case, the ontological representation of "Fault" provides a terminology to be used as the value for various properties with respect to other concepts in the overall ontology, such as publications, people's research interests, or the capabilities of resilience mechanisms in computer systems.

Using the CPV ODP in this fashion (Figure 9.4), implies that anonymous instances of "Fault", are re-interpreted as: (a) the topic of a publication when used as the value of a property intended for subjects of publications, (b) a research interest when used as the value of a property intended for people's research interests; or (c) a feature of resilience tolerance when used as the value of a property intended for resilience capabilities of computer systems. This application of the most generic version of the CPV ODP, addresses directly the needs captured in *Scenario (b)* of the "Fault" ontology for ReSIST as described in Section 1.1.3.

The development of the "Fault" ontology model throughout this evaluation, demonstrates how both version of the Class As Property Value ODP outlined in Chapter 3 coexist within the same model. On one hand, instances of the simplified version of the CPV ODP (Figure 3.2), in which the meaning of the classes used as property values are preserved, occur as part of applying the Faceted Classification ODP to the "Fault" concept. On the other hand, instances of the generic version of the CPV ODP (Figure 3.8), which re-interprets the semantic of the classes used as property values, are applied to re-use the representation of a real world "Fault" as a terminology or keyword index for other domain concepts in the ontology model. These clarifications in terms of applicability and interdependencies among the various patterns involved in the representation of multiple classification criteria, cover part of the objectives outlined in *Research Question 2* from Section 1.4.

10.1.5 Summary

As the title states, this work has presented a quest, at times fascinating, at times utterly frustrating and at times everything in between, "Towards Ontology Design Patterns to Model Multiple Classification Criteria of Domain Concepts in the Semantic Web". With the idea of bringing together all the material covered, the conclusions itemized in the previous sections of this chapter, could be expressed in terms of the research questions advanced at the beginning of this work.

In that sense, *Research Question 1* from Section 1.4, raised the following concerns:

- Are there consistent and systematic techniques and guidelines to represent multiple classification criteria (or to some extent multiple inheritance) of domain concepts in ontology models suitable for deployment in the context of the Semantic Web?

Our survey of the Ontology Engineering landscape, indicates that such guidelines are not readily available. The representation of multiple classification criteria does not seem to have a very prominent role in the Ontology Engineering literature. It is not a case scenario being proactively and regularly considered and analyzed as part of the ontology design and modelling process. Thus, there is not a sense of awareness or consensus in the Ontology Engineering community, in terms of best practices on how to approach it.

- What could be learnt from fields such as Object-Oriented Design and Faceted Classification, which have already been exposed to the design of multiple classification criteria conceptual models for much longer than Ontology Engineering?

Our survey also noticed that the design scenario of multiple classification criteria is in fact, known within the Object-Oriented Design and Library Information Science fields. Object-Oriented Design can count on specific patterns to address: (a) the case of *nested generalizations*, which is one of the modeling problems that multiple classification criteria can lead to; and (b) the case of *view inheritance*, which is essentially a manifestation of multiple alternative classification criteria. In LIS, there is a specific classification system whose main purpose is to actually support the classification of resources according to multiple viewpoints. That is, Faceted Classification and Facet Analysis as part of it. The transformation guidelines to convert a given FCS into an ontology model that resulted into the creation of the FCS ODP, come to corroborate that indeed, Facet Analysis and Faceted Classification can play an important role "Towards Ontology Design Patterns to Model Multiple Classification Criteria of Domain Concepts in the Semantic Web". This claim is further illustrated with the creation (or decomposition) of ontology models of "Dishwashing Detergent", "Pizza", "Wine", and "Fault".

At the same time, *Research Question 2* from Section 1.4, referred to these other questions:

- Are there ODPs that could be applied to represent multiple classification criteria of domain concepts?

No single ODP was found to be explicitly aimed at the modelling of classification criteria as part of our survey, which included the two public ODP catalogues known to date. Although one pattern emerged as a possible starting point, that is, the Normalisation ODP. A closer examination of the Normalisation ODP revealed that an instantiation of the pattern required the instantiation of at least one flavour of a different pattern, the Class As Property Value, and could possibly include an instantiation of the Value Partition ODP. This relationship among the three patterns prompted us to review in detail the structural and semantic definition of the three. In the end, the alignments identified between the generic structure of a FCS in LIS and the Normalisation ODP, served the basis to introduce the FCS ODP, specifically designed to represent classification criteria of domain concepts in Semantic Web ontology models.

- If so, are they fully detailed or is there opportunity for ambiguity?

The three patterns identified, Normalisation, CPV, and VP, are properly detailed for their original purpose. It is when their definition is stretched in an attempt to accommodate the representation of classification criteria that certain aspects have not been explicitly documented, opening an opportunity for ambiguity. For example, it is important to know that the CPV ODP can be applied so that the original meaning of the classes acting as property values can be preserved (what has been referred to as the simplified version of the CPV) and do not need to be re-interpreted (as it is the case in Approach 4 of Noy (2005) and what hereto has been referred to as the most generic version of the CPV). In fact, this distinction, together with the comparative analysis carried out among the Normalisation, VP, and CPV ODPs, revealed the implications and interdependencies that an instantiation of one of these patterns could have in terms of instances of the others.

- In the case of having several ODPs, how do they relate to each other and what could be learnt from this?

As detailed already in previous sections of this chapter, yes, there are interdependencies and implications between the use of the Normalisation ODP, and the VP, and CPV ODPs. An interesting corollary that emerges from the outcome of the structural and semantic comparative analysis of the aforementioned patterns is that, in essence, three patterns that are by definition aimed at three different modelling scenarios, can ultimately be implemented in some cases, by a similar set of OWL idioms.

In closing, this thesis was set out to seek a significant advancement in the current Ontology Engineering landscape, when representing multiple classification criteria of domain concepts. To that end, the work presented throughout this thesis, provides evidence to claim that indeed, Facet Analysis and Faceted Classification principles should receive a prominent role as part of the Ontology Engineering tool-set. This statement does not imply that the artifacts of Facet Analysis and Faceted Classification alone, suffice to account for all conceptual elements that should be considered when creating an ontology model that includes various classification criteria of a domain concept. This research aims to convey however, the reverse implication. That is, that an ontology model that requires representing classification criteria, should consider all conceptual elements derived from the artifacts that result from the process of Facet Analysis and Faceted Classification.

Even though, these contributions do not solve all challenges of the modeling problem considered, they make explicit many modeling decisions previously taken implicitly in the ontology development field providing a valuable resource available to ontology engineers when dealing with this particular and very recurrent modeling scenario. There are still several opportunities for improvement and open concerns to pursue, which are outlined in the section that follows.

10.2 Future Work

The following sections cover opportunities for improvement of the work performed throughout this research, and topics open for further investigation.

10.2.1 Automation of the Faceted Classification ODP

Perhaps the most appealing opportunity in the short-term, is the automation of the required steps to create the normalised ontology model. Provided a source FCS and the mappings identified here, an application could automatically or semi-automatically generate the corresponding normalized ontology artifact.

Depending on the complexity of the source FCS, user intervention might be required to disambiguate among several valid design choices available and assist the application to select the preferred option.

To materialize this application, the development of a plug-in or extension for some of the most popular open ontology development frameworks is being evaluated.

10.2.2 Multiple FCSs

There are additional design scenarios that present attractive incremental challenges. Consider the situation where two (or more) different domain-specific FCSs are to be transformed into a single normalized ontology model. For example, a FCS for "Dishwasher Detergent" and a different FCS for "Tooth Cleaning Products". A situation with two FCSs can lead to the following design scenarios:

- Case 1: FCS1 and FCS2 do not have any element in common (facet or facet term).
- Case 2: FCS1 and FCS2 do have some element in common (facet or facet term).
- Case 3: The domain of discourse (TDC) of FCS1 appears as a facet or as a facet term in FCS2 (or vice versa).

Case 1 would be the simplest. The Faceted Classification ODP can be applied separately to FCS1 and FCS2 and the outcome combined into a single ontology model. The only difference between Case 1 and having only one FCS, is that the ontology model obtained will include two $:TDC$ classes, (provided by FCS1 and FCS2 respectively) and the rest of the ontology elements ($:Facet_i$, $:hasFacet_i$, $:F_iTerm_j$, etc.) will be populated with the elements of both FCS1 and FCS2.

Case 2 and 3 on the other hand, could potentially lead to a myriad of different modeling issues that have not been yet explored. The idea going forward, is to extend the transformation guidelines to support scenarios such as Case 2 and 3.

Consider another situation where two (or more) different FCSs of the *same* domain concept are to be transformed into a single normalized ontology. For example, two FCSs for "Dishwasher Detergent" developed separately.

10.2.3 Complex FCSs

This initial version of the Faceted Classification ODP has considered for now a simple generic structure of the source FCS to transform into an ontology model. The structure outlined in Definition 6.1, which is grounded on the FCS example by Denton (2003).

It is the aim in the future, to analyse the support of more complex structures of FCSs, starting with a thorough examination of FCSs that may include subfacets.

A common aspect of the visualisation of faceted classification systems is the representation of value intervals, such as prices ranges, age groups, time period intervals, etc. The representation of these type of value intervals or ranges have not been explored so far in the current version of the Faceted Classification ODP.

10.2.4 Bidirectionality of the Faceted Classification ODP

As the examples of "Pizza" and "Wine" have illustrated, there can be ontology models aligned to the Normalisation ODP, that when attempting to be transformed back to the hypothetical FCS associated to them, do not produce a valid FCS.

This is typically the case, when despite the identified alignments between FCS and Normalisation ODP, it is not clear how some of the elements in the ontology model should be placed into the FCS.

The bidirectionality of the Faceted Classification ODP requires further investigation to account for such situations.

10.2.5 OWL 2 Punning

The release of version 2 of the W3C standard OWL language edited by Krötzsch et al. (2009), provides the feature of metamodelling or punning.

OWL 2 punning allows to use of the same identifier as an owl:Class and as an owl:NamedIndividual, while still preserving the expressivity level of the ontology model within OWL 2 DL.

With the punning feature, the example of Approach 1 of the Class As Property Value ODP by Noy (2005), where classes as used directly as property values, (i.e. the value of the dc:subject property is the actual class :Lion itself), would be still a valid OWL 2 DL ontology model.

On that basis, it would be relevant to revisit the Class As Property Value ODP, the version of the Value Partition ODP, in which the feature space is implemented as an owl:Class; and the Normalisation ODP using the punning feature. Punning would eliminate the need of using anonymous individuals in the three patterns as property values, making it possible to use directly the corresponding classes of those individuals instead.

A thorough analysis of the impact that a punning version of the three patterns would have in the work presented here, remains as one of the main and more interesting tasks that requires further investigation in the near future.

10.2.6 Ontology-based Product Data Management

Ontology-based Product Data Management[1] (OPDM) is a collaborative research project funded by the EUREKA's Eurostars initiative within the European Union Seventh Framework Program. OPDM aims to address one of the most important challenges in e-commerce, namely the efficient management of product data, data processes workflows and the supply of complete and structured product information. The application framework of the project requires the creation of +70 seed ontologies to represent common products in a broad range of categories, that are typically purchased on-line via web-based e-commerce platforms.

Each product ontology of OPDM, is created extending the GoodRelations[2] ontology developed by Hepp (2008) as a baseline. Goodrelations is a well-known ontology that adheres to the OWL DL profile, tailored for the domain of e-commerce and that supports modelling of the key elements involved in practically any scenario of buy/sell, offer/demand of product or services over the Web. Documentation available on the ontology portal[2] or Chapter 13 of Allemang and Hendler (2011), provide relevant examples that illustrate some of the features of GoodRelations.

The extendability of GoodRelations consists of introducing additional conceptual elements to represent the relevant characteristics of each product in question. The development of these product ontologies is currently in progress, although there are some initial prototypes already available, for domain concepts such as: "Bicycle"[3], "DVD/Blu-

[1] http://opdm-project.org/
[2] http://purl.org/goodrelations/
[3] http://purl.org/opdm/bicycle#

ray Player"[4], "Book"[5], "Coffee Machine"[6], "Dishwasher"[7], "Garment"[8], "Microwave"[9], "Perfume"[10], "Printer"[11], "Fridge/Freezer"[12], "Shoe"[13], etc.[14], (final ontology URIs subject to change - last accessed on April 5th, 2012).

Parts of the Faceted Classification ODP introduced in Chapter 6, are being applied to the creation of these +70 seed ontologies within the OPDM project application framework. More specifically, the principles of Facet Analysis during the conceptualization phase and the generic structure of the pattern captured in Figure 6.1.

As more product-specific ontologies are developed, certain alignments between the GoodRelations ontology and the Faceted Classification ODP seem to emerge. Most notably, classes that extend the class gr:QualitativeValue[15] or gr:QuantitativeValue[16] seem to align to the notion of facet and thus, to the :$Facet_i$ element of the FCS ODP generic structure. For example, the class obcc:BicycleType from the "Bicycle" product ontology, the class obk:BookFormat from the "Book" ontology, or classes from the "Shoe" product ontology such as osho:Style, osho:ClosingType, or osho:SizeScale, are all subclasses of gr:QualitativeValue and seem to align to the notion of facet as a principle of division for their domain of discourse (their target domain concept :TDC element).

The future road map of this research aims to formalize and characterize all of these emergent alignments between the Faceted Classification ODP presented in this thesis and parts of the GoodRelations ontology.

[4] http://purl.org/opdm/blurayplayer#
[5] http://purl.org/opdm/book#
[6] http://purl.org/opdm/coffeemachine#
[7] http://purl.org/opdm/dishwasher#
[8] http://purl.org/opdm/garment#
[9] http://purl.org/opdm/microwave#
[10] http://purl.org/opdm/perfume#
[11] http://purl.org/opdm/printer#
[12] http://purl.org/opdm/refrigerator#
[13] http://purl.org/opdm/shoe#
[14] http://purl.org/opdm/
[15] http://www.heppnetz.de/ontologies/goodrelations/v1#QualitativeValue
[16] http://www.heppnetz.de/ontologies/goodrelations/v1#QuantitativeValue

Appendix A

Copyright Clearance for Third-party Figures

Clearance for Figures 1.1, 1.2 and 1.3

Caption of Figure 1.1

"The elementary fault classes." **(Fig. 4 in Avizienis et al. (2004) p. 15)**

Caption of Figure 1.2

"The classes of combined faults (a) Matrix representation." **(Fig. 5(a) in Avizienis et al. (2004) p. 16)**

Caption of Figure 1.3

"The classes of combined faults (b) Tree representation." **(Fig. 5(b) in Avizienis et al. (2004) p. 16)**

Subject: Re: Kind request for copyright clearance to reproduce figure in PhD thesis
From: Brian Randell <Brian.Randell@ncl.ac.uk>
Date: 1/24/12 5:46 PM
To: Benedicto Rodriguez Castro <benedicto.rodriguez@unibw.de>
CC: Brian Randell <Brian.Randell@ncl.ac.uk>, "Carl E. Landwehr" <landwehr@isr.umd.edu>, Algirdas Avizienis <aviz@cs.ucla.edu>, Algirdas Avizienis <aviz@adm.vdu.lt>

Dear Dr Rodriguez-Castro:

I am pleased to give you my permission to include the listed figures from our paper in your thesis.

Kind regards

Brian Randell

On 24 Jan 2012, at 16:08, Benedicto Rodriguez Castro wrote:

> Dear Authors,
>
> My name is Benedicto Rodriguez-Castro. Recently, I completed a PhD degree in Semantic Web Technologies at the University of Southampton.
>
> As part of my thesis (see [1] below), I included three figures from your work (see [2] below) with full citation and attribution, to provide a concise depiction of a taxonomy of faults.
>
> However, in addition to full citation and attribution, the thesis submission guidelines of the University of Southampton require to provide copyright clearance of all reproduced materials from external sources in order to feature in the final hard-bound copy of the archived thesis.
>
> The figures in question from [2] are:
> - Page 15, Fig. 4. The elementary fault classes.
> - Page 16, Fig. 5(a). The classes of combined faults (a) Matrix representation.
> - Page 16, Fig. 5(b). The classes of combined faults (b) Tree representation.
>
> And they are reproduced in [1] respectively as:
> - Page 4, Figure 1.1: "The elementary fault classes." (Fig. 4 in Avižienis et al. (2004) p. 15)
> - Page 6, Figure 1.2: "The classes of combined faults (a) Matrix representation." (Fig. 5(a) in Avižienis et al. (2004) p. 16)
> - Page 7, Figure 1.3: "The classes of combined faults (b) Tree representation." (Fig. 5(b) in Avižienis et al. (2004) p. 16)
>
> In that sense, I was wondering if you could authorize copyright clearance for the figures referred to.
> Your help is greatly appreciated.
>
> Thanks a lot for your time and please, let me know any other information that you may need.
>
> Best regards,

> Bene Rodriguez-Castro
>
> --
> Research Associate
> E-Business and Web Science Research Group
> Department of General Management and E-Business
> Bundeswehr University of Munich (Germany)
> Phone: +49 89 6004-4850
> Email: benedicto.rodriguez@unibw.de
> Web: http://purl.org/beroca
>
> --
> [1] http://eprints.ecs.soton.ac.uk/22175/
> Rodriguez-Castro, B. (2011)
> Towards Ontology Design Patterns to Model Multiple Classification
> Criteria of Domain Concepts in the Semantic Web.
> PhD thesis, University of Southampton.
>
> [2] Algirdas Avižienis, Jean-Claude Laprie, Brian Randell, and Carl
> Landwehr.
> Basic concepts and taxonomy of dependable and secure computing.
> IEEE Transactions on Dependable and Secure Computing, 01(1):11{33, 2004.
> ISSN 1545-5971.

--
School of Computing Science, Newcastle University, Newcastle upon Tyne,
NE1 7RU, UK
EMAIL = Brian.Randell@ncl.ac.uk   PHONE = +44 191 222 7923
FAX = +44 191 222 8232 URL = http://www.cs.ncl.ac.uk/people/brian.randell

Clearance for Figure 1.4

Caption

"Usage of the ontology representation languages (a) and of the three OWL species (b)." **(Fig. 1 of d'Aquin et al. (2007) p. 3)**

RE: Kind Request for Copyright Clearance of Reproduced Figure

Benedicto Rodriguez
Sent:04 November 2011 13:28
To: Enrico Motta [e.motta@open.ac.uk]

Dear Enrico,

Apologies for a very late reply.
In this case, your email reply will suffice.
There is proper attribution and citation on the "caption" of the figure as well as on the "narrative" that refers to the figure.

Thanks a lot again for your help,
Bene Rodriguez

From: Enrico Motta [e.motta@open.ac.uk]
Sent: 28 October 2011 22:50
To: Benedicto Rodriguez
Cc: m.daquin@open.ac.uk; c.baldassarre@open.ac.uk; s.angeletou@open.ac.uk; r.m.sabou@open.ac.uk; e.motta@open.ac.uk
Subject: Re: Kind Request for Copyright Clearance of Reproduced Figure

Dear Benedicto

sure, no problem if you wish to reuse a figure from our paper, as long as there is a proper attribution.

Please let me know whether this is OK or you need something more formal

Cheers

Enrico

At 14:16 +0000 24/10/11, Benedicto Rodriguez wrote:
>Dear Authors,
>
>My name is Benedicto Rodriguez-Castro. Recently, I completed a PhD
>degree in Semantic Web Technologies at the University of Southampton.
>
>As part of my thesis [1], I included Figure 1 from your publication
>[2] with full citation and attribution, to provide an overview of
>the usage of ontology representation languages in the Web of Data.
>(Please, see the aforementioned inclusion on page 10 of [1]).
>
>However, in addition to full citation and attribution, the thesis
>submission guidelines of the University of Southampton require to
>provide copyright clearance of all reproduced materials from
>external sources in order to feature in the final archived thesis
>hard-bound copy.
>
>In that sense, I was wondering if you could authorize copyright
>clearance for the figure referred to. Your contribution is greatly
>appreciated.
>
>Thanks a lot for your time and please, let me know any other
>information that you may need.
>
>Best regards,

>Bene Rodriguez-Castro
>
>[1] http://eprints.ecs.soton.ac.uk/22175/
>Rodriguez-Castro, B. (2011) Towards Ontology Design Patterns to
>Model Multiple Classification Criteria of Domain Concepts in the
>Semantic Web. PhD thesis, University of Southampton.
>
>[2] http://ceur-ws.org/Vol-329/paper01.pdf
>Mathieu d'Aquin, Claudio Baldassarre, Laurian Gridinoc, So?a
>Angeletou, Marta Sabou, and Enrico Motta. Characterizing knowledge
>on the semantic web with watson. In Raul Garcia-Castro, Denny
>Vrandecic, Asuncion Gomez-Perez, York Sure, and Zhisheng Huang,
>editors, EON, volume 329 of CEUR Workshop Proceedings, pages 1-10.
>CEUR-WS.org, 2007.
>
>--
>The Open University is incorporated by Royal Charter (RC 000391), an
>exempt charity in England & Wales and a charity registered in
>Scotland (SC 038302).

--

The Open University is incorporated by Royal Charter (RC 000391), an
exempt charity in England & Wales and a charity registered in
Scotland (SC 038302).

Clearance for Figure 1.5

Caption

"NeOn Glossary of Processes and Activities". **(Figure 4 in Suarez-Figueroa et al. (2008) p. 15)**

Subject: Re: Kind Request for Copyright Clearance of Reproduced Figure
From: Asunción Gómez Pérez <asun@fi.upm.es>
Date: 1/4/12 12:48 AM
To: Benedicto Rodriguez Castro <benedicto.rodriguez@unibw.de>
CC: mcsuarez@fi.upm.es, klaasd@uni-koblenz.de, mfernandez.eps@ceu.es, holger.lewen@kit.edu, mdzbor@kmi.org

Sure, no problem.
Asun

El 03/01/2012 15:28, Benedicto Rodriguez Castro escribió:

> Dear Mari Carmen and all,
>
> My name is Benedicto Rodriguez-Castro. Recently, I completed a PhD degree in Semantic Web Technologies at the University of Southampton.
>
> As part of my thesis [1], I included Figure 4 from your publication [2] with full citation and attribution, to provide a high level picture of all the processes and activities involved in Ontology Engineering. (Please, see the aforementioned inclusion on page 12 of [1]).
>
> However, in addition to full citation and attribution, the thesis submission guidelines of the University of Southampton require to provide copyright clearance of all reproduced materials from external sources in order to feature in the final hard-bound copy of the archived thesis.
>
> In that sense, I was wondering if you could authorize copyright clearance for the figure referred to. Your contribution is greatly appreciated.
>
> Thanks a lot for your time and please, let me know any other information that you may need.
>
> Best regards,
> Bene Rodriguez-Castro

--
Prof. Asunción Gómez-Pérez
Director of the Ontology Engineering Group
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo, sn
Boadilla del Monte, 28660, Spain
Home page: www.oeg-upm.net
Email: asun@fi.upm.es
Phone: (34-91) 336-7417
Fax: (34-91) 352-4819

Clearance for Figure 2.2

Caption

"Ontology Design Patterns types". **(Figure 2.2 in Presutti et al. (2008) p. 19).**

Subject: Re: Copyright Clearance to Reproduce Figure in PhD Thesis
From: valentina presutti <valentina.presutti@istc.cnr.it>
Date: 1/10/12 7:40 PM
To: Benedicto Rodriguez Castro <benedicto.rodriguez@unibw.de>
CC: valentina presutti <valentina.presutti@istc.cnr.it>, aldo.gangemi@istc.cnr.it

Sure, no problem.

Valentina

On Jan 3, 2012, at 3:56 PM, Benedicto Rodriguez Castro wrote:

> Dear Valentina and Aldo,
>
> My name is Benedicto Rodriguez-Castro. Recently, I completed a PhD degree in Semantic Web Technologies at the University of Southampton.
>
> As part of my thesis [1], I included Figure 2.2 from your publication [2] with full citation and attribution, to provide a high-level picture of an introductory classification of Ontology Design Patterns types. (Please, see the aforementioned inclusion on page 22 of [1]).
>
> However, in addition to full citation and attribution, the thesis submission guidelines of the University of Southampton require to provide copyright clearance of all reproduced materials from external sources in order to feature in the final hard-bound copy of the archived thesis.
>
> In that sense, I was wondering if you could authorize copyright clearance for the figure referred to. Your contribution is greatly appreciated.
>
> Thanks a lot for your time and please, let me know any other information that you may need.
>
> Best regards,
> Bene Rodriguez-Castro
>
> --
> Research Associate
> E-Business and Web Science Research Group
> Department of General Management and E-Business
> Bundeswehr University of Munich (Germany)
> Phone: +49 89 6004-4850
> Email: benedicto.rodriguez@unibw.de
> Web: http://purl.org/beroca
>
> --
> [1] http://eprints.ecs.soton.ac.uk/22175/
> Rodriguez-Castro, B. (2011)
> Towards Ontology Design Patterns to Model Multiple Classification Criteria of Domain Concepts in the Semantic Web. PhD thesis, University of Southampton.
>
> [2] http://www.neon-project.org/webcontent/images/Publications/neon 2008 d2.5.1.pdf
> Valentina Presutti, Aldo Gangemi, Stefano David, Guadalupe Aguado de Cea, Mari Carmen Suarez-Figueroa, Elena Montiel-Ponsoda, and Maria Poveda.
> A library of ontology design patterns: reusable solutions for collaborative design of networked ontologies.

> NeOn deliverable D2.5.1, Institute of Cognitive Sciences and Technologies (CNR), 2008.

Clearance for Figure 2.3

Caption

"Comparison of proposed ODPs (left) and previous work (right). Three criteria are used for comparison: application and target spotting methodologies, documentation and types of formalisms." (Figure 11 of Egana-Aranguren et al. (2008) p. 11).

Subject: Re: Fwd: Copyright Clearance to Reproduce Figure in PhD Thesis
From: Benedicto Rodriguez Castro <benedicto.rodriguez@unibw.de>
Date: 1/4/12 4:14 PM
To: Mikel Egaña Aranguren <megana@fi.upm.es>

Hi Mikel,

Thanks a lot for your authorization and prompt response.

Your email should be enough.

Best regards,
Bene Rodriguez

On 1/4/12 12:14 PM, Mikel Egaña Aranguren wrote:

> Hi;
>
> I authorize copyright clearance.
>
> Is this email enough or do I have to fill a form or something?
>
> Regards
>
> On ar., 2012.eko urtren 03a 17:44, Benedicto Rodriguez Castro wrote:
>
>> -------- Original Message --------
>> Subject: Copyright Clearance to Reproduce Figure in PhD Thesis
>> Date: Tue, 03 Jan 2012 17:39:59 +0100
>> From: Benedicto Rodriguez Castro <benedicto.rodriguez@unibw.de>
>> To: mikel.eganaaranguren@cs.man.ac.uk
>> CC: erant@psb.ugent.be, martin.kuiper@psb.ugent.be, robert.stevens@manchester.ac.uk
>>
>> Dear Mikel and all,
>>
>> My name is Benedicto Rodriguez-Castro. Recently, I completed a PhD degree in Semantic Web Technologies at the University of Southampton.
>>
>> As part of my thesis [1], I included Figure 11 from your publication [2] with full citation and attribution, to provide a high-level picture of an introductory classification of Ontology Design Patterns types. (Please, see the aforementioned inclusion on page 23 of [1]).
>>
>> However, in addition to full citation and attribution, the thesis submission guidelines of the University of Southampton require to provide copyright clearance of all reproduced materials from external sources in order to feature in the final hard-bound copy of the archived thesis.
>>
>> In that sense, I was wondering if you could authorize copyright clearance for the figure referred to. Your contribution is greatly appreciated.
>>
>> Thanks a lot for your time and please, let me know any other information

> > that you may need.
> >
> > Best regards,
> > Bene Rodriguez-Castro

--
Research Associate
E-Business and Web Science Research Group
Department of General Management and E-Business
Bundeswehr University of Munich (Germany)
Phone: +49 89 6004-4850
Email: benedicto.rodriguez@unibw.de
Web: http://purl.org/beroca

Clearance for Figures 2.5 and 2.6

Caption of Figure 2.5

"Classification of the valid categories of inheritance" (Figure 1 in Meyer (1996) and Figure in Meyer (2000)(§ 24.5, p. 824).

Caption of Figure 2.6

"Classification through views" (Figure in Meyer (2000)(§ 24.10, p. 853).

Subject: Correction -- RE: Copyright Clearance to Reproduce Figure in PhD Thesis
From: Bertrand Meyer <Bertrand.Meyer@inf.ethz.ch>
Date: 1/3/12 8:29 PM
To: 'Bene Rodriguez' <benedicto.rodriguez@ebusiness-unibw.org>

Sorry for misspelling your name! It's corrected below.

-- BM

-----Original Message-----
From: Bertrand Meyer [mailto:Bertrand.Meyer@inf.ethz.ch]
Sent: 03 January 2012 20:28
To: 'Bene Rodriguez'
Subject: RE: Copyright Clearance to Reproduce Figure in PhD Thesis

Dear Benedicto:

Please consider this email as my permission for you to use the figures mentioned in your message, under the conditions that you describe, in your thesis.

Congratulations on completing this work and best wishes,

-- Bertrand Meyer

-----Original Message-----
From: Bene Rodriguez [mailto:benedicto.rodriguez@ebusiness-unibw.org]
Sent: 03 January 2012 18:15
To: bertrand.meyer@inf.ethz.ch
Subject: Copyright Clearance to Reproduce Figure in PhD Thesis

Dear Bertrand,

My name is Benedicto Rodriguez-Castro. Recently, I completed a PhD degree in Semantic Web Technologies at the University of Southampton.

As part of my thesis [1], I included two figures with full citation and attribution, to provide a concise depiction respectively of: (a) the different types of valid inheritance in object-oriented design; and (b) an example of "View Inheritance".

However, in addition to full citation and attribution, the thesis submission guidelines of the University of Southampton require to provide copyright clearance of all reproduced materials from external sources in order to feature in the final hard-bound copy of the archived thesis.

The figures in questions are:
- "Classi
cation of the valid categories of inheritance", figure 1 in [2] and the figure in section 24.5, page 824 of [3].
- "Classi
cation through views", the figure in section 24.10, page 853 of [3].

Please, see the aforementioned reproduction of the figures on pages 29 and 30 of [1] respectively.

In that sense, I was wondering if you could authorize copyright clearance for the figure referred to. Your contribution is greatly appreciated.

Thanks a lot for your time and please, let me know any other information that you may need.

Best regards,
Bene Rodriguez-Castro

--
Research Associate
E-Business and Web Science Research Group Department of General Management and E-Business Bundeswehr University of Munich (Germany)
Phone: +49 89 6004-4850
Email: benedicto.rodriguez@unibw.de
Web: http://purl.org/beroca

--
[1] http://eprints.ecs.soton.ac.uk/22175/
Rodriguez-Castro, B. (2011)
Towards Ontology Design Patterns to Model Multiple Classification Criteria of Domain Concepts in the Semantic Web. PhD thesis, University of Southampton.

[2] Bertrand Meyer.
The many faces of inheritance: A taxonomy of taxonomy.
Computer, 29:105{108, 1996. ISSN 0018-9162.

[3] Bertrand Meyer.
Object-Oriented Software Construction (Book/CD-ROM) (2nd Edition).
Prentice Hall PTR, March 2000. ISBN 0136291554.

Clearance for Figure 3.1

Caption

"Using members of a class as values for properties" **(Fig. 4 in Noy (2005)).**

Subject: Re: Copyright Clearance to Reproduce Figure in PhD Thesis
From: Natasha Noy <noy@stanford.edu>
Date: 1/4/12 9:27 PM
To: Benedicto Rodriguez Castro <benedicto.rodriguez@unibw.de>

Dear Bene,

The copyright is help by W3C and not the editors. The following is the text of the W3C copyright on the document:
http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231

You get to it from the Copyright link. Per this license, you are free to include the figure as long as you follow the conditions that are specified at the link above. i would suggest that you forward the link to your university and go from there.

Best regards,

Natasha

On Jan 3, 2012, at 4:43 PM, Benedicto Rodriguez Castro wrote:

> Dear Natasha,
>
> My name is Benedicto Rodriguez-Castro. Recently, I completed a PhD degree in Ontology Design Patterns at the University of Southampton (UK).
>
> As part of my thesis [1], I included Figure 4 "Using members of a class as values for properties" from [2] with full citation and attribution, to provide a visual depiction of that specific approach of the design pattern. (Please, see the aforementioned inclusion on section 3.2, page 45 of [1]).
>
> However, in addition to full citation and attribution, the submission guidelines of the university require to provide copyright clearance of all reproduced materials from external sources in order to feature in the final hard-bound copy of the archived thesis.
>
> In that sense, I was wondering if you could authorize copyright clearance for the figure referred to. Your contribution is greatly appreciated.
>
> Thanks a lot for your time and please, let me know any other information that you may need.
>
> Best regards,
> Bene Rodriguez-Castro
>
> --
> Research Associate
> E-Business and Web Science Research Group
> Department of General Management and E-Business
> Bundeswehr University of Munich (Germany)
> Phone: +49 89 6004-4850
> Email: benedicto.rodriguez@unibw.de
> Web: http://purl.org/beroca

--
[1] Rodriguez-Castro, B. (2011)
Towards Ontology Design Patterns to Model Multiple Classification Criteria of Domain Concepts in the Semantic Web.
PhD thesis, University of Southampton.
URL to thesis record: http://eprints.ecs.soton.ac.uk/22175/
URL to thesis pdf file: http://eprints.ecs.soton.ac.uk/22175/5/thesis2010-rev685.pdf

[2] Natasha F. Noy. (2005)
Representing Classes As Property Values on the Semantic Web.
Technical Report Note 5, W3C, Semantic Web Best Practices and Deployment Working Group.
http://www.w3.org/TR/swbp-classes-as-values/

W3C Document License

Public documents on the W3C site are provided by the copyright holders under the following license.

License

By using and/or copying this document, or the W3C document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to copy, and distribute the contents of this document, or the W3C document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on *ALL* copies of the document, or portions thereof, that you use:

A link or URL to the original W3C document.
The pre-existing copyright notice of the original author, or if it doesn't exist, a notice (hypertext is preferred, but a textual representation is permitted) of the form: "Copyright © [$date-of-document] World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231"
If it exists, the STATUS of the W3C document.

When space permits, inclusion of the full text of this **NOTICE** should be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of W3C documents is granted pursuant to this license. However, if additional requirements (documented in the Copyright FAQ) are satisfied, the right to create modifications or derivatives is sometimes granted by the W3C to individuals complying with those requirements.

Disclaimers

THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.

Notes

This version: http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231

This formulation of W3C's notice and license became active on December 31 2002. This version removes the copyright ownership notice such that this license can be used with materials other than those owned by the W3C, moves information on style sheets, DTDs, and schemas to the Copyright FAQ, reflects that ERCIM is now a host of the W3C, includes references to this specific dated version of the license, and removes the ambiguous grant of "use". See the older formulation for the policy prior to this date. Please see our Copyright FAQ for common questions about using materials from our site, such as the translating or annotating specifications.

Bibliography

Harith Alani, Peter Chandler, Wendy Hall, Kieron O'Hara, Nigel Shadbolt, and Martin Szomszor. Building a pragmatic semantic web. *IEEE Intelligent Systems*, 23(3):61–68, 2008. ISSN 1541-1672.

C. Alexander, S. Ishikawa, and M. Silverstein. *A pattern language: towns, buildings, construction*. Center for Environmental Structure series. Oxford University Press, 1977. ISBN 9780195019193.

D. Allemang and J. Hendler. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Morgan Kaufmann. Elsevier Science, 2011. ISBN 9780123859655. Second Edition.

Dean Allemang and James Hendler. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008. ISBN 0123735564, 9780123735560.

Mikael Amborn. Facet-Oriented Program Design. Master's thesis, Department of Computer and Information Science, Linköping University, May 2004. ISRN: LITH-IDA-EX04/047SE.

Tom Anderson, Zoe Andrews, John Fitzgerald, Brian Randell, Hugh Glaser, and Ian Millard. The ReSIST Resilience Knowledge Base. In *DSN 2007 - The 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2007.

Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 01(1):11–33, 2004. ISSN 1545-5971.

Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003. ISBN 0-521-78176-0.

Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001.

G. Bhattacharyya. Popsi: its fundamentals and procedure based on a general theory of subject indexing languages. *Library Science with a Slant to Documentation*, 16(1): 1–34, 1979.

Eva Blomqvist. Ontocase - a pattern-based ontology construction approach. In Robert Meersman and Zahir Tari, editors, *OTM Conferences (1)*, volume 4803 of *Lecture Notes in Computer Science*, pages 971–988. Springer, 2007. ISBN 978-3-540-76846-3.

Eva Blomqvist. Pattern ranking for semi-automatic ontology construction. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 2248–2255, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-753-7.

Eva Blomqvist. Ontocase-automatic ontology enrichment based on ontology design patterns. In Abraham Bernstein, David Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, editors, *The Semantic Web - ISWC 2009*, volume 5823 of *Lecture Notes in Computer Science*, pages 65–80. Springer Berlin / Heidelberg, 2009. 10.1007/978-3-642-04930-9_5.

Eva Blomqvist, Aldo Gangemi, and Valentina Presutti. Experiments on pattern-based ontology design. In *Proceedings of the fifth international conference on Knowledge capture*, K-CAP '09, pages 41–48, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-658-8.

Eva Blomqvist, Valentina Presutti, Enrico Daga, and Aldo Gangemi. Experimenting with extreme design. In Philipp Cimiano and H. Pinto, editors, *Knowledge Engineering and Management by the Masses*, volume 6317 of *Lecture Notes in Computer Science*, pages 120–134. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-16438-5_9.

Vanda Broughton. *Essential Classification*. Facet, London, 2004.

Vanda Broughton. The need for a faceted classification as the basis of all methods of information retrieval. In David Nicholas, editor, *New Information Perspectives*, volume 58 of *Aslib Proceedings*, pages 49–72. Emerald Group Publishing Limited, 2006. 10.1108/00012530610648671.

Vanda Broughton. Brian Vickery and the Classification Research Group: the legacy of faceted classification. In A. Gilchrist, editor, *Proceedings of the Second National ISKO UK Conference*. 2011.

Thomas A. Cargill. Controversy: The case against multiple inheritance in c++. *Computing Systems*, 4(1):69–82, 1991.

Oscar Corcho, Mariano Fernandez-Lopez, and Asuncion Gomez-Perez. Methodologies, tools and languages for building ontologies. where is their meeting point? *Data & Knowledge Engineering*, 46(1):41 – 64, 2003. ISSN 0169-023X.

Matteo Cristani and Roberta Cuel. A Survey on Ontology Creation Methodologies. *International Journal on Semantic Web & Information Systems*, 1(2):49 – 69, 2005.

Nikolai Dahlem, Jianfeng Guo, Axel Hahn, and Matthias Reinel. Towards an user-friendly ontology design methodology. In *Proceedings of the 2009 International Conference on Interoperability for Enterprise Software and Applications China*, I-ESA '09, pages 180–186, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3652-1.

Nikolai Dahlem and Axel Hahn. User-friendly ontology creation methodologies - a survey. In *Proceedings of the 2009 Americas Conference on Information Systems*, AMCIS 2009. Association for Information Systems Electronic Library (AISeL), 2009. Paper 117.

Mathieu d'Aquin, Claudio Baldassarre, Laurian Gridinoc, Sofia Angeletou, Marta Sabou, and Enrico Motta. Characterizing knowledge on the semantic web with watson. In Raul Garcia-Castro, Denny Vrandecic, Asunción Gómez-Pérez, York Sure, and Zhisheng Huang, editors, *EON*, volume 329 of *CEUR Workshop Proceedings*, pages 1–10. CEUR-WS.org, 2007.

Mike Dean and Guus Schreiber. OWL Web Ontology Language Reference. W3C recommendation, W3C, February 2004.

William Denton. How to make a faceted classification and put it on the web. Online, November 2003. http://www.miskatonic.org/library/facet-web-howto.html.

Vladan Devedzić. Understanding ontological engineering. *Commun. ACM*, 45:136–144, April 2002. ISSN 0001-0782.

Mikel Egana-Aranguren. Ontology Design Patterns for the Formalisation of Biological Ontologies. MPhil Dissertation, Bio-Health Informatics Group, School of Computer Science, University of Manchester, 2005.

Mikel Egana-Aranguren. *Role and Application of Ontology Design Patterns in Bio-ontologies*. PhD thesis, School of Computer Science, University of Manchester, 2009.

Mikel Egana-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. ISSN 1471-2105. http://www.biomedcentral.com/1471-2105/9/S5/S1.

Mariano Fernandez-Lopez and Asuncion Gomez-Perez. Overview and analysis of methodologies for building ontologies. *The Knowledge Engineering Review*, 17(02): 129–156, 2002.

Mariano Fernandez-Lopez, Asuncion Gomez-Perez, Jerome Euzenat, Aldo Gangemi, Y. Kalfoglou, D. Pisanelli, M. Schorlemmer, G. Steve, Ljilajana Stojanovic, Gerd Stumme, and York Sure. A survey on methodologies for developing, maintaining, integration, evaluation and reengineering ontologies. OntoWeb deliverable D1.4, Universidad Politecnia de Madrid, 2002. http://www.aifb.uni-karsruhe.de/WBS/ysu/publications/OntoWeb_Del_1-4.pdf.

Mariano Fernandez-Lopez, Asuncion Gomez-Perez, and Natalia Juristo. Methontology: from ontological art towards ontological engineering. In *Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering*, pages 33–40, Stanford, USA, March 1997.

Douglas Foxvog. Instances of instances modeled via higher-order classes. In *Foundational Aspects of Ontologies*, number 9–2005 in FOnt 2005, pages 46–54, Universität Koblenz-Landau, Institut für Informatik, Universitätsstr. 1, D-56070 Koblenz, 2005.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

Aldo Gangemi. Ontology Design Patterns for Semantic Web Content. In Yolanda Gil, Enrico Motta, V. Benjamins, and Mark Musen, editors, *The Semantic Web ISWC 2005*, volume 3729 of *Lecture Notes in Computer Science*, pages 262–276. Springer Berlin / Heidelberg, 2005.

Andres Garcia-Silva, Asuncion Gomez-Perez, Mari Carmen Suarez-Figueroa, and Boris Villazon-Terrazas. A pattern based approach for re-engineering non-ontological resources into ontologies. In John Domingue and Chutiporn Anutariya, editors, *The Semantic Web*, volume 5367 of *Lecture Notes in Computer Science*, pages 167–181. Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-89704-0_12.

Fausto Giunchiglia, Biswanath Dutta, and Vincenzo Maltese. Faceted lightweight ontologies. In Alexander Borgida, Vinay Chaudhri, Paolo Giorgini, and Eric Yu, editors, *Conceptual Modeling: Foundations and Applications*, volume 5600 of *Lecture Notes in Computer Science*, pages 36–51. Springer Berlin / Heidelberg, 2009. 10.1007/978-3-642-02463-4_3.

Hugh Glaser, Harith Alani, Les Carr, Sam Chapman, Fabio Ciravegna, Alexei Dingli, Nicholas Gibbins, Stephen Harris, m.c. schraefel, and Nigel Shadbolt. Cs aktive space: Building a semantic web application, 2004.

Hugh Glaser, Ian Millard, Tom Anderson, Zoe Andrews, John Fitzgerald, and Brian Randell. A knowledge base for dependability and security research. In *DSN 2009*, June 2009.

Hugh Glaser, Ian Millard, Tom Anderson, and Brian Randell. Resist project deliverable d10: Prototype knowledge base. Technical Report, January 2007a.

Hugh Glaser, Ian Millard, Bene Rodriguez-Castro, and Afraz Jaffri. Demonstration: Knowledge-enabled research infrastructure. In *4th European Semantic Web Conference*, 2007b.

Claudio Gnoli. Facets: A fruitful notion in many domains. *Axiomathes*, 18(2):127–130, June 2008. ISSN 1122-1151 (Print) 1572-8390 (Online).

Asuncion Gomez-Perez, Oscar Corcho, and Mariano Fernandez-Lopez. *Ontological Engineering : with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. First Edition (Advanced Information and Knowledge Processing)*. Springer, July 2004. ISBN 1852335513.

Michael Gruninger and Mark S. Fox. Methodology for the design and evaluation of ontologies. In *IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing, April 13, 1995*, 1995.

Nicola Guarino and Christopher Welty. Evaluating ontological decisions with ontoclean. *Commun. ACM*, 45(2):61–65, 2002. ISSN 0001-0782.

Nicola Guarino and Christopher A. Welty. An overview of ontoclean. In Peter Bernus, Jacek Blazewics, Gnter Schmidt, Michael Shaw, Steffen Staab, and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 201–220. Springer Berlin Heidelberg, 2009. ISBN 978-3-540-92673-3. 10.1007/978-3-540-92673-3_9.

Karl Hammar and Kurt Sandkuhl. The state of ontology pattern research : A systematic review of ISWC, ESWC and ASWC 2005-2009. In *Workshop on Ontology Patterns : Papers and Patterns from the ISWC workshop*, pages 5–17, 2010.

Martin Hepp. Goodrelations: An ontology for describing products and services offers on the web. In Aldo Gangemi and Jrme Euzenat, editors, *Knowledge Engineering: Practice and Patterns*, volume 5268 of *Lecture Notes in Computer Science*, pages 329–346. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-87695-3.

Martin Hepp and Jos de Bruijn. GenTax: A Generic Methodology for Deriving OWL and RDF-S Ontologies from Hierarchical Classifications, Thesauri, and Inconsistent Taxonomies. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, *The Semantic Web: Research and Applications*, volume 4519 of *Lecture Notes in Computer Science*, pages 129–144. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-72667-8_11.

Rinke Hoekstra. *Ontology Representation - Design Patterns and Ontologies that Make Sense*, volume 197 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009. ISBN 978-1-60750-013-1.

Markus Holi and Eero Hyvonen. Modeling degrees of overlap in semantic web ontologies. In Paulo C. G. da Costa, Kathryn B. Laskey, Kenneth J. Laskey, and Michael Pool, editors, *Proceedings of the ISWC Workshop Uncertainty Reasoning for the Semantic Web*. CEUR Workshop Proceedings, Nov 2005.

Matthew Horridge, Nick Drummond, Simon Jupp, Georgina Moulton, and Robert Stevens. A practical guide to building owl ontologies using the protege-owl plugin and co-ode tools edition 1.2. Technical report, The University Of Manchester, March 2009.

Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, and Chris Wroe. A practical guide to building owl ontologies using the protege-owl plugin and co-ode tools edition 1.0. Technical report, The University Of Manchester, August 2004.

Markus Krötzsch, Peter F. Patel-Schneider, Sebastian Rudolph, Pascal Hitzler, and Bijan Parsia. OWL 2 Web Ontology Language Primer. W3C recommendation, W3C, October 2009.

Barbara H. Kwasnik. The role of classification in knowledge representation and discovery. *Library Trends*, 48(1), 1999.

Kathryn La Barre. *The use of faceted analytico-synthetic theory as revealed in the practice of website construction and design*. PhD thesis, Indiana University, 2006. AAT 3219917.

Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. W3C recommendation, W3C, February 2004. http://www.w3.org/TR/owl-features/.

I.C. McIlwaine and N.J. Williamson. A feasibility study on the restructuring of the Universal Decimal Classification into a full-faceted classification system. In H. Albrechtsen and S. Oernager, editors, *3rd International Society for Knowledge Organization (ISKO) Conference: Knowledge Organization and Quality Management*, volume 4 of *Advances in Knowledge Organization*, pages 406–413. INDEKS Verlag, Frankfurt/-Main, June 1994.

Bertrand Meyer. The many faces of inheritance: A taxonomy of taxonomy. *Computer*, 29:105–108, 1996. ISSN 0018-9162.

Bertrand Meyer. *Object-Oriented Software Construction (Book/CD-ROM) (2nd Edition)*. Prentice Hall PTR, March 2000. ISBN 0136291554.

Ian Millard, Afraz Jaffri, Hugh Glaser, and Bene Rodriguez-Castro. Using a semantic mediawiki to interact with a knowledge based infrustructure. In *15th International Conference on Knowledge Engineering and Knowledge Management*, 2006.

Jack Mills and Vanda Broughton. *Bliss Bibliographic Classification*, volume 2nd edition. Butterworth, London, UK, 1977.

Natasha F. Noy. Representing Classes As Property Values on the Semantic Web. Technical Report Note 5, W3C, Semantic Web Best Practices and Deployment Working Group, 2005.

Natasha F. Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report SMI-2001-0880, Stanford Medical Informatics, Stanford, 2001.

Natasha F. Noy and Alan L. Rector. Defining n-ary relations on the semantic web. W3C note, W3C, April 2006. http://www.w3.org/TR/2006/NOTE-swbp-n-aryRelations-20060412/.

Daniel Oberle, Phil Tetlow, Holger Knublauch, and Evan Wallace. A semantic web primer for object-oriented software developers. W3C note, W3C, March 2006. http://www.w3.org/TR/2006/NOTE-sw-oosd-primer-20060309/.

P. Otlet and H. La Fontaine. Universal Decimal Classification. *Institut International de Bibliographie, Brussels*, 1905.

Helena Sofia Pinto and Joao P. Martins. Ontologies: How can they be built? *Knowledge and Information Systems*, 6:441–464, 2004. ISSN 0219-1377. 10.1007/s10115-003-0138-1.

Maria Poveda, Mari Carmen Suarez-Figueroa, and Asuncion Gomez-Perez. Common pitfalls in ontology development. In Pedro Meseguer, Lawrence Mandow, and Rafael Gasca, editors, *Current Topics in Artificial Intelligence*, volume 5988 of *Lecture Notes in Computer Science*, pages 91–100. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-14264-2_10.

Valentina Presutti, Aldo Gangemi, Stefano David, Guadalupe Aguado de Cea, Mari Carmen Suarez-Figueroa, Elena Montiel-Ponsoda, and Maria Poveda. A Library of Ontology Design Patterns: reusable solutions for collaborative design of networked ontologies. NeOn deliverable D2.5.1, Institute of Cognitive Sciences and Technologies (CNR), 2008.

S.R. Ranganathan. *Colon Classification*, volume 1st edition. Madras Library Association, Madras, India, 1933.

S.R. Ranganathan. *Colon Classification Basic Classification*, volume 6th edition. Asia Publishing House, New York, NY, USA, 1960.

S.R. Ranganathan. *Prolegomena to Library Classification*. Asia Publishing House, New York, NY, USA, 1967.

Alan Rector. Modularisation of Domain Ontologies Implemented in Description Logics and related formalisms including OWL. In *Proceedings of the 2nd international* *conference on Knowledge capture*, K-CAP '03, pages 121–128, New York, NY, USA, 2003. ACM. ISBN 1-58113-583-1.

Alan Rector. Representing Specified Values in OWL: "value partitions" and "value sets". Technical Report Note 17, W3C, Semantic Web Best Practices and Deployment Working Group, May 2005.

Alan 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 & common patterns. In Enrico Motta, Nigel Shadbolt, Arthur Stutt, and Nick Gibbins, editors, *Engineering Knowledge in the Age of the Semantic Web*, volume 3257 of *Lecture Notes in Computer Science*, pages 63–81. Springer Berlin / Heidelberg, 2004. 10.1007/978-3-540-30202-5_5.

Alan Rector, Chris Welty, Natasha Noy, and Evan Wallace. Simple part-whole relations in owl ontologies. Technical report, W3C, August 2005.

ReSIST. Resilience and Survivability in IST. Network of Excellence. Contract: IST 4 026764 NOE, EU and Sixth Framework Programme (FP6) and Information Society Technology (IST), 2005–2008. http://www.resist-noe.eu/.

Bene Rodriguez-Castro and Hugh Glaser. Untangling Domain Concepts in Ontology Design Patterns. In *Poster at the 5th European Semantic Web Conference*, June 2008a.

Bene Rodriguez-Castro and Hugh Glaser. Whose "Fault" Is This? Untangling Domain Concepts in an Ontology of Resilient Computing . In *Fast Abstracts at the 7th European Dependable Computing Conference (EDCC-7)*, April 2008b.

Bene Rodriguez-Castro and Hugh Glaser. Whose "Fault" Is This? Untangling Domain Concepts in Ontology Design Patterns. In *Workshop on Knowledge Reuse and Reengineering over the Semantic Web in the 5th European Semantic Web Conference*, June 2008c.

Bene Rodriguez-Castro, Hugh Glaser, and Les Carr. Faceted Classification Scheme ODP. In *The 2nd Workshop on Ontology Patterns (WOP2010) at the 9th International Semantic Web Conference (ISWC)*, November 2010a.

Bene Rodriguez-Castro, Hugh Glaser, and Leslie Carr. How to Reuse a Faceted Classification and Put It on the Semantic Web. In Peter Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Pan, Ian Horrocks, and Birte Glimm, editors, *The Semantic Web ISWC 2010*, volume 6496 of *Lecture Notes in Computer Science*, pages 663–678. Springer Berlin / Heidelberg, 2010b. ISBN 978-3-642-17745-3.

Bene Rodriguez-Castro, Hugh Glaser, and Ian Millard. View Inheritance as an Extension of the Normalization Ontology Design Pattern. In *Workshop on Ontology Patterns in the 8th International Semantic Web Conference (ISWC 2009)*, September 2009.

James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-oriented modeling and design*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991. ISBN 0-13-629841-9.

Markku Sakkinen. Disciplined Inheritance. *ECOOP'89: Proceedings of the 1989 European Conference on Object-Oriented Programming*, pages 39–56, 1989.

Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The semantic web revisited. *IEEE Intelligent Systems*, 21:96–101, 2006. ISSN 1541-1672.

Nigel Shadbolt, Nicholas Gibbins, Hugh Glaser, Stephen Harris, and m.c. schraefel. Cs aktive space or how we stopped worrying and learned to love the semantic web. *IEEE Intelligent Systems*, 19(3):41–47, 2004.

A. Snyder. Inheritance and the Development of Encapsulated Software Components. In *Research Directions in Object Oriented Programming*, pages 165–188. MIT Press, 1987.

John F. Sowa. *Knowledge representation: logical, philosophical and computational foundations*. Brooks/Cole Publishing Co., Pacific Grove, CA, USA, 2000. ISBN 0-534-94965-7.

Louise Spiteri. A simplified model for facet analysis: Ranganathan 101. *Canadian Journal of Information and Library Science*, 23(1/2):1–30, 1998.

Mari Carmen Suarez-Figueroa, Saartje Brockmans, Aldo Gangemi, Asuncion Gomez-Perez, Jos Lehmann, Holger Lewen, Valentina Presutti, and Marta Sabou. Neon modelling components. NeOn deliverable D5.1.1, Universidad Politecnica de Madrid, 2007.

Mari Carmen Suarez-Figueroa, Mariano Fernandez-Lopez, Asuncion Gomez-Perez, Klaas Dellschaft, Holger Lewen, and Martin Dzbor. Revision and extension of the neon development process and ontology life cycle. NeOn deliverable D5.3.2, Universidad Politecnica de Madrid, November 2008.

Mari Carmen Suarez-Figueroa and Asuncion Gomez-Perez. First attempt towards a standard glossary of ontology engineering terminology. In Hanne Erdman Thomsen Bodil Nistrup Madsen, editor, *The 8th International Conference on Terminology and Knowledge Engineering (TKE)*, Managing ontologies and lexical resources, Copenhagen, Denmark, August 2008a. ISBN 87-91242-50-9.

Mari Carmen Suarez-Figueroa and Asuncion Gomez-Perez. Towards a glossary of activities in the ontology engineering field. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odjik, Stelios Piperidis, and Daniel Tapias, editors, *Proceedings of the Sixth International Language Resources and Evaluation (LREC)*, Marrakech, Morocco, May 2008b. European Language Resources Association (ELRA). ISBN 2-9517408-4-0.

York Sure, Steffen Staab, and Rudi Studer. On-to-knowledge methodology. In S. Staab and R. Studer (eds.), editors, *Handbook on Ontologies*, Series on Handbooks in Information Systems, chapter 6, pages 117–132. Springer, 2003.

Ewan Tempero and Robert Biddle. Simulating multiple inheritance in Java. *Journal of Systems and Software*, 55(1):87–100, 2000. ISSN 0164-1212.

Eddy Truyen, Wouter Joosen, Bo N. Jørgensen, and Pierre Verbaeten. A Generalization and Solution to the Common Ancestor Dilemma Problem in Delegation-Based Object Systems. In Robert Filman, Michael Haupt, Katharina Mehner, and Mira Mezini, editors, *DAW: Dynamic Aspects Workshop*, pages 103–119, March 2004.

Yannis Tzitzikas, Nicolas Spyratos, Panos Constantopoulos, and Anastasia Analyti. Extended faceted ontologies. In Anne Pidduck, M. Ozsu, John Mylopoulos, and Carson Woo, editors, *Advanced Information Systems Engineering*, volume 2348 of *Lecture Notes in Computer Science*, pages 778–781. Springer Berlin / Heidelberg, 2006. 10.1007/3-540-47961-9_66.

Mohammad Nasir Uddin and Paul Janecek. Faceted classification in web information architecture: A framework for using semantic web tools. *The Electronic Library*, 25 (2):219–233, 2007.

Mike Uschold and Martin King. Towards a methodology for building ontologies. In *In Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95*, 1995.

Brian Vickery. *Faceted Classification: A Guide To Construction And Use Of Special Schemes*. Aslib, 1 1960.

Brian Vickery. Faceted classification for the web. *Axiomathes*, 18(2):145–160, June 2008. ISSN 1122-1151 (Print) 1572-8390 (Online).

Denny Vrandecic and Aldo Gangemi. Unit tests for ontologies. In Mustafa Jarrar, Claude Ostyn, Werner Ceusters, and Andreas Persidis, editors, *Proceedings of the 1st International Workshop on Ontology content and evaluation in Enterprise*, LNCS, Montpellier, France, OCT 2006. Springer.

W3C. Semantic Web Best Practices and Deployment Working Group, 2004–2005. http://www.w3.org/2001/sw/BestPractices/.

Jim Waldo. Controversy: The Case for Multiple Inheritance in C++. *Computing Systems*, 4(2):157–171, 1991.

Taowei Wang, Bijan Parsia, and James Hendler. A survey of the web ontology landscape. In Isabel Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Mike Uschold, and Lora Aroyo, editors, *The Semantic Web - ISWC 2006*, volume 4273 of *Lecture Notes in Computer Science*, pages 682–694. Springer Berlin / Heidelberg, 2006. 10.1007/11926078_49.

Chris Welty, Deborah L. McGuinness, and Michael K. Smith. OWL Web Ontology Language Guide. W3C recommendation, W3C, February 2004.

Charles P. White et al. Heterarchy & hierarchy, oh my my, May 2008. ontolog-forum mailing list: http://ontolog.cim3.net/forum/ontolog-forum/.

Peter J. Wild, Matt D. Giess, and Chris A. McMahon. Describing engineering documents with faceted approaches: Observations and reflections. In *Journal of Documentation*, volume 65, pages 420–445. Emerald Group Publishing Ltd., 2009. 10.1108/00220410910952410.

Travis Wilson. The strict faceted classification model, March 2006. Presentation at the Information Architecture Summit, Vancouver (Canada).

Hai Zhuge, Yunpeng Xing, and Peng Shi. Resource space model, owl and database: Mapping and integration. *ACM Trans. Internet Technol.*, 8(4):1–31, 2008. ISSN 1533-5399.