

Linköping Studies in Science and Technology

Licentiate Thesis No. 1606

Towards an Ontology Design Pattern Quality Model

by

Karl Hammar



Linköping University



SCHOOL OF ENGINEERING
JÖNKÖPING UNIVERSITY

Department of Computer and Information Science
Linköping University
SE-581 83 Linköping, Sweden

Linköping 2013

This is a Swedish Licentiate's Thesis

Swedish postgraduate education leads to a doctor's degree and/or a licentiate's degree.

A doctor's degree comprises 240 ECTS credits (4 year of full-time studies).

A licentiate's degree comprises 120 ECTS credits.

Copyright © 2013 Karl Hammar

ISBN 978-91-7519-570-4

ISSN 0280-7971

Printed by LiU Tryck 2013

URL: <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-93370>

Towards an Ontology Design Pattern Quality Model

by

Karl Hammar

September 2013

ISBN 978-91-7519-570-4

Linköping Studies in Science and Technology

Licentiate Thesis No. 1606

ISSN 0280-7971

LiU–Tek–Lic–2013:40

ABSTRACT

The use of semantic technologies and Semantic Web ontologies in particular have enabled many recent developments in information integration, search engines, and reasoning over formalised knowledge. Ontology Design Patterns have been proposed to be useful in simplifying the development of Semantic Web ontologies by codifying and reusing modelling best practices.

This thesis investigates the quality of Ontology Design Patterns. The main contribution of the thesis is a theoretically grounded and partially empirically evaluated quality model for such patterns including a set of quality characteristics, indicators, measurement methods and recommendations. The quality model is based on established theory on information system quality, conceptual model quality, and ontology evaluation. It has been tested in a case study setting and in two experiments.

The main findings of this thesis are that the quality of Ontology Design Patterns can be identified, formalised and measured, and furthermore, that these qualities interact in such a way that ontology engineers using patterns need to make tradeoffs regarding which qualities they wish to prioritise. The developed model may aid them in making these choices.

This work has been supported by Jönköping University.

Acknowledgements

There are several individuals and organizations that have enabled and supported this research project in different ways, to which I extend my deepest gratitude.

The supervision team consisted of **Kurt Sandkuhl**, **Vladimir Tarasov**, **Eva Blomqvist**, and **Henrik Eriksson**. This large and geographically distributed team have complemented one another splendidly throughout the process and have when needed provided guidance regarding everything from research methods and writing style to academic social protocol and how to arrange workshops and other events. I've especially appreciated their hands-off approach which has enabled me to explore different directions and learn from many mistakes.

My colleagues at the Department of Computer and Electrical Engineering at Jönköping University have made the last couple of years working in research not only interesting but also fun. **Christer Thörn** and **Ulf Seigerroth** are however worthy of particular mention in that they have not only been great coworkers but have also contributed directly to the success of this work, the former by reviewing and commenting, and the latter by keeping bosses off my back and ensuring I've had time to get it done.

Fredrik R. Krohnman has provided encouragement and many good discussions, keeping up my interest in computer technologies and research, and motivating me to continue on the academic path.

A special thank you goes to my wife **Jenny**, who has supported me through good and bad, and has had to put up with many a late night spent working, without any complaints. I love you.

Karl Hammar
Jönköping, June 2013

Contents

1	Introduction	1
1.1	Motivating Factors	1
1.2	Research Questions	3
1.3	Contributions	4
1.4	Thesis Outline	5
2	Ontologies and Ontology Design Patterns	7
2.1	Knowledge Modelling and Ontologies	7
2.1.1	Data, Information, and Knowledge	7
2.1.2	Terminological and Assertional Knowledge	9
2.1.3	Ontology Components	10
2.1.4	RDF, RDFS, and OWL	13
2.2	Ontology Applications	16
2.2.1	Ontology Types	17
2.2.2	Linked Data	18
2.2.3	Semantic Search	20
2.2.4	Reasoning Tasks	22
2.3	Ontology Development	25
2.3.1	METHONTOLOGY	25
2.3.2	On-To-Knowledge	26
2.3.3	DILIGENT	27
2.3.4	Ontology Development 101	28
2.4	Ontology Design Patterns	29
2.4.1	ODP Typologies	32
2.4.2	ODP-based Ontology Construction	34
2.5	The State of ODP Research	38
2.5.1	Results	39
2.5.2	Analysis and Discussion	40
3	Evaluation and Quality Frameworks	45
3.1	Related Quality Frameworks	45
3.1.1	MAPPER	45
3.1.2	Conceptual Model Quality	46
3.1.3	Entity Relationship Model Quality	48

3.1.4	Information System Quality	50
3.1.5	Pattern Quality	53
3.2	Ontology Evaluation	54
3.2.1	O ² and oQual	54
3.2.2	ONTOMETRIC	55
3.2.3	OntoClean	56
3.2.4	Terminological Cycle Effects	58
3.2.5	ODP Documentation Template Effects	58
4	Research Method	59
4.1	A Perspective on Methods in the Computing Disciplines	59
4.1.1	Systematic Literature Review	61
4.1.2	Case Studies	63
4.1.3	Interviews	64
4.1.4	Experimentation	65
4.2	Description of the Research Process	66
4.2.1	ODP Literature Study	66
4.2.2	Initial Quality Model Development	71
4.2.3	Knowledge Fusion Case Study	75
4.2.4	Learnability and Usability Evaluations	78
4.2.5	Performance Indicator Evaluation	81
5	Initial Quality Model	85
5.1	Quality Metamodel Development	85
5.2	Quality Model Development	87
5.2.1	ISO 25010 Adaptation	88
5.2.2	Thörn's Qualities	90
5.2.3	Reuse of ER Model Quality Research	91
5.2.4	Reuse of Established Ontology Quality Research	93
5.3	Empirical Pre-studies	95
5.3.1	ODP Documentation Structure Interviews	96
5.3.2	ODP Usage Experiment and Survey	97
5.4	The Developed Initial Quality Model	98
5.4.1	Quality Characteristics	98
5.4.2	Indicators and Effects	100
6	Quality Model Evaluations	107
6.1	The Knowledge Fusion Case Study	107
6.1.1	Case Characterisation	107
6.1.2	Data	109
6.1.3	Findings	110
6.2	Learnability and Usability Evaluations	114
6.2.1	Results	115
6.2.2	Findings	117
6.3	Performance Indicator Evaluation	119
6.3.1	Literature Study	119

6.3.2	Indicator Variance in ODP Repositories	122
6.3.3	Results	128
7	Refined ODP Quality Model	129
7.1	Metamodel	129
7.2	Quality Characteristics	129
7.3	Indicators and Effects	132
7.3.1	Updated Indicators	133
7.3.2	New Indicators	136
8	Conclusions and Future Work	139
8.1	Summary of Contributions	139
8.2	Research Questions Revisited	141
8.3	Future Work	142
	Bibliography	145
	List of Figures	157
	List of Tables	159

Chapter 1

Introduction

This licentiate thesis concerns the development of a quality model for Ontology Design Patterns, with a goal towards simplifying selection and use of such patterns for non-expert ontology engineers. In the following sections the background and motivation for the licentiate project are discussed, the research questions introduced, contributions of the work summarised, and structure of the remainder of the thesis briefly described.

1.1 Motivating Factors

The work presented in this thesis falls within the Information Logistics research area, which is concerned with various types of problems pertaining to information provision and information flow. To name but a few: in the academic domain, where researchers seeking to find unexplored areas for study need structured information about what kind of research is being done and where; in product development, where keeping track of different sets of possibly conflicting requirements is necessary to avoid costly product failures; and in disaster management and recovery, where the need for rapid and correct information on which to base decisions is crucial in preventing injuries or even saving lives.

In order to explore and solve information logistics problems three different perspectives are commonly employed [1]:

- The *information demand* perspective, in which researchers study what individual or group has need of which information, in which presentation format, in a certain context or situation. This can involve factors such as the task for which information is to be used, the effect of geographical location on information demand, needs of proper timing of information delivery to avoid information congestion/overload, and many other things.

- The *information content* perspective, in which focus is on the information content that is to satisfy said information demand, and how one goes about finding, sorting, matching, and aggregating this content.
- The *information distribution* perspective, in which the considerations involve how the information is to be delivered to the place (physical or logical) where it is to be used or consumed.

These three perspectives are of course interrelated in many ways; for instance information demand governs many aspects of information content and distribution, whereas information distribution factors may limit information content and vice versa. In all three types of research ontologies can be, and have been, employed as tools.

One of the most commonly used definitions of the term *ontology* within the information sciences is attributed to Studer et al., who write that an ontology is a “*formal, explicit specification of a shared conceptualisation*” [2, p. 25]. In layman’s terms, it is a commonly agreed upon (*shared*) model of a particular domain of discourse (*conceptualisation*) that is specific and clear enough that it can be interpreted by a computer (*formal, explicit*).

Such ontologies allow organisations to formally define how they view their information, in turn enabling harmonisation of information systems across the organisation. Engineers can build systems using ontologies as specifications, or, in other cases, ontologies can be directly applied as concrete artefacts in systems defining schemas or formats of information. Returning to the three perspectives on information logistics, one can say that in these usages, ontologies define the structure of *information content* according to system requirements representing a real world *information demand*. The information harmonisation that they enable, in turn, supports *information distribution*, also governed by said *information demand*.

Ontology languages have several technical advantages over other types of data or knowledge representation languages - they are flexible and easily accommodate heterogeneous data, they are platform and programming-language independent, and being based on formal logics they can be computed on by reasoning software, allowing for the inference of new knowledge based on that which is already known. This computability capability can also help ensure the consistency and quality of information encoded using ontology languages. Examples of different types of ontology use in information logistics range from competence modelling [3] to requirements management [4] to general knowledge fusion architectures [5].

Ontology engineering is the discipline or trade of developing ontologies. Performing this trade well and developing suitable ontologies for different purposes has in the past required having both a thorough understanding of the domains under study, and a solid understanding of how these domains are best represented in terms of the logic axioms that make up ontologies. The ontology engineer has had to be both subject matter expert and modelling expert. However, over the years the knowledge modelling and

subsequently the Semantic Web research communities have put much effort into developing tools, techniques, and methods for simplifying ontology engineering, often with the expressed goal of making this work easy and intuitive enough that a domain expert may perform it with some measure of efficiency and correctness.

One such technique, proposed independently by Blomqvist and Sandkuhl [6] and Gangemi [7], is the reuse of established best practice in the form of *Ontology Design Patterns* (often abbreviated ODPs). Since their introduction in 2005, such patterns have received quite a bit of research attention, and a community has formed¹ based on the developments of these ideas as explored within the NeOn project [8]. Pattern workshops have been held at the largest academic Semantic Web and Knowledge Modelling conferences, and a number of Ontology Design Patterns have been published.

There are several proposed types of Ontology Design Patterns being studied, concerning everything from naming standards to reasoning procedures [8]. Of these pattern types, Content ODPs in particular have received significant attention. Such patterns package commonly recurring features as small ontology building blocks, to be imported and reused by Ontology Engineers in development. Content patterns are believed to aid in ontology engineering in two ways – firstly, by reducing the amount of modelling work needed for implementing common features, pattern usage ought to lower the cost in terms of time and resources for ontology engineering projects. Secondly, by promoting the encoding and reuse of best practice solutions to common modelling problems, pattern usage ought to lead to better ontologies displaying fewer modelling errors and inconsistencies. The validity of the former assumption has to the author’s best knowledge not been established, but the second is supported by some empirical evidence [9].

However, as the author has previously shown [10] (summarised in Section 2.5), the published work on Ontology Design Patterns is lacking in some aspects. While many patterns have been presented and while patterns are being used in various system development projects, there are few papers documenting and evaluating the effects of using these patterns for different purposes. Less work still has been done on the structure and design of patterns themselves, and consequently, little is known about what qualities or properties of patterns are beneficial in ontology engineering tasks, and inversely, what properties are not helpful or are possibly even harmful in such tasks.

1.2 Research Questions

This thesis aims to remedy the aforementioned lack of established knowledge on the structure and quality of Ontology Design Patterns. To guide in this

¹<http://www.ontologydesignpatterns.org>

endeavour and to provide delimitations to an otherwise very open-ended enquiry, the following research questions have been established:

1. Which quality characteristics of Content Ontology Design Patterns can be differentiated, and through what indicators can they be measured and observed?
2. How do the quality characteristics of Content Ontology Design Patterns interact and affect one another?

As can be inferred from the research questions, only *Content* Ontology Design Patterns are the subject of study of this licentiate project. In Section 2.4.1 the interested reader may learn about the NeOn typology of Ontology Design Patterns and the other types of ODPs that have been proposed. While these other types of patterns, intended for tasks such as logical reasoning or concept alignment, are indeed interesting and worthy of study, it is the author's opinion that they differ too much from the more common content patterns in both structure and usage to be studied under the same conditions. Consequently, in the remainder of this thesis (unless stated otherwise) the terms Ontology Patterns or Ontology Design Patterns both refer to Content Ontology Design Patterns per the NeOn definition.

1.3 Contributions

To aid in answering the research questions, the author has developed a quality model for Ontology Design Patterns. Such a model, in addition to providing a framework within which the research questions are studied and answered, aids ontology engineers in selecting patterns suitable for reuse in modelling for particular cases. It also illustrates trade-offs that ontology engineers may need to make, when developing or formalising Ontology Design Patterns that they find reappearing in the course of their ontology development work. Furthermore, it provides a well-founded basis for researchers wishing to further explore issues of Ontology Design Pattern quality and usage. This ODP quality model provides the following contributions:

- A conceptual understanding of quality, as it relates to Ontology Design Patterns.
- A set of Ontology Design Pattern quality characteristics, capturing the different relevant perspectives on ODP quality.
- Indicators and methods for quantifying and measuring ODP quality characteristics.
- Recommendations on suitable values for said indicators, or aspects to consider when measuring them.

1.4 Thesis Outline

The remainder of this thesis is structured as follows:

- Chapter 2 introduces basic concepts with which the reader may wish to familiarise themselves, including semantic technologies, description logics, ontology engineering methods, and Ontology Design Patterns.
- Chapter 3 introduces relevant and reusable existing works in quality models and quality frameworks for other types of data models, conceptual models, and information systems.
- Chapter 4 discusses issues of method in computer and information systems research, and gives an overview of how these methods have been applied in this thesis in order to answer the research questions.
- Chapter 5 presents an initial Ontology Design Pattern quality model, derived from literature study and small-scale pre-studies.
- Chapter 6 describes three studies evaluating and testing the initial ODP quality model.
- Chapter 7 presents a refined Ontology Design Pattern quality model, updated based on performed evaluation work.
- Chapter 8 summarises the contributions of this thesis, revisits the research questions, and reflects upon directions for future work.

Chapter 2

Ontologies and Ontology Design Patterns

The following chapter is intended for the reader who is new to the Semantic Web, ontologies, and knowledge-based systems. It provides an overview of concepts, technologies and research in the field, with a special focus on topics relevant to the work presented in this thesis.

2.1 Knowledge Modelling and Ontologies

Even though some of the technical standards for using ontologies on the Semantic Web are fairly recently developed, the use of ontologies for structuring information has a long tradition in the knowledge modelling and artificial intelligence fields. In this section some general knowledge modelling and ontology basics are first introduced, and the modern day standards of RDF, RDFS, and OWL are then briefly described.

2.1.1 Data, Information, and Knowledge

As explained in Chapter 1, this thesis is concerned with the application of Ontology Design Patterns for reuse in development of ontologies for information logistics purposes. Ontologies were in said chapter also mentioned as knowledge representation artefacts. While the words knowledge and information may appear synonymous to the layman, in knowledge management and information logistics research these two terms are often considered conceptually different, and a brief discussion on their definitions is therefore warranted.

A commonly used model of the relationship between data, information, and knowledge in these fields is the Knowledge Hierarchy, or Knowledge Pyramid, as defined by Ackoff [11] and described by Bellinger et al. [12]. By

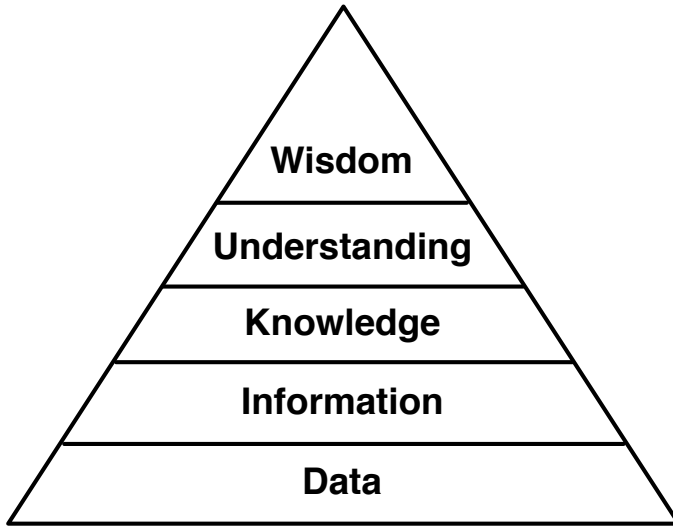


Figure 2.1: Ackoff's Knowledge Hierarchy

this model, displayed in Figure 2.1, several different levels of understanding of phenomena are defined:

- **Data** – Raw facts, with no greater meaning or connection to other facts. A spreadsheet holding cells of numbers, with no context, relation, or labelling to signify meaning, is data.
- **Information** – Data given meaning by some connection to other data. Commonly exemplified by a relational database that through foreign keys link different data rows into coherent information.
- **Knowledge** – Information collected and structured in such a way as to be appropriate or useful for some human purpose.
- **Understanding** – An understanding implies being able to analyse the underlying factors and principles behind some particular information or knowledge, and being able to extend and generate new knowledge based upon this.
- **Wisdom** – The highest level of consciousness, involving deeper analysis and probing of phenomena.

Treating the highest two levels of this model, understanding and wisdom, are at the time of writing outside of the realm of the computationally feasible, even had we known how to go about it conceptually, and we shall therefore leave them aside.

As indicated by the model, these levels build on and refine one another, such that without data, we have no information, and without information, no knowledge. Furthermore, as also indicated by the model, a relatively large amount of data can be required in order to infer a relatively modest amount of information or knowledge.

There are competing schools of thought concerning the meaning of the knowledge level in this model. There are scholars who put forward the opinion that knowledge is something which can only exist internalised in the human mind, and that it cannot be stored in some artificial construct such as a computer system. Examples include Tsoukas and Vladimirou [13] and Stacey [14], who argue that in order for knowledge to be useful in guiding human action (as per the above definition), a context is required that a computer cannot provide.

Another perspective is that of Newell [15], who reasons that knowledge certainly can be modelled and represented in a computer system and acted upon by software, in a fully automated deterministic manner. In the latter perspective, the dividing line between information and knowledge is slightly fuzzier, but essentially comes down to a matter of intent and use of information. In this thesis and in his research, the author sides with the latter perspective. Data is considered simple raw facts without context; information is data that is linked to provide a greater understanding; knowledge is information that is reasoned with by either a human or a machine, in order to perform some task. As we will see in the following sections, ontologies are well suited for use in such reasoning tasks.

2.1.2 Terminological and Assertional Knowledge

In knowledge representation tasks it is often useful to distinguish between two types of knowledge with differing characteristics and uses. There is terminological knowledge, which describes concepts and properties in the general case but without specifying individual instances of such concepts or properties. For instance, the sentences “*all cars have three or more wheels*”, or “*voltage is an attribute that describes batteries*” are both typical examples of such terminological knowledge. When these concepts and properties are then used to describe instances of things, we speak of assertional knowledge. Examples of assertional knowledge include “*my Audi A4 is a car*”, or “*this D-battery puts out 1.5 volts*” [16].

In any computer system dealing with information or knowledge this distinction between the general (a database schema, a vocabulary, a class definition) and the specific (database rows, RDF instance data, instantiated objects) is made. The former are used to structure operations on and presentations of the latter. The word *conceptualisation* is sometimes used as a synonym for the terminological knowledge of a certain domain. In Gruber’s words:

“A body of formally represented knowledge is based on a conceptualization: the objects, concepts, and other entities that are presumed to exist in some area of interest and the relationships that hold them. A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly.”
Gruber [17, p. 1]

The line demarcating terminological from assertional knowledge is often context and use dependent. For instance, had the car example given earlier instead read “*an Audi is a car*”, then the usage context would define which way the term Audi should be modelled: as an individual car manufacturer (i.e., assertional knowledge), or as a classification of all car instances matching a certain manufacturer (i.e., terminological knowledge).

Revisiting Studer et al.’s [2] ontology definition from Section 1.1 (a formal, explicit specification of a shared conceptualisation) the value of ontologies in software engineering may now be more apparent. By grouping together all the relevant terminological knowledge describing a certain area in a formal machine processable way, an ontology provides a vocabulary with which data within this area can be organised, queried for, and operated upon in an unambiguous, structured way, by humans or software programs.

2.1.3 Ontology Components

Different ontology languages support different types of features, and even to the degree that they share features, often use different terminology for describing them. In this thesis, the author uses the Semantic Web stack of languages and standards, as described in Section 2.1.4. Within these languages, the basic building blocks are classes, properties, and individuals. The following sections describe these building blocks in brief. Figures 2.2 and 2.3 are used to graphically illustrate the concepts. In these figures, rectangles denote classes, rounded rectangles denote properties, ellipses denote individuals, and diamonds denote simple data values. The prefixes associated with some concepts in the figures indicate which namespace the concepts are defined in, that is, whether they belong to the RDF, RDFS, or OWL standards (these standards are introduced in Section 2.1.4).

Classes

Classes are a way of grouping together things that are similar in some respects, such that individuals can be asserted to belong to them. Depending on which type of ontology language is employed, classes can be viewed as extensional (i.e., sets that are defined by their constituent individuals) or as intensional, (i.e., with a defined meaning independent of any member individuals). In the latter case, one might assert that the class *Car* has the

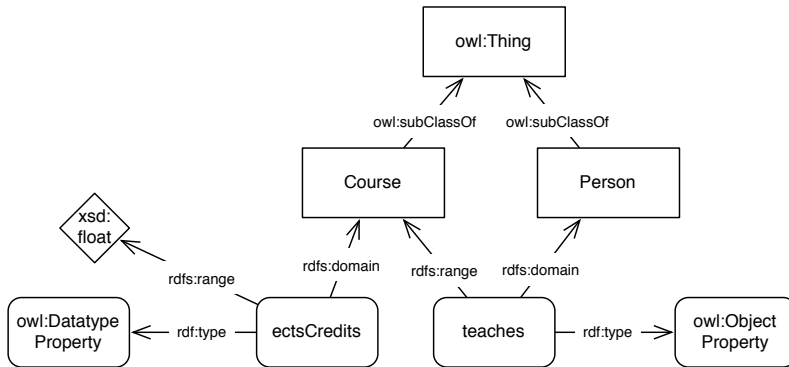


Figure 2.2: Course ontology example

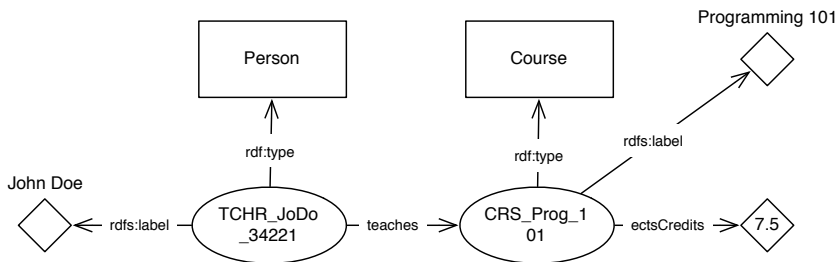


Figure 2.3: Course data expressed using the ontology in Figure 2.2

intentional definition “a four-wheeled vehicle with an internal combustion engine”. This definition then holds true no matter whether there are zero or one million individuals asserted to be cars. In the OWL language, the class concept is defined as being intensional, as per the latter perspective. One of the main tasks of a reasoner software is to sort individuals into classes based on the properties that they exhibit and the intensional definitions of the classes [18, 19].

Classes can be related to one another through equivalence or subsumption relations, such that a certain class can be defined as being a subclass of another class, or as being extensionally equivalent to it. The notion of subclasses and subsumption is closely related to the view of a class as a set of individuals, in that the individuals belonging to a subclass by definition are a subset of the individuals belonging to the superclass. Sub- and super-classing is transitive, such that if a superclass *A* has a subclass *B*, and *B* in turn has a subclass *C*, then it holds that *C* is also a subclass of *A*, transitively. In many languages there exist a defined top class (called *Thing*, *Top*, or something similar) which all other classes are subclasses of and which, consequently, all individuals are members of. In Figure 2.2 the classes *Course* and *Person* are defined to be direct subclasses of the top-level class *Thing* [18, 19].

In other knowledge modelling languages classes are known varyingly as concepts, types, categories, etc. In this thesis the terms class and concept are used interchangeably.

Properties

Properties (or *relations* as they are also known) define the links that can hold between two individuals of different classes or between an individual and a data value. They are, together with the class subsumption hierarchy, the main way of defining the semantics of the domain of discourse.

Some languages, including OWL, differentiate between properties that relate individuals to data values (datatype properties) and properties that hold between two individuals (object properties) [19]. Other languages, such as Protégé-Frames, do not distinguish between the two types of properties, but treat both as simple slots on a class definition that can be filled out by an individual or a data value. In both formalisms, properties are defined to hold over some domain(s) (i.e., be applicable to certain classes) and have some range(s) (i.e., are satisfied by links to some other classes, or data types). In Figure 2.2, the properties *ectsCredits* and *teaches* are defined. The former is a datatype property with the domain *Course* and range *float*. The latter is an object property with the domain *Person* and range *Course* [20].

Individuals

Individuals are the basic entities in an ontology-backed knowledge base, and represent some individual fact or resource. While they are most often treated

and modelled as part of the assertional knowledge part of such a knowledge base rather than the terminological knowledge, there are some cases when it makes sense to refer to individuals in an ontology. One such case is when defining classes extensionally, i.e., by an explicit listing of member individuals. Another is when defining classes based on value restrictions, that is, saying that a class consists of all individuals that have some relation R to a specific defined individual. Individuals are sometimes, in other works and in the following text, referred to as *instances* or *objects*. In Figure 2.3 two individuals are defined to exist, are labelled in a human-readable manner (*John Doe* and *Programming 101*), are stated to belong to the relevant classes, and to be connected via the *teaches* property such that *John Doe teaches Programming 101*. Furthermore, it is stated that *Programming 101 covers 7.5 ECTS credits*, via the *ectsCredits* property.

2.1.4 RDF, RDFS, and OWL

In the 1980s and 90s there were for a long time multiple competing and non-interoperable knowledge representation formats and knowledge bases, representing different directions of research taking place at research groups and systems vendors. Then, in 2001, Tim Berners-Lee et al. published the article calling for development of a new Semantic Web [21], via which humans and computers alike could find, consume, and reason over published knowledge. What Berners-Lee saw was that this vision of the future Web could never come to fruition unless decentralised and open knowledge representation systems were developed, systems in which no single node should be required to hold all knowledge, but where knowledge could be merged from different systems knowing parts of the truth. For such a process to work, interoperability standards were obviously required, and the W3C set about developing such standards over the course of the following decade. The existing RDF data model was used as a foundation, and was developed further along with the SPARQL, RDFS, OWL, and RIF standards, among others. Figure 2.4 gives an overview of the structure of the Semantic Web stack as it stands today. The following section gives an introduction to some of the layers of the stack.

RDF

The Resource Description Framework (RDF) standard was originally released as a W3C Recommendation in 1999, and was updated in 2004. The RDF standard consists of two major components: a data model and language for representing distributed data on the Web, and syntax standards for expressing, exporting, and parsing said data model and language [23].

The RDF data model is based on graphs, as opposed to the tuples that underlie traditional relational data models. Under RDF, a data graph is constructed by the union of a number of three part assertions called *triples*.

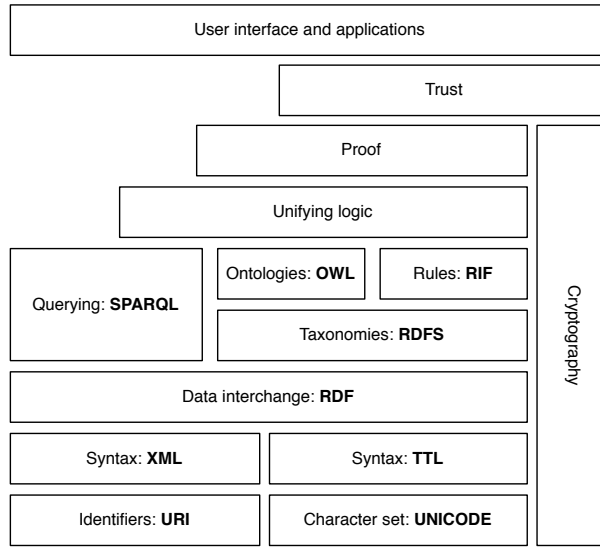


Figure 2.4: The Semantic Web layer cake (adapted from [22])

A triple consists of a *subject*, a *predicate*, and an *object*, in which the subject is an entity about which some data is expressed, the predicate can be seen as the typing of the related data, and the object is the actual related data relevant to the subject. For example, Listing 2.1 shows in a simplified syntax four triples extracted from the graph displayed in Figure 2.3. *Programming_101* and *John_Doe* are subjects, *type*, *ectsCredits*, and *teaches* are predicates, and *Course*, *7.5*, *Person*, and *Programming_101* are objects.

Listing 2.1: RDF triples example

```
Programming_101 rdf:type Course
Programming_101 rdfs:label 'Programming 101'
Programming_101 ectsCredits 7.5
John_Doe rdf:type Person
John_Doe rdfs:label 'John Doe'
John_Doe teaches Programming_101
```

As illustrated in this example and in Figure 2.3 subjects and objects make up the nodes in the RDF graph, and predicates make up the edges linking the nodes together. We can also see that there are two types of nodes in such a graph: resources (entities such as *Course* and *John_Doe*) and literals (data values, including floating point values such as 7.5, strings such as “John Doe”, or other XML schema datatypes). Predicates are in fact also resources, enabling them to act as subjects or objects (i.e., nodes) when needed for meta-modelling purposes. RDF also defines a particular

predicate, *rdf:type*, which implies a type relationship between the two resources that are linked via it. However, the semantics of typing in pure RDF is rather vague, and one has to go to higher-order languages such as RDFS and OWL to model class extensions as discussed in Section 2.1.3.

All resources are in RDF referenced using URIs (not shown in the example), enabling global lookup of distributed knowledge via HTTP, FTP, or other distribution mechanisms supported by the URI standard. In order to simplify modelling, namespaces are used to group related content. This also provides an easy extension mechanism to RDF, which is used by RDFS, OWL, and other standards, covered in the following sections.

The RDF syntax standards describe how these triples are serialised into files. There are currently two main standards for this task, XML/RDF and Turtle. The former standard was defined at the time RDF was developed, and works on the principle of embedding RDF structures in XML. This provides interoperability with existing XML-based infrastructure and tools, but generates rather complicated files that are difficult to parse and understand by human readers. The latter standard is newer and takes a different approach, by providing a set of convenient short-hands for writing down a large number of triples in simple text files. At the time of writing both of these standards are supported by most tools and programming frameworks in use. In this thesis, to the extent that RDF data is shown, the Turtle format will be used due to its superior readability.

RDFS and OWL

The RDF Schema (RDFS) standard, defined along with the second generation of RDF in 2004, defines a number of classes and properties that extend the base RDF vocabulary and provides support for more expressive knowledge modelling semantics. Some of the key additions in RDFS include [24]:

- **rdfs:class** – defines the concept of a class to which resources may belong, strengthening the definition of the RDF type predicate.
- **rdfs:subClassOf** – defines that a certain class is subsumed by a superclass, and that consequently, all instances of the subclass are also instances of the superclass
- **rdfs:domain** – defines a class of instances that may act as subjects to a certain predicate.
- **rdfs:range** – defines a class of instances that may act as objects to a certain predicate.

Using the RDFs vocabulary it is possible to model complex data structures, including basic ontologies. The language allows for some reasoning and inferencing, based on domains and ranges of employed properties, or

subclass and subproperty assertions. As pointed out by Lacy in [25], the RDFS language does however have some restrictions in expressivity that prevents it from being able to express richer ontologies. For instance, RDFS provides no way of expressing limitations on property cardinalities, or class extension equivalences. The Web Ontology Language (OWL) was developed simultaneously with RDFS in order to provide better support for such higher expressiveness. Some key features of OWL include [26]:

- **class and property equivalences** – defining that two classes or two properties are synonymous, such that all instances of one are also instances of the other. This is a key feature in implementing integration between distributed ontologies where classes or properties are defined by different URIs at different knowledge sources, but are in fact semantically equivalent.
- **sameAs and differentFrom** – defines individual equivalence or disjointness. As with the above point, this is important in integrating distributed datasets where individuals may have different URIs but in fact refer to the same information.
- **disjointWith** – defines class disjointness, i.e., that two defined classes may not have any joint individuals.
- **inverse, transitive, and functional properties** – in OWL, a great deal can be said about the semantics of properties that is not possible to express in RDFS. Transitive properties in particular are important in modelling classification trees, where descendant nodes many steps down the tree can be inferred to be related to higher nodes via them.
- **property cardinality restrictions** – delimits the number of times a predicate may occur for a given subject, such that for instance a car can be defined to have a maximum of four wheels, or a parent a minimum of one child.

Since its original release, OWL has seen widespread adoption as an ontology engineering language in the research community and industry alike. A number of new features (keys, property chains, datatype restrictions, etc.) were added to the standard when it was updated in 2009 [27].

2.2 Ontology Applications

As previously touched upon, ontologies are of use in various tasks related to the organisation and distribution of information. The following section describes different types of ontologies, and exemplifies how ontologies are being used for some different purposes. The usage areas exemplified have been selected because of the potential benefit that ODP usage could bring to them – they all concern situations where modelling and management

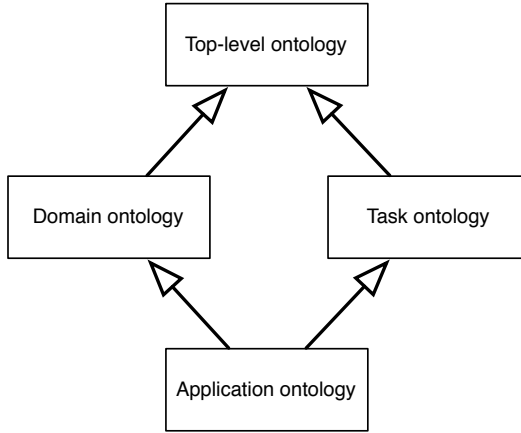


Figure 2.5: Guarino’s ontology classification hierarchy [28]

of knowledge could be performed by domain experts rather than ontology engineers. In publishing Linked Data, or applying Semantic Search engines, these domain experts have an understanding of what types of gains could be had by integrating, reusing, or searching over their information, that an ontology engineer would not necessarily have. In deploying different types of reasoning systems, whether it be for purposes of Complex Event Processing, profile matching, or ubiquitous computing, system users and administrators being able to themselves develop the ontologies that govern system behaviour, would be superior to handing off such configuration tasks to an ontology engineer.

2.2.1 Ontology Types

When classifying or structuring ontologies, one common approach is to organise them by intended usage domain, such that biomedical ontologies are differentiated and studied separately from for instance business process modelling ontologies or library ontologies. This is likely the result of differing academic disciplines picking up ontology modelling for different purposes. When dealing with reuse and patterns, such a view on ontology classification can be counterproductive. After all, a pattern is supposed to be a reusable component, ideally reusable across domain boundaries.

The categorisation presented by Guarino in [28] and displayed in Figure 2.5 is of another kind, differentiating between ontologies based on their level of generality. The intuition underlying this hierarchy is that it can be difficult to reconcile existing ontologies from a bottom-up perspective, but that certain top-level concepts are general enough that they can be agreed upon regardless of domain. Thus, the *top-level ontologies* in the model cover

very general things such as space, time, tangible or intangible objects, and so on, independent of any particular use case or usage domain. These top-level ontologies can then be used as a foundation to construct either *domain* or *task* ontologies. The former are ontologies specialised to cover a given domain (banking or the academia, for instance) irrespective of what task one wishes to use the ontologies for. The latter are ontologies specified for a generic task (such as content annotation or situation recognition) irrespective of usage domain. Finally, *application ontologies* are developed to help solve a particular tasks within particular domains, and therefore often reuse and build upon both domain and task ontologies. This perspective on ontology classification has seen significant adoption in the research community.

2.2.2 Linked Data

There are vast amounts of data stored at both government institutions and private corporations, which could be published on the Web for citizens or customers to access, query, and work with. However, simply publishing that data online brings less benefit than if a few more steps are taken. The goal of the Linked Data community (originally a W3C project) is to promote the publication of data that follows these Linked Data principles, as outlined by Berners-Lee [29]:

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)
4. Include links to other URIs, so that they can discover more things.

Datasets published according to these principles can easily be integrated with other linked datasets on the Web, helping users query across the totality of the available data (which in the Semantic Web vision is the whole Web). Several organisations and institutions have recognised that by allowing users and customers get access to data in this manner, those users can help in constructing innovative analyses, visualisations, and interpretations of the data that the host organisations could not themselves have produced. Furthermore, to the extent that the host organisations are government agencies, there are political and philosophical points to be made that data produced using tax-payers' funds should be made available to said tax-payers.

While ontologies are not strictly speaking required in order to develop and publish linked data, they are essential to doing so in an efficient and interoperable manner. By sharing ground definitions regarding the structure of data, each linked data provider does not have to individually construct schemas for their data, but instead existing ontologies can be used. As

an example of this, the FOAF ontology¹ is almost exclusively used when publishing data about individuals or organisations.

DBPedia² is probably the most influential linked data source on the Web. It consists of structured data sourced from Wikipedia, in particular from its article infoboxes, which are known to follow a certain structure and schema, and are therefore easy to extract data from. Due to the impressive size and coverage of Wikipedia, a great deal of real world entities (just short of four million at the time of writing) are covered by DBPedia and consequently, have assigned DBPedia URIs. These URIs are very often used for interlinking purposes by other linked data providers who wish to enable lookup of related information from their data. As a result, DBPedia has become a foundational element to the Web of Data, as illustrated by the Linking Open Data diagram³, in which DBPedia is positioned at the center. The DBPedia ontology mirrors the structure of the infoboxes on Wikipedia, and contains some 350 classes and more than 1700 properties.

A particularly important Linked Data player, both in terms of the number of datasets published and in terms of public awareness impact, is the United States government, through its Data.gov initiative⁴. This website gathers the datasets published by the US federal government agencies, over 4700 at the time of writing, of which some 2500 are considered high-value (that is, usable for improvement suggestions regarding agencies' operations, accountability, and responsibility). Furthermore, some 1300 tools (of which 500 classes as high-value ones) to make use of these datasets are published via the site, covering everything from airline on-time arrival data to FDA recalls to USGS geospatial data. Unfortunately the Data.gov datasets are only to a limited degree expressed as RDF, and in few cases linked to existing linked data sources. Ding et al. have been instrumental in overcoming this divide by helping interlink government data to public linked open data sources such as DBPedia [30, 31].

A pedagogical example of ontology integration to support linked data publication and consumption is that provided by the Semantic Web Dog Food Corpus⁵. This site gathers metadata about Semantic Web conferences. The ontology used to structure this dataset is a combination of FOAF, SWRC⁶, SIOC⁷, and Dublin Core⁸. Its ontology structure can be seen as a partial validation of the Guarino [28] ontology hierarchy mentioned in 2.2.1, in that none of these domain or task ontologies on their own provide an appropriate vocabulary for structuring Semantic Web conference metadata, but when combined into an application ontology, the task can be solved.

¹Friend-Of-A-Friend, <http://www.foaf-project.org>

²<http://dbpedia.org>

³<http://lod-cloud.net/>

⁴<http://www.data.gov>

⁵<http://data.semanticweb.org>

⁶Semantic Web for Research Communities, <http://ontoware.org/swrc/>

⁷Semantically-Interlinked Online Communities, <http://sioc-project.org>

⁸<http://dublincore.org>

2.2.3 Semantic Search

A common problem in information logistics is finding the correct information needed for performing some task or fulfilling some role. The two main options open to a modern day knowledge worker are all too often to either step by step search through some file server directory structure and try to find a folder or filename that looks reasonable, or to run a full text search using some document management system, often returning hundreds of hits. Neither of these two approaches allow the knowledge worker to query over the information content of the documents in question. Semantic search methods, as described below, aim to solve this problem in different ways.

Semantic fact search

One of the earlier and very influential papers on semantic search, [32] by Guha et al., proposes a new type of search by which users may search over the Semantic Web to find knowledge triples related to a particular entity or concept. The authors exemplify the utility of such a search by augmenting existing Google searches with facts about the recognised entities from the search results. As suitable querying languages did not exist at the time of the paper's writing, the authors develop their own API for querying remote servers for RDF data via SOAP. Their `GetData()` call queries a server for all resources matching a submitted RDF subject and predicate combination. In order to look up the initial subject URI to query for from a given search string, a lookup via the TAP knowledge base is performed. This basic approach to semantic fact search, i.e., first finding a canonical URI corresponding to a search string, and then querying known knowledge bases to aggregate more RDF triples involving this URI, is still in use in modern solutions, though it is now more common to use SPARQL as query language and DBpedia-based entity names. Sometimes these systems present the data gathered alongside documents found using traditional search methods, and sometimes they return RDF triples exclusively.

Uren et al. discuss approaches and methods for semantic search extensively in [33]. They classify three different types of queries over semantic facts, which they name the search for *entities*, searches for *relations*, and *parametrised* searches. Entity search is the search for more information regarding some RDF resource, as exemplified by the search method developed by Guha et al., mentioned above. This is the simplest type of semantic query. Relation-based search looks to find the path connecting two RDF resources, i.e., how two known concepts or individuals are connected in a dataset. Parametrised search, finally, is used when the user has a clear and formally defined need for knowledge that is consistent with the ontology by which the data is expressed. This enables the user to create templates (including parameters) that can be applied to an RDF graph in order to “stamp out” those parts of the RDF graph that are consistent with the parametrised search query. For instance, a query could be constructed ask-

ing for the CEOs of all companies that are in the telecom sector and that has some office in Asian countries, and the result would be a set of subgraphs of the original dataset that match these conditions. Parametrised queries provide a great expressivity, but they can be difficult to build intuitive and usable user interfaces for, because of their complexity and the many form fields or logic expressions required for constructing such a query.

Text-based search with annotations

The most common usage of semantic search engines is in enriching search results across document repositories or the Web by using semantic annotations. In this method, the documents over which the search is executed are indexed not only by their textual content, but by the semantic meaning of parts of that content. An example of this type of search is the method presented by Kyriakov et al. in [34]. In their solution, primarily geared towards business news content, individual news articles are crawled and annotated using various information extraction methods for entity and property recognition, such that indexes over these documents include not only the document content but also extracted individuals, and the annotations matching documents to those individuals. This method allows users of their system to query for both instance data extracted from the news articles, and for news articles that mention particular instance data. Combining these two types of searches yields a method where users can first search for the facts that they are interested in (as in the aforementioned semantic fact search approaches), and then bring up the documents related to those facts. The KIM platform developed by the authors has been released as a dual licensed software product released by OntoText⁹.

While the KIM platform provides good functionalities for searching, it is like many other semantic search softwares, somewhat lacking in terms of usability and intuitiveness. Many of these systems have interfaces that assume that users are already familiar with graph shaped data, ontologies, and Semantic Web technologies. In [35] Lei et al. attempt to develop a system that hides much of the logic formalisms behind a simple Google-like search interface. This interface makes use of a controlled natural language to help formalise user queries into SeRQL queries on the backend, such that users can pose queries in pseudo-english (with certain restrictions) as opposed to themselves creating a formal query. Their engine then uses the generated queries to return results from a knowledge base of metadata extracted from a web portal. The returned facts include pointers to the source document that this metadata was originally extracted from, allowing the user to get access to documents as opposed to only semantic facts.

Both of the systems described in [34] and [35] make use of information extraction techniques to retrieve metadata from published documents. Such extraction can be simplified significantly if the source documents comply

⁹<http://www.ontotext.com/kim>

with some metadata structure to begin with. The `schema.org`¹⁰ vocabulary is an attempt to standardise such a metadata structure, by providing a set of simple vocabularies for web content. This initiative is backed by some of the major search engines on the Web, including Google, Bing, and Yahoo. While the vocabularies provided through `schema.org` are not detailed enough to fit all purposes, they provide a good starting point for developing web site template and content from which supports semantic search features.

Hybrid search

An exciting recent development in semantic search is the use of hybrid search techniques, as exemplified by Bhagdev et al. in [36] and [37]. In these techniques, traditional search methods and semantic search methods are combined to provide better results. For instance, if only part of a user-entered query is suitable for use in a semantic search, that part of the query can be processed by a semantic search system, and the remainder of the query be processed using traditional keyword search methods. The final results are then computed by joining the two result sets. This enables context-based keyword search, such that for example, documents containing the phrase “salary” are returned provided that the documents’ metadata asserts that they concern an individual who is classified as an instance of the class “Professor” and who has an employment at a particular university.

2.2.4 Reasoning Tasks

The description logic foundations of Semantic Web ontologies makes them suitable for a variety of uses where the logic consequences of a certain set of data or knowledge needs to be computed. This type of computation is most often performed using a Semantic Web reasoning engine, typically capable of a number of reasoning tasks including consistency checking, subsumption calculation (i.e., which classes subsume one another in the inheritance hierarchy), individual realisation (calculate which classes an individual belongs to), and concept satisfiability (whether a certain class is defined in such a way as to allow individuals to exist). Additionally, system- or case-specific logic is often added in application code or rules, to enforce or test for particular relations and facts when the base ontology language does not suffice. The following sections illustrates some such advanced usages of ontologies and semantic technology.

Complex Event Processing

Complex Event Processing (CEP) is a set of methods for scanning through time-indexed data in order to detect patterns signifying the presence of particular events or situations that are of interest. The idea is initially introduced by Luckham and Frasca in [38]. In their approach, patterns based

¹⁰<http://schema.org/>

on temporal or causal links between events are defined and formalised into mapping rules. When executed over incoming time-indexed data streams, these patterns connect lower level basic events to form higher level complex events. The approach is exemplified by a factory scenario, where the events concern communication with automated production machinery and mappings can be made between low-level network communication events and higher level workflow events such as “*begin process*” or “*setup machine*”. Other possible usages of CEP exist in a variety of areas, from improving operational efficiency in healthcare [39] to dynamic adaption of business process models [40].

As indicated by Anicic et al. in [41] traditional CEP approaches however have some drawbacks, particularly in terms of recognising events using background knowledge. Only those relations between events and entities which are made explicit in the input data stream can be used for detection and correlation purposes. In order to overcome these limitations, [41] suggests the use of Semantic Complex Event Processing (SCEP), in which background knowledge is encoded into knowledge bases that are accessed by a rules engine to support CEP. Apart from enabling reasoning over domain and background knowledge, this also enables detection of more complex situations, recommendations, event classification, clustering, filtering, etc.

Another approach to enabling semantic processing of time-indexed data is proposed by Barbieri et al. in [42] and further developed in [43] and [44]. Their contribution is twofold – to begin with they propose an extension of the SPARQL query language commonly used to query knowledge bases, enabling continuous querying over timestamped RDF graphs using configurable sliding windows. They also develop support for reasoning over such sliding windows, including dropping both facts and inferred knowledge once the window has passed (greatly reducing the computational power required to calculate inferences). Based on these approaches it is possible to construct SCEP systems using only semantic technologies.

Profile Matching

An interesting type of task made simpler by semantic technologies and ontologies is profile matching for different purposes. By developing a profile representing user needs, interests, capabilities, or other relevant information, and then at runtime matching that profile to associated data, it becomes significantly easier to provide the correct user with the correct data or task. While this type of work can be performed using traditional technologies, reasoning engines and ontology-backed knowledge bases are particularly suitable options, given their focus on classification operations as discussed earlier.

One example of this type of profile matching is the approach used by Tarasov [45] for competence profile management. Tarasov defines a formal logic vocabulary for modelling competence profiles, including concepts such as competencies, roles, processes, tasks, etc. He then defines which types

of operations this vocabulary and accompanying software needs to be able to perform, in order to support competence management in an enterprise. The logic vocabulary proposed is directly translatable into description logic concepts, and form the basis for a developed OWL ontology, while the operations required are translatable into SPARQL queries across said ontology. The developed system can be used to check that workers are of a sufficient competency for their tasks, to find suitable workers for new tasks, or to generate an aggregate overview of the competencies of an organisation.

Another example of profile matching is in matching of information to an interested party, as exemplified by Billig et al. in [46]. In this approach an ontology is constructed, covering the concepts which occur within the organisation or domain of study. Each user of the system is associated with a profile representing the concepts from said ontology which they are interested in. When a document is entered into the system, an information extraction system extracts which concepts are mentioned in the document, and the system then attempts to find the user profile with the least semantic difference to said document.

Ubiquitous computing

Berners-Lee's original vision for the Semantic Web [21] exemplifies the usage of a meaningful machine-interpretable Web through a ubiquitous computing scenario, where two parties schedule a set of activities based on resource availability and contextual restrictions, all through handheld or car-integrated devices operating with a large degree of autonomy and what we might for lack of a better word call intelligence. The paper lays out how a Semantic Web using ontologies can be used by such agent systems to help make human life more convenient, by removing tedious data lookup and integration work. Not only that, but by using inferencing engines and distributed knowledge bases, such systems could also help humans make better choices based on asserted or inferred information that they would otherwise not have easy access to.

Several systems have been developed that try to fulfil this vision, see for instance [47] and [48]. These types of systems generally model two (sometimes overlapping) areas: a usage domain and a usage context. The former concerns the types of operations and/or data that the ubiquitous computing system needs to support, such as scheduling appointments, monitoring environmental factors, supporting particular business processes, etc. The latter concerns the contexts in which the operations take place and in which the system needs to be able to support activities. In both of these types of modelling the use of ontologies allows for harmonisation of the formats in which data and knowledge are exchanged between interacting systems. Inferring the presence of a certain usage context and what consequences this usage context has on an ongoing activity is a typical use of a semantic reasoner.

2.3 Ontology Development

A variety of different methods and practices for ontology engineering have been developed in academia. While this thesis does not have enough room to cover and discuss all of them, a subset of commonly mentioned and discussed methods have been selected for presentation below. It is important to note that nearly all of these methods require that ontologies are created either by experienced ontology engineers on their own, or by ontology engineers and domain experts in cooperation. Few of them support domain experts developing Semantic Web ontologies on their own, which as we will see in the next chapter is one of the motivations behind the development of ODPs.

In all of the presented methods, requirements engineering plays an important role. In an ontology engineering context, requirements are often formalised into competency questions (often abbreviated CQs). Competency questions are introduced by Gruninger and Fox [49] as a set of problems that the logic axioms of an ontology must be able to represent and solve. In [49] a number of such competency questions are given as examples, including planning questions (*“what sequences of activities must be completed to achieve some goal?”* [49, p. 5]) and temporal projection (*“given a set of actions that occur at different points in the future, what are the properties of resources and activities at arbitrary points in time?”* [49, Ibid.]). For simple communicative purposes such questions are often presented in natural language format, but according to the Gruninger and Fox perspective, they must be formalisable into machine interpretable and solvable problems. In RDFS and OWL ontologies, competency questions are often formalised into SPARQL queries, and are considered satisfied if said SPARQL query returns the expected result when executed over the ontology in question.

2.3.1 METHONTOLOGY

The METHONTOLOGY methodology is presented by Fernández et al. in [50]. It is one of the earlier attempts to develop a development method specifically for ontology engineering processes (prior methods often include ontology engineering as a sub-discipline within knowledge management, conflating the ontology specific issues with other types of issues as knowledge acquisition). Fernández et al. suggest, based largely on the authors’ own experiences of ontology engineering, an ontology lifecycle consisting of a number of sequential work phases or *stages*: *Specification*, *Conceptualisation*, *Formalisation*, *Integration*, *Implementation*, and *Maintenance*. Supporting these stages are a set of support activities: *Planification*, *Acquiring knowledge*, *Documenting*, and *Evaluating*.

Implementing this general ontology lifecycle into an actual ontology development methodology, the following concrete development activity steps are proposed (and motivated by reference to empirical or theoretical sources):

1. Specification – In which a requirements specification for the ontology project is developed, including details on intended usage, level of formality, scope, etc.
2. Knowledge Acquisition – In which various sources of knowledge, including experts, books, documents, figures, tables, etc. are studied to gather the knowledge required to understand the domain and concepts therein.
3. Conceptualisation – In this step the gathered domain knowledge is structured in a glossary of concepts, instances, verbs, properties, etc. METHONTOLOGY proposes a conceptual intermediate representation format suitable for comparison of different ontologies independent of eventually used implementation language.
4. Integration – In order to speed up development, the reuse of existing ontologies and meta-ontologies (i.e., foundational vocabularies) is recommended whenever possible.
5. Implementation – In this step, the results of the aforementioned steps is codified into a formal language.
6. Evaluation – In which an ontology is validated against the original requirements specification, and verified to be formally correct.
7. Documentation – Unlike previously listed activities, the documentation activity takes place throughout the whole lifecycle process, in which a variety of documents detailing the work performed and the functionality developed are created.

The steps defined are rather coarse-grained and give guidance on overall activities that need to be performed in constructing an ontology. Fine-grained and specific task or problem solving guidance is not included, but it is rather assumed that the reader is familiar with the specifics of constructing an ontology.

METHONTOLOGY does not explicitly define or differentiate between the different roles involved in an ontology engineering project. In the text describing the different steps, field experts are mentioned as being involved in the knowledge acquisition step, but then only as sources of knowledge, not active participants in the ontology engineering process itself. In this way the method may prove helpful for ontologists looking to structure their work, but it is likely less useful in terms of helping improve semantic technology and ontology adoption among non-ontologists.

2.3.2 On-To-Knowledge

The On-To-Knowledge Methodology (OTKM) [51] is, similarly to METHONTOLOGY, a methodology for for ontology engineering that covers the

big steps, but leaves out the detailed specifics. OTKM is framed as covering both ontology engineering and a larger perspective on knowledge management and knowledge processes, but it heavily emphasises the ontology development activities and tasks (in [51] denoted the *Knowledge Meta Process*).

The method prescribes a set of sequential phases: *Kickoff*, *Refinement*, *Evaluation*, and *Application and Evolution*. These phases may be iterated over cyclically in larger or longer-running projects, such that output from an *Application and Evolution* phase may be input into a new *Kickoff* phase.

OTKM requires collaboration between domain experts and ontology engineers in the *Kickoff* phase, where an ontology requirements specification document (ORSO) and an initial semi-formal model is developed, and where representatives of both groups need to sign off on these artefacts sufficiently covering all requirements. In the subsequent *Refinement* phase an ontology engineer on their own formalises the initial semi-formal model into a real ontology, without aid of a domain expert. Once the ontology engineer is satisfied with the developed ontology fulfilling requirements, the phase is finalised and the *Evaluation* phase begins. In evaluation, both technical and user-focused aspects of the knowledge based system in which the ontology is used, are evaluated. The former aspects are assumed to be evaluated by an ontology or software engineer ([51] leaves this question unanswered but it is a reasonable assumption to make), while the latter are to be evaluated together with end-users, from the perspective of whether the developed solution is as good or better than already existing solutions. Finally, the *Application and Evolution* phase concerns the deployment of said knowledge based system, and the organisation challenges associated with maintenance responsibilities.

It is interesting to note that in this methodology also, the role of the domain expert is rather limited. It is assumed that a dedicated ontology engineer will perform the knowledge modelling tasks, with input from the domain experts early in the process (when formalising requirements), but later involvement of said domain experts is limited.

2.3.3 DILIGENT

DILIGENT, by Pinto et al. [52, 53], is an abbreviation for *Distributed, Loosely-Controlled and Evolving Engineering of Ontologies*, and is a method aimed at guiding ontology engineering processes in a distributed Semantic Web setting. The method emphasises decentralised work processes and ontology usage, domain expert involvement, ontology evolution management, and fine-grained methodological guidance. Pinto et al. differentiate between ontology engineers and knowledge engineers on the one hand, and ontology users on the other. In their view, the core of an ontology needs to be created by the former group of logic and knowledge experts (in cooperation with domain experts), but adaptations of the ontology are best performed by the latter group, the users who have direct personal knowledge of the specific

uses to which the ontology will be put. These implemented user adaptations may at times for maintenance and evolution reasons need to be back-ported into the core ontology, and such integration work should, to ensure quality and consistency, be performed by a control board of knowledge experts.

This distributed development process is formalised into five activities: *build*, *local adaptation*, *analysis*, *revision*, and *local update*. In the *build* phase, the ontology experts create the initial ontology. In the subsequent *local adaptation* phase, ontology users are allowed to copy and update their local variants on this ontology. In the *analysis* phase the central control board analyse the local variants that have been developed, to find similarities and candidate features for inclusion in the shared core. In the *revision* phase, the core is updated according to this analysis. Finally, in the *local update* phase, the updated core is pushed out to the ontology users, and their local variants are updated to remain compatible and compliant with the new version of the core. In order to support all of these steps a collaborative ontology engineering environment is required to be used.

In evaluating this approach in two case studies, Pinto et al. [52, 53] find the involvement of knowledge or ontology engineers throughout the development process to be crucial. In analysing user and group interaction in ontology engineering using a collaborative ontology engineering environment, they find that having an experienced moderator restricting and guiding user discussion is beneficial to the process. In studying how ontology users work in performing local adaptations, they note that these local adaptations almost exclusively deal with changes to the subsumption hierarchy. No new relations were defined, and only a little instance assignment. They conclude that users do not understand the logic structure or theory of an ontology: “*First of all our users did understand the ontology mainly as a classification hierarchy or their documents*” [53, p. 315]. They also note that reconciling the local adaptations when attempting to build a new version of the core is a task that needs to be performed by a knowledge engineer, and which cannot be automated.

2.3.4 Ontology Development 101

The Ontology Development 101 guide by Noy and McGuinness [54] does not present an ontology development methodology as such, but it is often referred to and recommended as a good introduction to ontology engineering for beginners, and it does provide a structured overview of required tasks in an ontology engineering project. Updated to correspond to the OWL terminology used in this thesis those required tasks are: *Scoping*, *Reuse*, *Term enumeration*, *Class hierarchy construction*, *Property elicitation*, *Property definition*, and *Instance creation*. Each step is explained and exemplified. Additionally, a section of the document discusses commonly occurring problems or issues associated with each step, to help the reader avoid the most common pitfalls.

The Ontology Development 101 guide is aimed at ontology users, domain experts, and students. It does not take any prior knowledge of ontology theory or practice for granted. It also gives very concrete and applicable guidance on practical issues of ontology engineering, such as the difference between subclassing and typing, or the difference between concepts and the labels of said concepts. In this way it fills some pedagogical gaps that the previously discussed methods (giving big-picture guidance aimed at people who are already ontologists) do not cover. However, due to the limitations in size of this guidance document it obviously cannot cover the entire set of easy mistakes and bad or good practices. Additionally, due to the limitation in scope (it is a general guideline, not a domain-specific one) difficult modelling issues which are usage area-specific are not covered. Finally, the guide is written to apply to the development of pre-Semantic Web frame-based ontologies in systems like Protégé 2000, and it consequently does not make use of the features (imports, resolvable URI references, namespaces, etc.) that more recent ontology technology enables.

2.4 Ontology Design Patterns

As illustrated by the methods mentioned above, it is still the case that ontology development for the Semantic Web is mostly carried out by ontology engineers and description logic experts, and the majority of tools and methods for this purpose are geared towards this category of developers, and not domain experts. This state of affairs is problematic for two reasons. Firstly, the additional knowledge elicitation and acquirement steps required when the roles of domain expert and ontology engineer are separate slows down the ontology development process in the individual case by requiring additional tasks to be performed. Secondly, ontology engineers are still most commonly academics and researchers, and the industrial uptake of semantic technologies is not as high as it could be. A higher degree of knowledge regarding these technologies and ontology engineering among non-academics and domain experts could go a long way toward furthering adoption of semantic technologies among practitioners.

Further, as the methods discussed also illustrate, established guidance and methods in ontology engineering has focused on the big picture, i.e., which overall phases or large granularity tasks that need to be performed in an ontology engineering task. With the exception of the Ontology Development 101 guide, none of the discussed methods go down to the level of detail of how to solve more concrete commonly occurring tricky modelling issues. While several methods mention reuse of existing ontologies, none give specific guidance regarding how such reuse is best achieved.

Ontology Design Patterns were introduced as potential solutions to these types of issues at around the same time independently by Gangemi [7] and Blomqvist and Sandkuhl [6]. The former define such patterns by way of a number of characteristics that they display, including examples such as “an

ODP is a template to represent, and possibly solve, a modelling problem” [7, p. 267] and “[an *ODP*] can/should be used to describe a ‘best practice’ of modelling” [7, p. 268]. The latter describes ODPs as generic descriptions of recurring constructs in ontologies, which can be used to construct components or modules of an ontology. Both approaches emphasise that patterns, in order to be easily reusable, need to include not only textual descriptions of the modelling issue or best practice, but also some formal ontology language encoding of the proposed solution. The documentation portion of the pattern should be structured and contain those fields or slots that are required for finding and using the pattern. For an example of what an Ontology Design Pattern description can look like, see Figure 2.6.

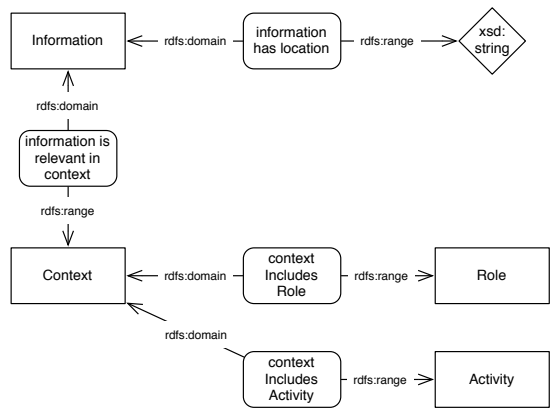
Gangemi [7] motivates the need for ODPs by noting how useful similar constructs have been in practical cases where domain experts were involved, both in terms of simplifying knowledge acquisition from these experts, and in terms of enabling said experts to perform basic ontology engineering themselves. Blomqvist and Sandkuhl [6] on the other hand motivate the need for patterns by referring to the potential for reuse that these bring, particularly in automatic or semi-automatic ontology engineering scenarios. In subsequent work [8, 55, 56] these two sets of motivations for Ontology Design Patterns have come together, such that at the time of writing, patterns are motivated by perceived gains with regards to both reusability and guidance. Additionally, as pointed out by Blomqvist in [55] communication benefits can also be achieved by such patterns, in that having a shared vocabulary of commonly occurring modelling problems and associated solutions can help simplify ontology engineering in a team environment.

It is important to note that the issue of pattern quality is strongly connected to the motivations of pattern use, which can, as shown above, vary. If one considers reuse to be the main and only motivation for the development and use of ODPs, then the associated ontology language encoding is likely to be the main object of study, whereas if one considers guidance and communication to be more important motivations, then the pattern documentation may be of greater importance. From a philosophical perspective it can be argued that both of these parts of the pattern are actually just different representations of an abstract phenomenon, the coupling of a problem and solution, which exists as a purely mental construction or idea. In this thesis the philosophical debate is sidestepped, and Ontology Design Patterns are taken to consist of both a documentation portion and a reusable code module portion, and are taken to be intended for supporting reusability, guidance, and communication.

Since their introduction, Ontology Design Patterns have been the subject of some research and work, see for instance the deliverables of the EU FP6 NeOn Project¹¹ [8, 57] and the work presented at instances of the Workshop on Ontology Patterns¹² [58, 59, 60] at the International Semantic Web

¹¹<http://www.neon-project.org/>

¹²<http://ontologydesignpatterns.org/wiki/WOP:Main>



Name	Context dependant information
Intent	To model the case when some information is deemed especially relevant for a particular role performing a particular action.
Competency questions	What information is available that in some way deals with task X? What documents are available that are relevant only for an Astronomer (role) doing task Y? I am a PhD Student (role). What documents are there that I could be interested in, of any topic?
Solution description	One or more roles are assigned to a person. The activities that are performed in the target domain are modelled as Activity instances. Both Role and Activity can be subclassed depending on one's needs. Roles and Activities are joined by context, for instance "Doctor doing diagnosis" or "Medically unskilled person doing diagnostics". What Information instance is deemed relevant for each context is decided by way of the "informationIsRelevantInContext" property.
Reusable OWL building block	http://www.infoeng.se/~karl/images/ift5/Context_Dependant_Information.owl
Consequences	No known consequences.
Scenarios	Medical doctors using different diagnostics manuals than non-medically trained people when diagnosing illnesses. Car mechanics using different guidelines when servicing exhaust systems than brake pedals.

Figure 2.6: Context Dependant Information ODP

Conference. There are to the author's best knowledge no studies indicating ontology engineering performance improvements in terms of time required when using patterns, but results so far indicate that their usage can help lower the number of modelling errors and inconsistencies in ontologies, and that they are perceived as useful and helpful by non-expert users [9, 61].

2.4.1 ODP Typologies

The use and understanding of Ontology Design Patterns has been heavily influenced by the work taking place in the NeOn Project, the results of which include a pattern typology [8] shown in Figure 2.7¹³, and the eXtreme Design collaborative ontology development methods, based on pattern use [57]. The typology of patterns has been developed further within the Ontology-DesignPatterns.org initiative and is used as a classification schema for this initiative's pattern repository. The typology is based on the uses to which patterns are put, whether they represent best practice in reasoning, naming, transformation, content modelling, etc. Certain categories of patterns are in this view subcategories of one or several other categories. While the work in this thesis concerns only Content ODPs, all of the top-level pattern categories from the NeOn typology are for completeness presented below with brief descriptions of their purpose and structure.

- **Content ODP** – Content ODPs solve modelling issues regarding ontology content, either in the general domain or in one specific domain of study. They provide solutions to problems that are known to be difficult to model correctly, or problems which are known to occur frequently and for which a conceptual harmonisation can be of use.
- **Structural ODP** – Structural ODPs are patterns that concern either design problems regarding ontology language insufficiencies and limitations or the overall structure and shape of an ontology. The former class of patterns are known as Logical ODPs, and include for example the *nary relation* ODP, which suggests a reification solution to the problem that many ontology languages only support binary relations. The latter class provide suggestions for the structure of an ontology as a whole, and include examples such as *Taxonomy* or *Modular Architecture*.
- **Correspondence ODP** – Correspondence ODPs deal with issues of reengineering and alignment of ontologies. Patterns that deal with the former consist of sets of transformation rules that can be applied to change an existing model (either an ontology or a non-ontological resource) into a new ontology. Patterns that deal with the latter are

¹³The figure shown here is based on an updated version of the typology as published on the ODP community portal, <http://ontologydesignpatterns.org>, in which Mapping patterns have been renamed Alignment patterns.

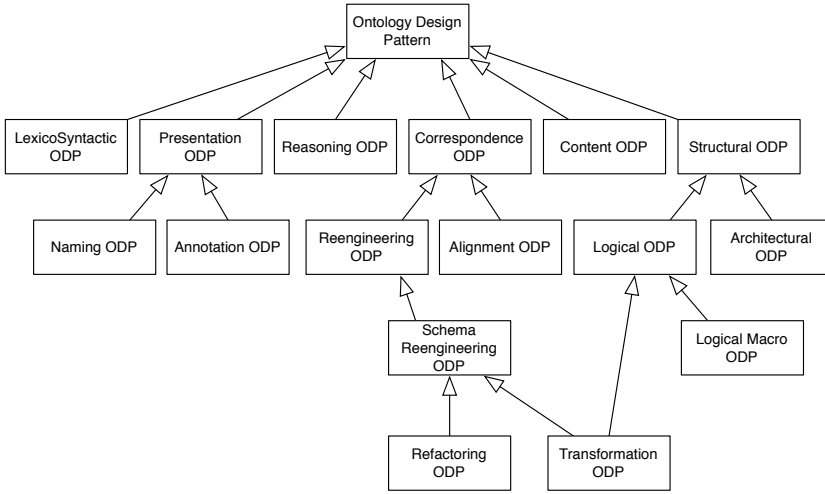


Figure 2.7: NeOn ODP Typology [8]

written as a set of semantic relations between classes and individuals in two different ontologies, in order to provide interoperability without discarding the existing models.

- **Reasoning ODP** – Reasoning ODPs model particular tasks that a reasoning engine could perform (such as *subsumption hierarchy materialisation* or *restriction de-anonymising*). The idea behind this type of pattern is that it can be useful in ontology normalisation and standardisation. At the time of writing no reasoning patterns fitting the above definition have been published, however, [62] presents a number of task-based patterns for the Semantic Web employing reasoning and ontologies, for instance *Service selection* and *Semantic enrichment*.
- **Presentation ODP** – Presentation ODPs are recommendations and best practices on how to name, annotate, graphically illustrate, and otherwise document ontologies in a way that promotes their learnability and usability.
- **Lexico-Syntactic ODP** – Lexico-Syntactic ODPs are mappings of language structures to ontology structures, intended to simplify Ontology Learning tasks. Examples include traditional Hearst patterns [63] mapped to ontology constructs.

While the NeOn view is influential and its accompanying typology is referenced frequently, it is neither universally accepted, nor the only perspective on the issue – for instance, Blomqvist [55] presents a different typology based on the level of abstraction and granularity of the reusable solution.

According to this categorisation structure, shown in Figure 2.8, there are four levels of ontology pattern abstraction that restrict the scope of the pattern and the granularity of the constructs that it concerns: *Application* patterns, *Architecture* patterns, *Design* patterns, and *Syntactic* patterns.

Application patterns concern the overall scope and purpose of an ontology with an application context, and describes how an ontology works together with executable code to provide some set of functionalities. A pattern on this level treats the ontology as a unit or a component in a software system, but does not break the ontology apart further. *Architecture* patterns do break apart the ontology further, and concern the internal structure of the ontology and the modules that make it up. Patterns on this level may include restrictions on design patterns or modules used in the ontology. However they do not deal with specific low level modelling issues. Those types of issues are instead dealt with in *Design* patterns (obviously, Blomqvist [55] does here not use the term Ontology Design Pattern synonymously with its use in the NeOn typology and in the rest of this thesis). Design patterns do deal with how to handle specific modelling challenges concerning the logical structure of difficult-to-model content. Design patterns work on the level of logical axioms and constructs, but do not delve into syntactical or ontology language-specific issues. This last category of problems is the domain of *Syntactic* patterns, which deal with the actual serialised on-disk representation of an ontology, i.e., string and character combinations.

2.4.2 ODP-based Ontology Construction

eXtreme Design (XD) is defined as “*a family of methods and associated tools, based on the application, exploitation, and definition of Ontology Design Patterns (ODPs) for solving ontology development issues*” [64, p. 83]. The method is influenced by the eXtreme Programming (XP) agile software development method, and like it, emphasises incremental development and continuous requirements management (as opposed to the more traditional method of separating requirements engineering and development phases). Like XP it also recommends pair development, test driven development, refactoring, and a divide-and-conquer approach to problem-solving [65].

Conceptually, XD describes approaches for selecting patterns for reuse based on the notions of problem space and solution space. The problem space consists of the set of modelling problems (Local Use Case, LUC) that an ontology engineer comes across in the course of a particular project. The solution space is the set of reusable solutions to common problems, i.e., patterns. Included in each pattern is a description of the Generic Use Case (GUC) in which it is applicable. By mapping LUC to GUC, the ontology engineer finds appropriate patterns that solve the modelling problems that occur in their problem space, as indicated in Figure 2.9 [64, 65].

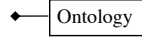
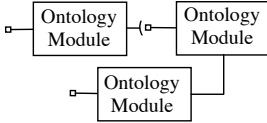
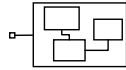

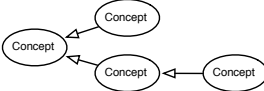

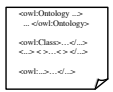
Abstraction	Granularity	Illustration of level
Application pattern	Complete ontology	 <p>Ontology requirements and interface, possibly described in a software architecture description language.</p>
Architecture pattern	Complete ontology	 <p>Overall ontology organisation and parts, possibly described in an ontology architecture description language.</p>
	Part of the ontology	<p>Ontology Module</p>  <p>An ontology part, e.g. module, and its overall organisation, possibly described in an ontology architecture description language.</p>
Design pattern	Complete ontology	 <p>Restrictions on the modelling of the overall ontology, described through an ontology modelling language.</p>
	Part of the ontology	 <p>An ontology part solving a specific modelling problem, described through an ontology modelling language.</p>
	Elements of the ontology	 <p>Individual ontology element, described through an ontology modelling language.</p>
Syntactic pattern	Complete ontology	 <p>Pattern guiding the representation of a complete ontology in an ontology representation language.</p>
	Part of the ontology	<pre><owl:Class> ... <owl:Class> <owl:Class>...</owl:Class> <owl:ObjectProperty> </owl:Class></pre> <p>Representation of parts of an ontology in an ontology representation language.</p>
	Elements of the ontology	<pre><owl:Class rdf:about=""> <rdflib:ClassOf rdfrsource=""/> </owl:Class></pre> <p>Representation of individual element in an ontology representation language.</p>
	Element representation	<pre><owl:Class rdf:about=""> </owl:Class></pre> <p>Primitive patterns of the representation language itself.</p>

Figure 2.8: Blomqvist's ODP Typology [55]

The proposed selection method is appropriate for finding patterns that satisfactorily solve a particular problem from a larger set of patterns. It may also be possible to formalise and automate, provided that an appropriate logical vocabulary for describing LUCs and GUCs is developed. It does however not guide the user in selecting, from a given set of functionally appropriate patterns, the one that is best suited for use in their particular

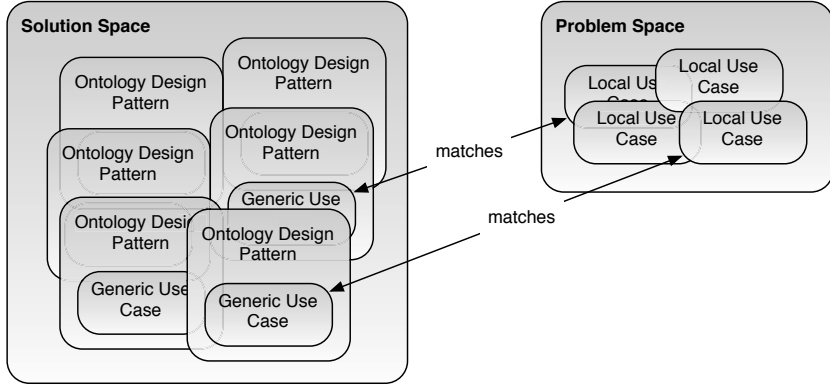


Figure 2.9: XD Pattern Selection Approach [64]

situation. The right choice then could depend on non-functional requirements on the ontology as a whole (expandability, performance, testability, etc.), or it could depend on quality attributes of the pattern itself (how easy is it to apply, how well is it documented, is there an example ontology using it, etc.). An ODP quality model could in this scenario guide the developer in selecting patterns to use that, apart from solving the functional requirements of their modelling problem, also has features and qualities that are appropriate and helpful to them.

The XD method consists of a number of tasks, as illustrated in Figure 2.10. The first three tasks deal with establishing a project context (i.e., introducing initial terminology and obtaining an overview of the problem), identifying a set of candidate ODP portals on the Web, and collecting initial requirements in the form of a prioritised list of user stories (describing the required functionality in layman's terms). These steps are performed by the whole XD team together with the customer, who is familiar with the domain and has an understanding of the required functionalities of the resulting ontology. The later steps of the process are performed in pairs of two developers (these steps are in the figure enclosed in a grey box). They begin by selecting the top prioritised user story that has not yet been handled, and transform that story into a set of competency questions, contextual statements, and reasoning requirements. Competency questions (introduced in Section 2.3) can be understood as example questions that the resulting ontology should be able to answer, and may be written in natural language, or in a more formal notation such as the SPARQL query language. Example competency questions are included in Figure 2.6. Contextual statements are general axioms that should hold within the modelled domain. Reasoning requirements are such requirements regarding reasoning capability of the resulting ontology and/or system that are difficult to express in competency

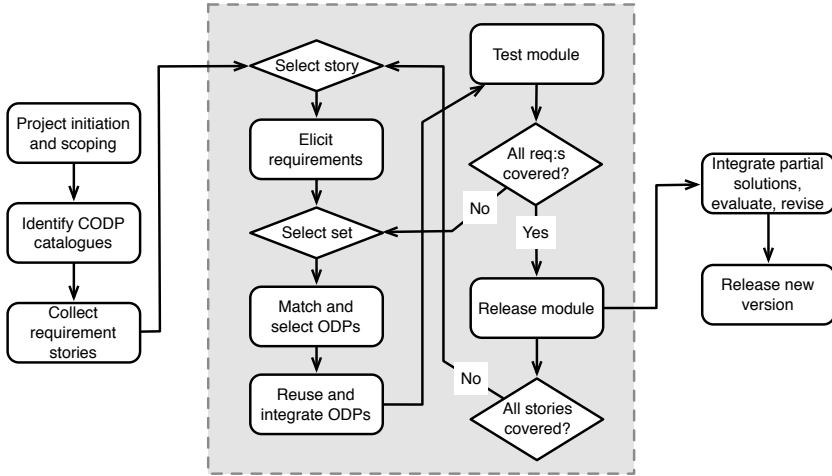


Figure 2.10: XD Workflow (adapted from [64])

question form. Customer involvement at this stage is required to ensure that the user story has been properly understood and that the elicited competency questions, contextual statements, and reasoning requirements are correctly understood. The development pair then selects one or a small set of interdependent competency questions for modelling [64, 65].

In the development process, a pattern matching the competency questions is selected by matching LUC to GUC as described earlier. There may be multiple matching ODPs found, in which case the development pair must select one based on their understanding of the problem domain and the modelling consequences associated with each matched pattern. The selected pattern is then adapted and integrated into the ontology module under development (or, if this iteration covers the first requirements associated with a given user story, a new module is created from it). The module is tested against the selected requirements, to ensure that it covers them properly. If that is the case, then the next set of requirements from the same user story is selected, a pattern found, adapted, and integrated, and so on. Once all requirements associated within one user story have been handled, the module is released by the pair, and integrated with the ontology developed by the other pairs in the development team. The integration may be performed either by the development pair themselves, or by a specifically designated integration pair [64, 65].

The XD method is supported by the XD Plugin for the Eclipse-based NeOn Toolkit¹⁴, an ontology engineering development environment produced within the NeOn project. The XD Plugin provides a number of

¹⁴<http://neon-toolkit.org/>

components to this environment that simplifies pattern browsing, selection, adaptation, and use. The browsing and selection components make use of the `ontologydesignpatterns.org` pattern repository, and allows users to browse the patterns in this repository, or search over them by competency questions. The adaptation component contains a number of wizards that allow for easier specialisation of generic patterns for use in specific projects. The assistant component provides warnings and suggestions to modellers based on known good or bad practices (i.e., patterns or anti-patterns). While the eXtreme Design method can be executed using any ontology engineering environment, the use of tooling such as the XD plugin can help streamline the process.

Related deliverables of the NeOn project also prescribe ways of finding or developing patterns [8], including reengineering from other data models, specialisation/composition of existing patterns, extraction from reference ontologies, and a method consisting of sequentially combining extraction, specialisation, generalisation and expansion. However, the proposed methods of pattern generation/extraction are not described in any detail, and they in particular leave out a discussion on the wanted or beneficial attributes of patterns (while a set of pragmatic ODP features is presented in [56], these features are quite general and do not provide measurable design criteria). The choices of what type of pattern to create, what characteristics to emphasise, how to compose and document the pattern, and so on, when employing these methods are left to the pattern developer. In such ODP development work, an ODP quality model would be useful in guiding the developer in producing high-quality patterns.

2.5 The State of ODP Research

This section details a systematic literature survey over ODP research performed at the start of this thesis project, the results of which motivated the choice of thesis project topic. The survey had as stated goals to answer the following questions:

1. What kind of research on ontology patterns is being performed?
2. How has research in the field developed over time?
3. Where is ontology pattern research performed?
4. How is research in ontology patterns being done?

For the purpose of this survey, the author studied conference and workshop proceedings of ISWC, ASWC and ESWC from 2005 to 2009. The proceedings were retrieved from various Internet databases including Springer-Link¹⁵ and CEUR-WS¹⁶. A total of 2462 papers were retrieved, divided

¹⁵<http://www.springerlink.com>

¹⁶<http://ceur-ws.org/>

Table 2.1: ODP papers by year

Year	Conferences	Workshops
2005	2	2
2006	2	2
2007	7	3
2008	1	4
2009	4	20 ¹

¹ 15 of which are from the Workshop on Ontology Patterns.

Table 2.2: Institutes by paper count

Institute name	Conference	Workshop
ISTC-CNR	4	2
University of Economics, Prague	2	4
Universidad Politecnica de Madrid	1	5
University of Karlsruhe	0	4
Jönköping U.	3	0
Salzburg Research GmbH	2	1
University of Innsbruck	1	2
University of Manchester	1	2
University of Sheffield	1	2

into 861 conference papers and 1601 workshop papers. After performing a full-text keyword search over this dataset, and removing false positives, 47 papers concerning Ontology Design Patterns were located; 16 conference papers and 31 workshop papers. These 47 papers were classified by a number of different dimensions, including publication metadata, pattern usage, research validation method, and the importance of ODPs in the papers.

The research presented here has previously been published in [10], which the reader is referred to for fuller details. Method and methodology issues associated with the literature survey are also presented in Section 4.2.1.

2.5.1 Results

The processes described above provided a large amount of data to analyse, a subset of which is presented in this chapter. The complete dataset is too large to include in full, but is available for download¹⁷.

Table 2.1 presents the number of papers in the dataset indexed by the year they were published, the idea being that this might give an indication as to whether research interest in the field is expanding. Table 2.2 lists the research organisations most often listed as affiliations of authors in the dataset (sorted by summarised publication count and alphabetically). The full list is considerably longer at 49 entries in total, but is for brevity here limited to organisations with three or more mentions.

Table 2.3 contains the results of the content classification process, that is, the labels denoting categories of pattern-related research and the number

¹⁷<http://purl.karlhammar.com/data/phl/wop2010/>

Table 2.3: Classification of the reviewed papers’ connection to ODPs.

Classification	Conferences	Workshops
Anti-patterns	0	2
Evaluation	0	2
New pattern presented	3	12
Pattern creation methods	1	1
Pattern features	1	5
Pattern identification	0	2
Pattern languages	1	4
Pattern usage method	4	11
Pattern typology	0	3
Patterns used	8	6

Table 2.4: Validation levels of reviewed papers.

Source	No validation	Anecdote	Example	Empiricism
Conferences	3	2	5	6
Workshops	3	2	19	7

of papers tagged with each such label. The results are divided into columns for conference papers and workshop papers.

Tables 2.4 and 2.5 present the results of the validation classification, i.e., how the findings presented in the papers were validated and, in the case of empirical procedures being used for this purpose, how well the experiments or case studies were described. Metrics from [66] are reused, according to which empirical validations are graded based on how thoroughly the context of the validation, the study design itself, and potential issues of validity or generalisability are described. Table 2.6 presents the institution counts of the papers in the dataset. Table 2.7, finally, shows the result of the ODP importance classification, i.e., in what parts of the papers that the topic of patterns were addressed.

2.5.2 Analysis and Discussion

The section below analyses the data resulting from the survey, attempting to answer the questions set forth at the beginning of Section 2.5.

Table 2.5: Quality of empirical validations.

Quality indicator	Weak	Medium	Strong
<i>Conference papers</i>			
Context description	4	1	1
Study design description	0	3	3
Validity description	5	1	0
<i>Workshop papers</i>			
Context description	5	2	0
Study design description	1	3	3
Validity description	6	1	0

Table 2.6: Institution counts

Institutions	Conferences	Workshops
1	12	16
2	2	10
3	2	5

Table 2.7: ODP importance classification of reviewed papers.

Group	Conferences	Workshops	Workshops (w/o WOP)
Title match	7	22	8
Abstract match	3	3	2
Body match	6	6	6

What kind of research on ontology patterns is being performed?

The results indicate that while patterns are used in various different ways in research and new patterns are being presented, there is much less work being done on how to formalise the creation and/or isolation of patterns. This could indicate one of two things. It could indicate that the best ways of creating and finding patterns have been established, and that there is thus little need for more research to be done in those areas. However, due to the youth of the field, this is believed to be less likely.

Instead, it is more likely that the results indicate a lack of sufficient research in these areas. This situation could be problematic if it indicates that the patterns that are used are not firmly grounded in theory or practice. If one adds to this the fact that relatively little work is being done on pattern evaluation, the overall impression is that patterns are being presented and used as tools, but are not being sufficiently studied as artefacts of their own.

Another area that appears to be understudied is anti-patterns, “worst practices” or common mistakes. This may be because finding such anti-patterns necessitates empirical study, which as reported below appears to be less common in the papers studied.

The distribution of research categories seems to be rather consistent between the set of conference papers and the set of workshop papers, with exception for the categories *Patterns used* and *Pattern creation methods*. The latter could be a simple statistical anomaly (as mentioned, there are few papers dealing with this topic, for which reason small discrepancies stand out more). The former seems a bit more notable however, but no explanation for this observation has yet been found.

How has research in the field developed over time?

There is a much larger number of papers matching the search criteria published in 2009 than in 2005, in large part due to the first Workshop on Ontology Patterns being held in 2009. If the WOP papers are removed, there is still a growth in volume.

One interesting note in relation to the previous discussed research question is that all of the work on pattern identification and pattern creation methods that was found was published in 2009. Furthermore, the papers dealing with pattern evaluation are all from 2008-2009, possibly indicating that researchers have noticed these gaps in theory and are attempting to do something about them.

Unfortunately both of these observations are based on so few papers and such a short period of time that it would be very difficult to claim them as a general trends or make predictions based on them.

Where is ontology pattern research performed?

The results indicate that ODP work is primarily taking place at European institutions. Table 2.2 shows that all of the top nine institutions publishing three or more papers are located in mainland Europe and the UK. However, even if one includes all of the institutions that have two publications in the dataset, one still does not find any non-european organisations. As a matter of fact, out of a total of 49 institutions that had published, only four were located outside of Europe. Out of these four, three were based in the USA and one in New Zealand.

While the dataset is dominated by research originating at universities, there are also a number of private corporations, research foundations, and other types of non-university organisations that work in the field. Out of the 49 institutions found, 17 were such non-university organisations (approximately 35 %). These proportions are slightly lower when taking into account the number of publications per institution, with the non-university organisations netting 31 % of all institutional mentions in the dataset.

How is research in ontology patterns being done?

Studying academic cooperation, one easily accessible metric is the number of authors per paper. However, author counts on their own can be misleading. In some academic cultures students' advisors get authorship, while in others they do not. Instead looking at both author counts and institution counts gives a better indication of actual research cooperation.

Out of a total of 47 papers, 19 (just over 40 %) list more than one affiliated institution and 7 (just under 15 %) list three institutions. These figures, however, include papers written by only one author. Looking only at the subset of 40 papers that were written by more than one author, the figures are 47.5 % and 17.5 % respectively. No matter which way you slice it, these numbers indicate a quite healthy degree of cooperation between research institutions in the field.

Interestingly, there seems to be a difference between work published at workshops and work published at the main conferences - of the former, 51.6 % are credited to single institutions, whereas of the latter, 75 % are. This possibly indicates that the prestige and/or difficulty associated with the

higher barrier of entry to full conferences cause researchers to keep such papers “in-house” to a larger degree.

With regards to the validation and evaluation of ODP research, there may be some work to be done. Nearly one third of the papers published at full conferences contain no validation or only anecdotal validation of the presented work. Another near-third validates the work via examples, but provide no real-world testing to ensure validity. For workshop papers, a smaller proportion of the papers have no validation or anecdotal validation, but on the other hand, a much larger proportion validate only through example. Of the papers that do contain empirical testing, it is uncommon to see discussions on the limits of validity of said testing. This situation may be somewhat problematic. Though not all types of research invite the opportunity to perform experiments or case studies, nor actually require them, quite a few papers in the dataset could have benefitted from a more thorough testing procedure.

Finally, looking at how central ODPs are to the content of the papers, the most common situation is actually that patterns are mentioned already in the title (see Table 2.7), indicating that they are quite central to the papers. This remains the case even when filtering out the WOP papers which naturally skew the results. One possibility is that this indicates that Ontology Design Patterns are primarily used within the ODP research community and thus written about by people who consider them to be important enough to warrant inclusion in the title.

Chapter 3

Evaluation and Quality Frameworks

While this thesis represents a first attempt at understanding the quality of Ontology Design Patterns, it is by no means unique in dealing with the development of quality models for IT artefacts. There are many developed frameworks and models for evaluating software and conceptual models. Given this existing work in related fields, starting from an entirely clean slate when approaching the issue of ODP quality would be unwise. The model developed in this thesis instead builds upon existing work in neighbouring fields, work that is summarised in this chapter's first two sections. The last section of the chapter details some of the author's earlier work in surveying the state of ODP research, which initially motivated this thesis project.

3.1 Related Quality Frameworks

The following section presents and discusses existing approaches for evaluating the quality of models, systems, and patterns, which are suitable for reuse or consideration in the development of a quality model for Ontology Design Patterns.

3.1.1 MAPPER

The MAPPER validation framework [67, 68] was developed within the MAPPER project, the overall goal of which was to develop model-based approaches to improving product and process engineering. The author's proximity to this project enabled access to project deliverables, including work on the validation framework, which proved both influential and useful. Within the MAPPER project, the validation framework served to help harmonise communication regarding and evaluation of the different project artefacts, be they objectives, processes, conceptual models, or other types of deliverables.

The framework is consequently rather complex, as it aims to cover a large number of use cases and tasks related to evaluation. For this reason, while key perspectives of this framework as presented below have influenced the development of the ODP quality metamodel (presented in Section 5.1), the entirety of the validation framework or its metamodel has not been adopted outright, nor is it exhaustively covered here.

The MAPPER validation framework [67, 68] is represented as a visual model in which different quality-related concepts are linked by relationships. The typing of the concepts and the relations allowed between them adheres to the MAPPER validation framework metamodel (illustrated in Figure 3.1). In evaluating a particular objective, criterion, hypothesis, etc., this metamodel is instantiated and each metamodel concept “filled” with one or more quality-related concepts. In the metamodel validation aspects are based on validation criteria, which in turn are associated with measurement methods. Per this perspective there is a differentiation between those more general and not directly measurable quality aspects (exemplified by “Resource use”) and the more tangible and directly measurable quality criteria (exemplified by “Average POI length”). In following measurement methods for these criteria (that is, in performing evaluations) certain actions are performed, actions that are affected by case context, and give rise to results. The more general quality aspects can be refined in such a manner that a hierarchy of quality aspects can be established. The metamodel also indicates that development objectives refer to criteria, that is, that validation of artefacts cannot be seen as independent of intended artefact usage objectives.

The most relevant perspectives from the MAPPER validation framework [67, 68] metamodel that may apply also to an ODP quality metamodel concern the differentiation between measurable criteria and immeasurable general quality aspects, the possibility of nesting of different quality aspects into a hierarchy, the idea that the metamodel is instantiated into a “filled out” quality model when applied, and the importance of representing use case and context in modelling artefact quality. These metamodel perspectives are, as shown in Sections 3.1.2 and 3.1.4, compatible with other quality models including the Thörn quality model for variability models [69] and ISO 25010 [70], which have supported this thesis.

3.1.2 Conceptual Model Quality

Ontologies are essentially models of the world, or at least, a domain of discourse. Ontology design patterns can therefore be seen as small reusable models of a particular recurring concept or set of concepts. It is therefore reasonable to assume that existing conceptual model quality research is to some degree transferrable to the field of Ontology Design Patterns.

The PhD thesis *On the Quality of Feature Models* [69] by Christer Thörn deals extensively with how to study and ascertain the quality of conceptual

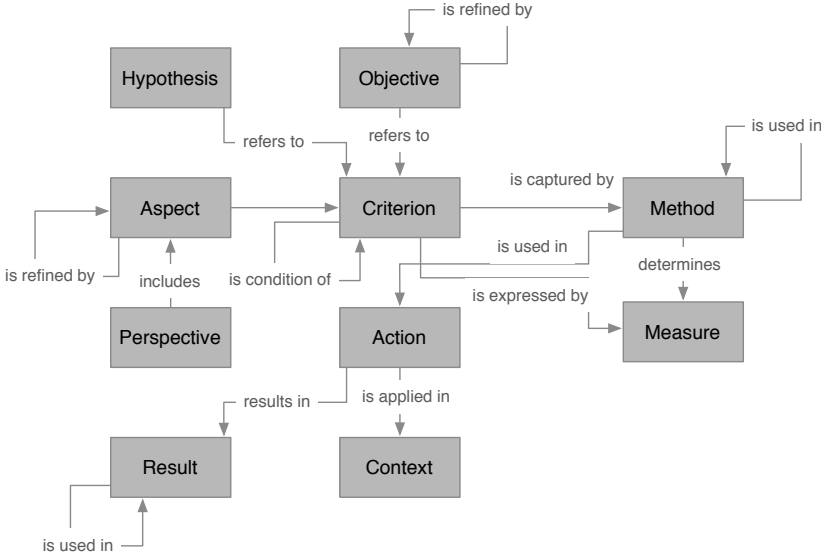


Figure 3.1: MAPPER validation framework metamodel [67, 68]

models. The artefacts studied in his thesis are feature models (or variability models), that is, models for displaying dependencies and requirements between different component parts or subsystems in a product, commonly a software system. Such models are different from ontologies and Ontology Design Patterns in certain respects, but similar to them in others. While feature models do employ certain language semantics, these are relatively simplistic and specific to feature modelling [71, 72]. The models are primarily intended for supporting requirements engineering and design work, as opposed to structuring large amounts of data or inferring knowledge using reasoning engines. Like ontologies however, feature models are artefacts that are used collaboratively in performing engineering tasks. Like ontologies, they can grow very large and become difficult to overview, and like ontologies they do employ a certain semantics, expressing a real world problem or case according to those semantics. Feature models have been used for variability modelling in industry since many years, by users of varying technical background and skills [73, 74, 75]. Ontology Design Patterns are intended to support users of varying technical background and skills in conceptual modelling, a not entirely dissimilar task.

In [69] Thörn develops and presents a quality model containing six quality factors believed to be most relevant for feature models¹:

¹For brevity, the shortened definitions from [69] are shown.

- *“Changeability – Ability to evolve the model while maintaining the uses of previous versions.*
- *Reusability – Ability to reuse (parts of) the model when evolving or developing other models.*
- *Formalness – Ability to manage the model in a formalised manner, e.g. for machine management.*
- *Mobility – Ability to be moved, transferred and integrated with other systems.*
- *Correctness – Correspondence (mapping) between the model and the modelled artefacts.*
- *Usability – User-friendliness and ease of learning and communication to new users.” [69, p. 152]*

These quality factors were selected from an initially larger set of factors and attributes based on evaluation in a thorough case study observing which qualities practitioners prioritised in real world modelling cases. Each of the quality factors is associated with a textual definition and a set of indicators observed to affect the quality factor. While the quality factors are more general in nature and possibly applicable to ODPs as well, the indicators in this model are specific to feature models and unsuitable for reuse in an ODP context.

Thörn’s quality model [69] does not define a generally applicable prioritisation of quality factors, nor a method for establishing such prioritisation based on established context/case types – instead, in each feature model development case, stakeholders are required to select which quality factors are deemed most important in the given context, based on a method of pair comparison. Several developmental principles are presented, intended to guide feature model developers in constructing models adhering to the defined prioritisation of quality factors.

3.1.3 Entity Relationship Model Quality

Entity Relationship (often abbreviated ER) models are a type of conceptual models specifically used to model relational (i.e., tabular) datasets and the relations between such datasets. While many types of conceptual models (including to a certain degree the variability models discussed in the previous section) are intended as visualisations and communicative artefacts, ER models are formalised to the degree that they can be used to generate database schemas. These models are in broad use in both industry and academia, and are clearly found useful by practitioners. Consequently, developed indicators for measuring ER model usage are likely to be well grounded in empirical evidence, and therefore extra relevant for study and possible inclusion in an ODP quality model.

In [76] Genero et al. study a set of metrics believed to affect the learnability and modifiability of ER models. This study was performed in a small-scale experimental setting with 40 participant subjects. The participants were given a set of ER models that differed in the metrics being studied, and were tasked with first filling out a questionnaire to evaluate their understanding of the ER model, and secondly, to modify the ER model in accordance with a newly introduced set of additional requirements. In analysis, the incorrect responses and modifications were discarded, and the time taken to respond to the questionnaire and to perform the model changes was instead studied to ascertain the relative understandability and modifiability of the ER models that exhibited different values for the metrics of study. Genero et al. find a significant correlation between high values for certain studied metrics and an increased understandability and learnability time. While the metrics studied are specific to ER models, the approach in evaluating understandability and modifiability is clearly applicable also outside of this domain.

Moody and Shanks [77] discuss the issue of redundancy and simplicity, arguing that a simpler model is proven to be more flexible, easier to implement, and easier to understand. They suggest that if the size of a model is calculated from the number of entities and relationships in the model, the simplest solution is the one that minimises this size. However, it is important to note that the model must still be suitable for its intended purpose. This mirrors the perspectives brought forward by Lindland et al. in [78]: a high-quality model is constrained by both completeness and relevance criteria, such that it should be as small as possible, yet not so small as to not fulfil its purpose.

Moody and Shanks [77] also suggest that usability-related qualities be ascertained by user evaluation and rating. While not detailed by the authors, such a rating could in many cases be performed via questionnaire surveys or interviews. Moody and Shanks emphasise the importance of capturing opinions of several different stakeholders, with different perspectives. They suggest that these perspectives are captured via rating by three categories of users, depending on roles: *business user rating*, *data administrator rating*, and *application developer rating*. The first category of user can verify that the data model is consistent with business requirements; the second category can verify that the model is compatible with and can be integrated with existing data models in the enterprise; and the third category must be able to verify that the data model can be implemented in system development.

In summary, while much of the work on quality metrics and indicators in the ER field is specific to the features and structure of Entity Relationship models, certain methods for evaluating practical work using these artefacts (e.g. survey and interview methods for measuring usability, or measuring the time required for performing work), and certain general quality indicators (e.g., size, completeness) are likely to be suitable for reuse in studying Ontology Design Pattern quality also.

3.1.4 Information System Quality

Ontologies are almost always used as components within an information system. As such, research on software artefact quality is likely to be useful and relevant in understanding the demands on an ontology from an information system perspective. This field has seen considerable work going back to the 1970s [79, 80, 70]. While some of this work deals with technical measures and metrics that is only relevant to executable code (branching points, function lengths, and so forth), there are also quality frameworks that include more general aspects of quality in software systems. In particular, the ISO standard 25010 [70] introduces quality models for software artefacts that are reused in the latter portions of this thesis.

ISO 25010 defines two quality models supporting different uses and artefacts in a software engineering context, the *Quality In Use Model*, and the *Product Quality Model*. The former model defines quality in terms of outcomes of interaction with a complete information system by humans, organisations, or other information systems. The latter model defines quality in terms of characteristics of a particular software product or computer system that includes a certain software. The two quality models are expressed according to a framework that defines certain basic concepts, which is also used in the related standards in the SQuaRE series (ISO 25000 through 25099). These definitions include among other things [70]:

- Software Quality Characteristic – Category of software quality attributes that bears on software quality. Software quality characteristics can be refined into multiple levels of sub-characteristics and finally into software quality attributes.
- Quality Measure Element – Measure defined in terms of an attribute and the measurement method for quantifying it, including optionally the transformation by a mathematical function.
- Quality Measure – Measure that is defined as a measurement function of two or more values of quality measure elements.

Per this view, the concepts of quality characteristic and quality measure are disjoint, though linked by quality attributes. Measures are as the name implies measurable and quantifiable. Quality characteristics are more abstract and general.

The Quality in Use model is displayed in Table 3.1. It consists of three top level quality characteristics and eleven quality sub-characteristics. Each of these top level and sub-characteristics are associated with textual descriptions, written from a general usage perspective. For instance, *efficiency* is defined as “*resources expended in relation to the accuracy and completeness with which users achieve goals*” [70, p. 8]. These quality characteristics are affected by and measure factors relating to both the individual software product, the information system in which the product is part, the usage

Table 3.1: ISO 25010 Quality In Use Model (adapted from [70])

Quality characteristic	Subcharacteristic
Effectiveness	Effectiveness
Satisfaction	Usefulness Trust Pleasure Comfort
Context coverage	Context completeness Flexibility
Efficiency	Efficiency
Freedrom from risk	Economic risk mitigation Health and safety risk mitigation Environmental risk mitigation

environment of that information system, and the categories of users that make use of the information system – in short, the Human-Computer System. There is consequently a certain degree of overlap between the quality characteristics defined in this model and the quality characteristics of the Product Quality Model, which focuses exclusively on the effects associated with a software product and the computer system in which that software product executes.

The Product Quality Model [70] is displayed in Table 3.2. It defines a set of eight quality characteristics, each composed of two to six sub-characteristics. These quality characteristics have definitions written from a system or product perspective. For instance, the quality characteristic *availability* is defined as *degree to which a system, product or component is operational and accessible when required for use*. These types of definitions are much narrower and more specific than those used in the Quality In Use model. They are however still not on the level of granularity that they are measurable by some defined metric. It is interesting to note that several of the quality characteristics defined in the Quality In Use model, if reformulated as requirements on a system rather than qualities of that system, would be fulfilled by achieving “high marks” on the Product Quality Model quality characteristics. For instance, high levels of *reliability*, *security*, and *maintainability* (all Product Quality Model characteristics) would help achieve *Economic risk mitigation* (a Quality In Use characteristic). While ISO 25010 [70] as mentioned also defines the existence of concrete and quantitatively measurable quality attributes contributing to these quality characteristics, no specific instances of such quality attributes are introduced or formalised.

ISO 25010 [70] is a complete standard for IT software systems, which is widely used in practice. Since, as mentioned, ontologies and ODPs are used in such systems, is quite likely that parts of this standard will be suitable for reuse and adaptation in modelling the quality of Ontology Design Patterns. While the ISO 25010 Quality in Use model may be less suitable

Table 3.2: ISO 25010 Product Quality Model (adapted from [70])

Quality characteristic	Subcharacteristic
Functional suitability	Functional completeness Functional correctness Functional appropriateness
Performance efficiency	Time behaviour Resource utilisation Capacity
Compatibility	Co-existence Interoperability
Usability	Appropriateness recognisability Learnability Operability User error protection User interface aesthetics Accessibility
Reliability	Maturity Availability Fault tolerance Recoverability
Security	Confidentiality Integrity Non-repudiation Accountability Authenticity
Maintainability	Modularity Reusability Analysability Modifiability Testability
Portability	Adaptability Installability Replaceability

for this purpose (as it concerns interactive information and software systems of a different nature than ODPs, which are by comparison rather passive components), the Product Quality Model holds many quality characteristics that could apply to ODPs also, and is a strong candidate for reuse.

3.1.5 Pattern Quality

Ontology Design Patterns are of a sort of design patterns, that is, packaged solutions to commonly occurring problems. In this, they share purpose (if not domain) with other types of software design patterns, the most common of which are object oriented design patterns. The author therefore initially assumed that some quality indicators associated with such design patterns could be possible to apply also to design patterns in the ontology domain.

The usefulness of object oriented design patterns in software engineering have been shown many times, see for instance [81, 82, 83]. These patterns encode common best practices for how to solve particular types of tricky tasks in the design and programming of software systems, and they have been found to aid in producing flexible, maintainable, and extensible software. However, there are less results relating to the quality of such patterns, how to evaluate them, quality models for them, etc. The below section presents some work in this area.

Prechelt et al. [84] report on two experiments in which the characteristics of design pattern code implementations (in particular, the number of pattern comment lines, PCL, associated with each such implementation) affect the maintainability of software products in which the patterns are used. Their experiments take place in a software engineering setting, with some 96 students as test subjects, tasked with performing software maintenance work. They find that in this context, the more well documented and explicit that design pattern usage in software code is (i.e., the higher the PCL value), the faster and better (in terms of rarity of errors made) maintenance tasks are performed over that software code.

Hsueh et al. develop a quantitative method for evaluating the quality of design patterns, in [85]. In their approach a pattern is considered to consist of intents (i.e., requirements that the pattern aims to fulfil, each intent being classified as either functional or non-functional), of a proposed solution structure that fulfils said intent (again, differentiating between functional and non-functional solution structures), a quality focus indicator (defining the relative importance of functional vs. nonfunctional requirements), and a transformation function that maps a solution structure for functional intents to a solution structure for nonfunctional intents, (i.e., a recommendation on how to make a pattern solution not only fulfil formal functional requirements but more quality-oriented non-functional requirements also). The approach builds on the use of the QMOOD object-oriented design quality assessment model by Bansiya and Davis [86], which is used to structure quality characteristics and indicators. By modelling and mapping intents and solution

structures, Hsueh et al. [85] validate how well the proposed solution lives up to the original pattern intent, both in terms of functional and non-functional requirements.

From a formal viewpoint the method presented by Hsueh et al. [85] is impressive. It is however very complex and therefore likely to be inaccessible to practitioners. Furthermore, the method's dependence on QMOOD means its immediate applicability outside of the field of object-oriented software design patterns is limited (even though the underlying idea for matching of intents to structural solutions may in part be portable to other fields). The results of Prechelt et al. [84] are likely to be of more use, in that they give insights regarding the importance of documenting patterns extensively, that may be relevant to the development of an ODP quality model.

3.2 Ontology Evaluation

One perspective on ODPs is to consider them as being simple, small, and reusable modular ontologies. The following sections introduce work on ontology evaluation frameworks, methods, and indicators, that have been studied during and in several cases influenced development of the ODP quality model. In addition to the ontology evaluation work introduced below, the interested reader is referred to [87], in which Gómez-Peréz et al. summarise and discuss several types of common taxonomical errors and anti-patterns in ontologies.

3.2.1 O^2 and $oQual$

A thorough study on the evaluation of ontologies is performed by Gangemi et al. in [88] and [89]. Their approach is based on two perspectives of ontologies, formalised into two meta-ontologies for understanding, classifying, and selecting ontologies, O^2 and $oQual$.

To begin with, the O^2 meta-ontology views ontologies as semiotic objects, i.e., information objects with intended conceptualisations to be used in particular communication settings. O^2 holds concepts such as *Rational agent*, *Conceptualisation*, *Graph*, and so on – representing the settings in which ontologies are used, the users of said ontologies, their intended meaning of the ontologies, the actual implementation (i.e., graph models), and so on. This model also holds the concept *QOOD*, or *Quality Oriented Ontology Description*, which is intended to capture the roles and tasks associated with processes and elements of the ontology – in essence, a type of requirements specification for part of, or a whole, ontology engineering project.

Based on the concepts in O^2 , three dimensions of quality or evaluation are presented and discussed – *structure*, *functionality*, and *usability*. The first kind of measures treat the ontology as a directed graph (which is consistent with the RDF data model) and concern the structures present in this a graph. This includes measures like subsumption hierarchy depth, breadth,

fan-out, cycle ratios, density, etc. The second kind of measures concern the intended functionality of the ontology, and include such examples as precision, coverage, and accuracy, which are all measured against some set of requirements over the ontology. The third kind of measures, finally, concern the communication aspects of the ontology, i.e., how it is documented, annotated, and understood by users. This includes measures related to recognition, efficiency, and interfacing. While the structural measures included in [89] are presented using formal definitions (in many cases even mathematic formulas) the functionality measures are less formally defined (giving some specific indicators, but mostly general method suggestions), and usability measures are even less defined (given almost entirely as examples or areas for future development).

The *oQual* formal model for ontology validation (and its associated meta-ontology) provides the bridge connecting the ontology engineering situation and context (modelled according to O^2) with the aforementioned measures, enabling validation that a certain developed ontology is sufficient and appropriate for the use for which it was developed. *oQual* includes concepts like value spaces and parameters over ontology elements, ordering functions for selecting parameters from different QOODs to prioritise, trade-offs that may need to be made, etc.

3.2.2 ONTOMETRIC

ONTOMETRIC, introduced in [90], is an approach for formalising ontology suitability for different tasks, heavily influenced by the Analytic Hierarchy Process (AHP) [91], an established method for aiding decision-making when dealing with multi-criteria problems. Per this method, a multi-criterion problem is broken down into the different criteria that need to be met (sometimes including sub-criteria, organised in a so-called decision tree structure or decision hierarchy). These criteria are sorted using a comparison matrix, such that the “competing” criteria on each level of the decision hierarchy are easily and intuitively compared pairwise, and the resulting prioritisation used to calculate a weighting of the total set of criteria with respect to the problem. When selecting between the available alternatives, the weighting can be used to calculate a total suitability score for each option. In an optimally performed AHP process, the alternative that receives the highest suitability score represents the best alternative given the character of the problem and the prioritisation of the criteria (in the case of ONTOMETRIC, the highest score would be associated with the most suitable ontology for reuse).

In order to support this method, ONTOMETRIC [90] provides a set of criteria for ontology suitability that can be used to populate an AHP decision hierarchy. These general criteria of ontology suitability are divided up into five different dimensions, representing different aspects of an ontology suitability problem: content, language, methodology, tool, and costs. In em-

ploying ONTOMETRIC, an ontology engineer will characterise the problem that their ontology aims to solve by using these five dimensions as top-level branches in their AHP decision hierarchy, and the individual ONTOMETRIC criteria as child nodes. The criteria will then be compared pairwise, a weighting generated, and the alternatives compared based on this weighting.

For the intended usage (i.e., ontology selection guidance for ontology engineers) ONTOMETRIC [90] seems very suitable. However, it does not seem possible to reuse in developing an ODP quality model. The method requires extensive ontology engineering knowledge to begin with, in order to perform the weighting of the provided (very technical) criteria, which are not in themselves associated with any predicted positive or negative effects. ONTOMETRIC does not in itself suggest which criteria are useful in which situations, it merely provides them as examples of things that can be measured and prioritised by the user. Given that an ODP quality model would need provide guidance for development, selection and use of Ontology Design Patterns by laymen and experts alike, the ONTOMETRIC criteria are unsuitable.

3.2.3 OntoClean

One of the most well-known methodologies for evaluating the conceptual consistency of ontologies is OntoClean by Guarino and Welty [92, 93, 94]. OntoClean uses a logical framework including very general ground notions and definitions from philosophy, believed to hold in any reasonable representation of the world, including an ontology model. The framework includes the notions of *rigidity*, *identity*, and *unity* as characteristics applicable to classes in an ontology, and a set of definitions regarding which taxonomic relations may exist between classes that exhibit these different characteristics. The listed characteristics are in OntoClean parlance denoted *metaproperties*. Below these metaproperties are explained and exemplified (examples are taken from [93] and [94]):

- **Rigidity** – Rigidity is related to *essence*. A class is considered *essential* for some individual entity if the entity must logically be member of said class at all times. Essence can be exemplified by the class *HardThing*, which is essential for a hammer (every hammer always must be hard), but not for a sponge (not every sponge must be hard all the time, though some might be at some times). By this definition a class is *rigid* if it is essential to all of its instances. The class *Human* can be considered rigid, because every instance of it must be part of it at all times – it is not possible to cease being a human and still exist. The class *Student* can be considered *anti-rigid*, that is, being a student is a non-essential for every student – all students can cease being students and still exist. *Semi-rigid* classes finally, are those that are essential to some instances, but not to others. As illustrated, *HardThing* is a semi-rigid class, as it is essential to hammers but not sponges.

- Identity – A class exhibits some *identity criterion* if that criterion (i.e., a property) can be used for recognising whether individual entities are the same or different. In database terms, this is a unique key for said class. A distinction is made between classes that carry their own identity criterion, and classes that inherit identity criteria from super-classes.
- Unity – Unity concerns whether instances of a class are considered whole entities. Consider the entity *5 cl of water*, which can be part of an ontology, but which cannot be said to be a *whole* object or entity of its own, and contrast it to the entity *Atlantic Ocean*, which is clearly a whole self-standing entity. In order to distinguish what is meant by whole and what defines wholeness, a *unity criterion* is used, examples of which include topology, morphology, functionality, etc. OntoClean differentiates between three types of classes: those carrying *unity* (all their entities are wholes sharing unity criteria), those carrying *non-unity* (all entities are wholes, but possibly with different unity criteria), and those carrying *anti-unity* (not all entities are required to be wholes). By this definition the class *Ocean* would carry unity and the class *AmountOfWater* would carry anti-unity. Non-unity can be exemplified by the class *LegalAgent*, provided that its instances could include both people and companies (which have different unity criteria).

Given these definitions, a set of constraints on the subsumption hierarchy are then defined [94]:

1. If a superclass is anti-rigid, then its subclasses must be anti-rigid.
2. If a superclass carries an identity criterion, then its subclasses must carry the same criterion.
3. If a superclass carries a unity criterion, then its subclasses must carry the same criterion.
4. If a superclass has anti-unity, then its subclasses must also have anti-unity.

By annotating the classes in an ontology using the OntoClean metaproperties and checking whether the above constraints hold, a developer can test whether their ontology is conceptually and philosophically consistent with regards to the notions of rigidity, identity and unity. It should be noted that this is not a guarantee that the ontology is sound with respect to real world phenomena or requirements. The methodology has been applied beneficially in a number of projects [95, 96, 97, 98]. Plugins supporting the use of OntoClean in different ontology engineering environments have also been developed, including WebODE [99] and Protégé 2000².

²<http://protege.stanford.edu/ontologies/ontoClean/ontoCleanOntology.html>

3.2.4 Terminological Cycle Effects

In [100] Lefort et al. study the structures of ontologies, in particular those structures that result from adhering to the W3C Semantic Web best practices workgroup recommendation for meronymy modelling, and the effect of said structures of the computability of an ontology. Using a number of different state-of-the-art reasoning engines they find that reasoner performance over large ontologies to a large degree is dependent on the structure of the Ontology Design Patterns within, and that in particular, the existence of asserted or inferred terminological cycles is detrimental to performance. Such terminological cycles occur when a concept occurs on both sides of a description logic equivalency definition, i.e., when a concept is defined wholly or partially in terms of itself. In meronymy this can easily occur in a reasoner inferencing process if both of the inversely related *hasPart* and *isPartOf* properties are used in class definition restrictions. Furthermore Lefort et al. [100] note that the computational performance characteristics of a reasoner-ontology pair is highly dependent on the description logic language used.

3.2.5 ODP Documentation Template Effects

In their master thesis Lodhi and Ahmed [101] study and suggest improvements to the presentation of ODPs, that is, how ODP documentation is structured and displayed. They focus on three main issues: firstly, resolving which information in pattern documentation that is most important for establishing an understanding of said patterns allowing the pattern to be used, secondly, whether novice and expert pattern users differ with respect to this question, and thirdly, how existing practice for presenting patterns can be improved or complemented in light of this. The patterns and documentation templates studied are all taken from the NeOn Ontology Design Patterns portal³. Lodhi and Ahmed [101] perform two online surveys, targeting novice and expert users respectively. Both of these surveys indicate that there are certain fields of ODP documentation that are by users considered particularly important in understanding a pattern, and that those fields include the graphical representation of the pattern, pattern scenarios (i.e., example usages), an OWL building block, competency questions, etc. Fields which are not considered of as high importance in these regards (though still relevant) include textual descriptions of ODP elements and ODP domain classification.

³<http://ontologydesignpatterns.org>

Chapter 4

Research Method

One of the key differentiators between ad-hoc trial-and-error and a planned research project is the selection and application of methods suitable to the research task at hand. Throughout this licentiate project, several such methods have been employed. The following chapter introduces some methods in the computing disciplines and details how these methods have been employed in gathering knowledge required to answer the established research questions.

4.1 A Perspective on Methods in the Computing Disciplines

Before entering into a discussion on method, it is relevant to frame the work performed in this thesis in terms of the academic tradition to which it adheres and the methods employed within said tradition. The following section gives a brief introduction to the relevant disciplines and some commonly used methods.

In academia, the development and use of computer and information systems are studied within a number of related academic disciplines, each of which carries their own academic tradition with certain more or less accepted perspectives on epistemology and philosophy of science, preferences with regard to methods and methodological issues, and well known and oft-quoted figurehead names. On a coarse grained level, the computer-related sciences can be divided into three such disciplines (each of which can be subdivided many times): *Computer Science*, *Software Engineering*, and *Information Systems* [102] (while in some perspectives the latter two disciplines are considered to be sub-disciplines within Computer Science topic-wise, for the purpose of method tradition enumeration it is sufficient to consider them distinct).

Of these three, the work performed in this thesis aligns most closely to Software Engineering: it concerns the utility and quality of IT artefacts from which software systems are built (i.e., Ontology Design Patterns), from both a technical and usage-oriented perspective. Consequently, the methods described and discussed below are approached from a Software Engineering perspective. As illustrated by Basili [103], research in Software Engineering concerned with products, processes, or people is best performed using an inductive as opposed to deductive approach, using either quantitative or qualitative methods. For the benefit of the layman reader, these terms are briefly explained below.

In the deductive paradigm, hypotheses are derived from a predictive theory. These hypotheses are tested empirically, and if they are invalidated by this testing, the theory is disproven. In this paradigm, the scholar applies general theory to a specific case [103, 104]. The deductive paradigm is most clearly exemplified by a physics experiment, in which a natural sciences theory (for instance the Newtonian law of universal gravitation) is used to develop a hypothesis (a falling object will accelerate towards the earth at approximately $9,81 \text{ m/s}^2$), which can be evaluated via experiment, possibly disproving the original theory.

In the inductive paradigm on the other hand, individual empirical observations about some phenomenon are used as grounding for the postulation of general laws and generalisations regarding said phenomenon [104]. Theories generated inductively are developed in an evolutionary manner, and updated as more observations and experiments are made, supporting parts of them and disproving other parts. In this perspective, a theory can be viewed as a model of reality that the researcher through different means tries to capture and understand [103].

In performing research within either of these two paradigms, the scholar may employ quantitative or qualitative research, data-gathering, and analysis methods. In quantitative research, empirical investigation is performed via (often large scale) numerical/statistical study and analysis of some phenomena. The data gathered is most commonly structured and homogenous. Examples include measuring the performance metrics of some software, using a survey form with graded questions (e.g., “How do you rate this feature on a scale of 1-5”), or studying the prevalence of some characteristic in a large population of entities. In these types of approaches, the scholar prepares the data gathering activity, but does not intervene in the actual data collection process, instead remaining the impartial observer [105]. Because of this, there is little risk that the data is tainted by the researcher’s own opinions or prejudices. However, this “hands-off approach” also means that quantitative method approaches cannot easily be used for exploratory research in a new field, in which the scholar adapts data gathering based on what comes up in the process. Instead, a theory or hypothesis must be established already [105].

Such quantitative methods can be contrasted by qualitative methods,

in which the emphasis is not on the volume of the gathered data, nor the homogeneity and structure of it, but rather its depth and explanatory potential. In employing these methods, the researcher studies the “hows” and the “whys” of some phenomenon under study [106, 107]. Examples include in-depth interviews with experts, observation studies within software engineering projects, or usability evaluation of design prototypes. The emphasis on depth of observations as opposed to volume makes qualitative results difficult to generalise to a broader population, but for illuminating a single case, or a set of cases, within a limited scope or domain, qualitative method approaches can prove superior to more shallow quantitative studies [108].

In addition to the distinctions of deductive versus inductive knowledge gathering and quantitative versus qualitative methods, methods can also be grouped based on scope. Some method frameworks encode large and overarching theories, making rather general recommendations including perspectives on philosophical ontology (not to be confused with Semantic Web ontologies) and epistemology. Examples of this include Design Science and the Pragmatist approaches. Other methods have a more narrow focus, recommending how to approach a certain problem in terms of selecting and employing data gathering methods in order to answer research questions, or how to work with, or study, concrete situations. This level can be exemplified by research method approaches like case studies, action research projects, ethnographies, etc. Finally, on the lowest and most detailed level, we have concrete data collection methods, prescribing how to gather and analyse data in practice, by using interviews, questionnaires, experimental procedures, participant observation logs, etc.

In the following sections the different research methods employed in this thesis are introduced and classified in terms of the above discussed dimensions. Note that these classifications are somewhat simplified views of an often rather complex reality – in many categorisation schemes, research methods may fit into several different boxes, and many times researchers do not even agree on what those boxes should be (see for instance the divergent classifications used by [102] and [109]). The following presents the author’s perspective on method issues, as relevant in the context of this thesis.

4.1.1 Systematic Literature Review

In order to gain an overview of a certain phenomenon, or of the state of research concerning that phenomenon, systematic literature reviews can be employed. By this method, a large number of research articles (ideally all available relevant ones) are found, evaluated, interpreted, and summarised in order to either answer some research question regarding the phenomenon in question, or to learn what type of work has been done on researching said phenomenon. Such a study can be very helpful in theory generation, or in isolating so far unresolved research questions for further study. The method

is frequently used in healthcare, where the same phenomenon or class of phenomena is often studied in the scope of different projects and research groups each with their own publications [110].

An important difference between a systematic literature review and other types of unstructured study of academic papers, is that the former employs a highly structured and procedural approach to finding and analysing the available literature. By being so formal in selecting papers to read, selection bias (i.e., that researchers only read and use papers that they agree with) is avoided. What it in practice means is that in each step of the literature review, search parameters, analysis methods, and metadata extracted must be defined before search, analysis or extraction takes place. In reporting the review, these parameters and methods are, for the sake of transparency and repeatability of the study, published alongside the results of the review itself. Once the selection parameters and analysis methods have been selected, the researcher's role becomes to use their judgement and analysis skill to perform whatever grouping, coding or analysis of articles is required, within these confines.

Kitchenham [111] suggests a number of tasks to perform in such a systematic literature review, covering all steps of the process, from topic definition and database search to analysis and documentation (the lattermost being required for the results to be accepted in a peer review context). Kitchenham also emphasises that this process may often require iterating over steps and backtracking in the process. For instance, sometimes key parameters for literature search need to be updated if the found volume of papers is too low, or if the usefulness of the returned papers in answering the defined research question is insufficient.

In terms of the aforementioned dimensions of method, systematic literature reviews occupy an interesting middle ground between qualitative and quantitative methods. The systematic way in which they are performed and the shared criteria by which all returned papers are evaluated are to an extent quantitative and return rather homogenous data, while the analysis process for each paper requires the use of human judgement and evaluation, which is inherently qualitative. That the method has characteristics of both traditions means that it, in order to be palatable for scholars on both sides of the qualitative/quantitative fence, needs to be extensively documented and structured, so as to be considered both transparent and trustworthy.

Another issue to note is that since a review does not in itself produce new knowledge, but rather depends on what has been published before, employing this method requires that a sufficient volume of research has already been published on the subject of study. Performing a systematic literature review in a new or young research field is unlikely to yield rich results. Examples of systematic literature reviews being employed in the Software Engineering field includes Ivarsson & Gorschek's review of technology transfer studies in the Requirement Engineering Journal [66], and Schneider et al.'s review of solutions to globalisation challenges in Software Engineering [112].

4.1.2 Case Studies

In the case study method, the researcher studies real world cases relevant to the object of study, in that object's natural context, with the goal of developing a more thorough understanding of the object than may be had in an artificial, or lab, situation. The term has sometimes been used sloppily to refer to any research activity influenced or driven by a real world scenario or example. Such usage is unfortunate, as it devalues the term and may give rise to the perception that case study-based research lacks in structure and is essentially just a subjective experience report. On the contrary, performing an empirical case study in accordance with recommended practice requires careful planning and consideration regarding which cases to select for study, how to approach them, how to analyse them, and how to triangulate data across multiple cases or multiple data gathering activities within a single case, in order to obtain useful and trustworthy results [113]. Yin's case study definition succinctly summarises this last point:

1. *A case study is an empirical enquiry that*
 - *investigates a contemporary phenomenon in depth and within its real-life context, especially when*
 - *the boundaries between phenomenon and context are not clearly evident.* [114, p. 18]
2. *The case study enquiry*
 - *cope with the technically distinctive situation in which there will be many more variables of interest than data points, and as one result*
 - *relies on multiple sources of evidence, with data needing to converge in a triangulating fashion, and as another result*
 - *benefits from the prior development of theoretical propositions to guide data collection and analysis.* [114, Ibid.]

Case studies are qualitative in nature, in that they focus on the deep understanding of one or a few cases, rather than a more general understanding of a large number of occurrences of a phenomenon (though within the context of the individual case it may sometimes be possible to apply quantitative data gathering methods). It is important to note however that case studies are not only employed when needing to understand the single case. They can also be used to compare two different situations (i.e., cases) that differ in ways in which the scholar is interested, to disprove existing theory (a task for which a single case may be sufficient), or in exploratory research as grounds for development of new theory [108, 114].

In developing and performing a case study, there are a number of steps that need to be taken. The case(s) relevant to the research question need to

be identified, the scholar needs to gain access to these, to establish a sufficient degree of trust with case participants, data collection activities need to be planned and executed, the data analysed, and the results communicated to the research community [114]. In the optimal situation the researcher has access to many promising cases that can help illuminate the phenomenon under study. It is the experience of the author however that this is seldom the case.

As mentioned in the above definition, a case study depends on a variety of different types of data regarding each case being gathered and analysed. Since many of these activities tend to be qualitative and therefore at risk of subjective researcher bias, the case study researcher aims to support any assertions made by multiple data sources, such that for instance interview material, observation logs, and document studies support one another and point in the same direction regarding the aspect of the phenomenon under study that a researcher assertion concerns. This is what is mentioned in the definition as “*data needing to converge in a triangulating fashion*” [114, p. 18].

4.1.3 Interviews

Interview methods are often employed to gather data from stakeholders or case participants familiar with the phenomenon of study, without necessarily observing the phenomenon itself. In a software engineering context, such interviews can be performed to for instance gain an understanding of how an artefact is appreciated by users, or to elicit requirements on a system or service. Sometimes interviews are used to follow up on an observational study, in which the interview subject is asked to explain or motivate observed behaviour, in order to establish a better understanding of the observations made [115].

Interviews can be *structured*, *semi-structured*, or *unstructured*. In the structured interview case, the interviewer has designed questions beforehand, and ask the questions exactly as written, making no deviations from the interview script. The resulting interview transcript and recorded answers can then be analysed in a statistical/quantitative manner, much in the same way as if the interviewee had filled out a questionnaire form. This type of interview is suitable in the case that the researcher has a very clear and specifically defined information need. In unstructured interviews on the other hand, the data gathered is of a much more qualitative nature. Here the interview questions are more open-ended and the interviewer and interviewee can have a rather broad discussion. Such interviews are suitable for establishing a basic understanding of a new field, where the researcher cannot know beforehand what type of knowledge they are looking for. Semi-structured interviews finally, lie somewhere in between, in that the interviewer has an interview script, but allows deviations from this to

occur if an interesting and potentially valuable topic of discussion comes up [115, 116].

When working with data gathered via interviews, the interview recordings are first transcribed into text, before those text transcripts are analysed in a documented and transparent manner using some established analysis method. For instance, in analysing qualitative interview material, it is common to split the transcripts into thematic fragments, and tag those fragments using keyword codes. Once the entirety of the material has been coded in this manner, the researcher can easily review the material and summarise the available interview material concerning a particular theme or aspect of the phenomenon of study. In order to reduce the impact of any individual researcher bias, these processes are often performed in parallel by multiple researchers [117]. Consequently, this transcription and analysis process can be very time-consuming, for which reason entirely unstructured interviews are not common in practice – it is simply not cost-effective to perform such analysis work without guiding the interview towards topics of interest to the researcher [115].

4.1.4 Experimentation

A typical experiment in the natural sciences is characterised by the testing of a hypothesis by studying the effect on a set of output parameters (*dependent variables*) by changes to some input parameters (*independent variables*). In a controlled experiment, the assignment of research subjects to experimental input conditions is randomised, and any environmental variables not themselves being studied are kept identical between different groups of subjects. Furthermore, such an experiment features at least one control group, in which the research subjects are not subjected to changing independent variables.

Basili [103] argues that the Software Engineering research community is lacking in such experimental maturity and that it needs to establish methods for how to apply experimental procedure in practice. He suggests a differentiation between evolutionary experiments, in which some model's or tool's suitability as solution to a problem is evaluated for the purpose of improving said solution, and revolutionary experiments, in which entirely new solutions are developed. In both of these approaches experiments are used not in order to test hypotheses in the classical deductive sense, but to develop understanding by refinement, through an inductive process; by developing better tools and models, the researcher develops better understanding of the underlying problem. Experiments of this nature may take place in the lab or in the field, and the results may be *descriptive* (some patterns in the data are found), *correlational* (correlations between independent and dependent variables are observed), or *cause-effect* (a causal relationship can be traced between independent and dependent variables).

Basili emphasises that in order for an activity to be considered an experiment rather than an observational study or a simple development activity, certain criteria need to be fulfilled. Firstly, there must be a goal of developing a new, deeper understanding of the underlying model or problem, that is, there must be thorough evaluation, measurement and analysis taking place. Secondly, there needs to be some defined treatment or researcher-controlled variable identified. Aside from these restrictions, [103] does not define many constraints on what may be considered an experiment or not – for instance, there is by this understanding no requirement that data resulting from Software Engineering experiments need necessarily to be quantitative in nature. The author finds this perspective on what constitutes an experiment to map very well to the realities of Software Engineering research, and has adopted Basili’s perspective on experimentation in this thesis.

4.2 Description of the Research Process

The methods described in Section 4.1 have been employed in different parts of this thesis work, as illustrated in the project overview displayed in Figure 4.1. In that figure, rectangles denote research activities, ellipses denote artefacts resulting from said activities, and rounded rectangles indicate existing (reused) research work. The work performed in and the method issues associated with the main research activities are introduced in the following sections, while the results of said activities are presented in Section 2.5 and Chapters 5 and 6.

4.2.1 ODP Literature Study

In the early phases of the thesis project an overview of the state of research into Ontology Design Patterns was needed, in order to find which areas of this research were underdeveloped and possible to do an interesting and relevant licentiate project in. To this end, a systematic literature review was planned and performed – as mentioned in Section 4.1.1, such a review can be helpful in learning what type of work is being done on studying a certain phenomenon or within a certain research field. Specifically, the purpose of the literature review was to learn what kind of research on ODPs was being performed, how the field has developed over time, where such research is taking place, and how it is performed. The results of this literature review are presented in Section 2.5.

The method used to perform the literature survey is illustrated in Figure 4.2. It consisted of first finding the relevant research papers (i.e., papers that discuss ontology patterns), then classifying and sorting them based on various metrics and measures in order to obtain the data needed to answer the research questions. The method, the selection criteria, and the steps to be taken in were defined beforehand and applied equally to all studied papers, as required by the systematic literature review method discussed in

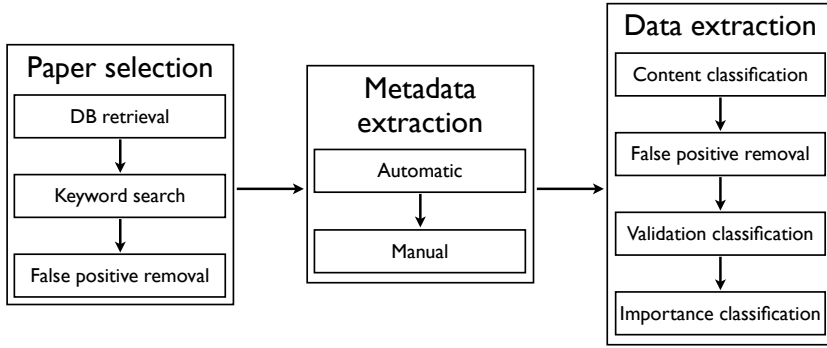


Figure 4.2: ODP literature study overview.

Section 4.1.1. The following sections discuss each of these performed steps in more detail.

Paper selection

The literature review covered the ISWC, ESWC, and ASWC conference series from 2005 to 2009, and the associated workshops. The reasons for these delimitations were three:

- The conferences in question are three of the most well-established and prestigious Semantic Web conferences in the academia. Papers that have been accepted to them are therefore likely to be of a high quality and representative of the general direction in which the Semantic Web research field is heading.
- The current interpretation of Ontology Design Patterns was introduced in [7] and [6], both of which were published in 2005. This is therefore a natural starting year for the survey. 2009 was selected as the ending year for the simple reason that the 2010 conferences had not all been held at the time the review was carried out.
- After initially performing the selection process detailed below with only the main conference proceedings as source, it was found that the number of papers returned were rather few. In order to gain more data, the scope was widened to include the workshop proceedings also, on the intuition that the workshops while being narrower in focus may still be relevant markers of trends and directions in ontology pattern research (see for instance the WOP¹ workshops that deal specifically with ontology patterns).

¹Workshop on Ontology Patterns, <http://ontologydesignpatterns.org/wiki/WOP:Main>

The available publications from these conferences and workshops were retrieved from the relevant publication databases. In order to find the subset of papers dealing with ontology patterns, the downloaded papers were subjected to a full-text search. All papers containing the phrases *ontology patterns* or *ontology **word** patterns* (**word** denoting any one single word) were selected for further analysis. Thereafter, in order to weed out false positives, papers mentioning patterns only in the reference list were removed from the dataset.

Metadata Extraction

Four types of metadata was determined to be useful for the purpose of this literature study: publication year, author provenance (research institution), author count, and institution count. These pieces of information were retrieved from the dataset. The publication year was retrieved automatically by way of database queries and web spidering scripts when downloading the source material, whereas the research institutions and institution/author counts were ascertained by studying the papers manually.

Research institutions were counted and ranked by the number of mentions they received in the headers of papers in the dataset. However, due to time constraints, further study of the specific organisational structure of the various institutions was not performed. Consequently, different departments of the same university were counted as belonging to the same unit, as were different branches of a company or other research organisation. This gives a course-grained view of what universities and organisations are involved in this type of research.

Data Extraction

While the metadata extracted could sufficiently answer some of the questions motivating the literature review, some of the questions (in particular the questions on what type of research is being done, and how it is being done) required structured analysis of the actual contents of the research papers. For this purpose the papers were categorised by ODP usage, by validation method, and by how central they considered ODPs.

For the ODP usage classification, the papers were read and tagged with one or more labels categorising how they related to or made use of ontology patterns. Since it was not known beforehand what categories would best describe the collected material, the labels used could not be decided ahead of time. Instead, the labels were selected in an exploratory fashion as the readings of the papers progressed. However, to ensure as unbiased a classification as possible, each label was paired with a definition covering papers belonging to the category. The labels and definitions are listed in Table 4.1. The papers were studied twice, once before and once after deciding upon the categories. At the end of this tagging procedure the papers that were

Table 4.1: Content categories and definitions.

Category	Definition
New pattern presentation	The paper presents a new ontology pattern.
Pattern usage method	The paper presents a new general approach or method of using patterns to achieve some goal(s).
Pattern creation method	The paper presents a new approach for isolating or creating Ontology Design Patterns (including re-engineering from other knowledge representation forms).
Patterns used	The paper describes a case where patterns have been used for achieving some goal(s)
Evaluation	The paper focuses on evaluating patterns or pattern usage/creation methods
Pattern typology	The paper discusses or suggests different types or groupings of patterns.
Pattern features	The paper discusses specific features of ontology patterns. This includes features of not only the reusable reference implementation, but also the documentation and meta-data associated with it.
Pattern identification	The paper deals with finding instances of patterns in an existing ontology.
Pattern languages	The paper discusses languages or formalisms for representing or displaying patterns.
Anti-patterns	The paper concerns anti-patterns or worst practices.
Not relevant	The paper is a false positive - it does not deal with ontology pattern research, but only mentions the term in passing.

classified as belonging to the *Not relevant* category were pruned from the dataset.

In order to survey how ontology pattern research is validated, two procedures were followed. To begin with, the papers were categorised according to in what manner validation or testing of the proposed ideas and theories had been performed. For this purpose, the following four categories were used:

- **No validation** – there is no mention of any validation of the ideas presented.
- **Anecdotal validation** – the paper mentions that the research has been validated by use in an experiment or in a project but it provides no detail on how this validation was performed or its results.
- **Validation by example** – one or more examples are presented in the text, validating the concepts presented in a theoretical manner.
- **Empirical validation** – some sort of experimental procedure or case study has been performed.

Each paper was assigned to one validation category only. In the cases where a paper matched more than one category, the category mapping to a higher level of validation was selected, i.e., empiricism trumps example which in turn trumps anecdote. Having categorised the papers by validation technique, a further study of validation quality was performed against the papers

categorised as belonging to the *Empirical validation* group. For this study some of the metrics and corresponding measurement criteria developed in [66] were reused: *Context description* (grading how well a paper describes the context, academic or industrial, in which empirical data gathering took place), *Study design description* (grading how well a paper describes the setup and method of the performed empirical validation), and *Validity description* (grading how well a paper describes validity issues and limits of generalisability associated with the performed empirical validation).

In order to learn how important the use of or research into ontology patterns is to each particular paper, it was studied in which section of the paper that patterns were mentioned. The intuition was that this information would give a crude indication as to whether ontology patterns were considered an essential core part of the research (warranting inclusion in the title or abstract) or not. Title inclusion indicates greater importance than abstract inclusion, which in turn indicates greater importance than mere inclusion in the body text. For this measure, terms such as “Knowledge patterns”, “Semantic patterns”, or just plain “patterns” were also considered acceptable synonyms for ontology patterns, provided they were used in a manner indicating a link to ontologies and the use of such patterns in an ontology engineering context.

The results of applying this method are presented in Section 2.5. These results led up to the research questions posed in Section 1.2.

4.2.2 Initial Quality Model Development

Rather than develop a quality model entirely from scratch, the author decided early on to take as input and inspiration established work on similar models from neighbouring research fields, adapting these to fit the specific characteristics and uses of Ontology Design Patterns. This approach grounds the proposed model in established theory and practice, and provides a solid starting point for development.

Quality metamodel development

A large number of quality models have been proposed for various different types of IT artefacts, as discussed in Chapter 3. These differ not only in their *content* (i.e., which specific instances of qualities, indicators, attributes, methods, or other concepts that they include and relate) but also in their *metamodel*, that is, how they understand, conceptualise and represent quality as it relates to whichever type of artefact that they are intended to support. Thus, establishing such a metamodel structure upon which to build the ODP quality model was from the outset considered important both in terms of structuring the problem and developing design and evaluation methods, and in terms of communicating the results to the research and practitioner communities.

The metamodel was developed iteratively and updated based on observations in pre-studies and prototype evaluations. Development of the metamodel and the quality model with which it is populated was roughly sequential in that the base of the metamodel was designed first and the quality model second. However, minor shortcomings of the metamodel discovered during the second half of the work were rectified and the metamodel updated accordingly.

The design of the first iteration of the quality metamodel was influenced by perspectives on how to model quality aspects originating from the MAPPER project [67, 68], introduced in Section 3.1.1. In this project, a metamodel is formalised which supports the development of a project result validation framework. While the domain of study in MAPPER is different than that studied in this thesis, the metamodel used is general enough to capture broader understanding of quality concepts in different fields. The result of the metamodel development work is presented in Section 5.1.

Quality model

While the MAPPER project framework influenced the thesis project metamodel design, the artefacts evaluated by this framework were deemed far too different from Ontology Design Patterns for the framework to be reused for the actual quality model. Instead, inspiration was taken from software quality models and conceptual model quality frameworks. Quality characteristics and indicators were drawn from the ISO 25010 software quality standard [70], from the PhD thesis “On the Quality of Feature Models” by Christer Thörn [69], from a number of papers on conceptual models (including UML and ER models), and from small scale pre-studies involving master students at Jönköping University.

Development of the initial ODP quality model consisted of five main steps:

1. Starting with the full set of quality characteristics defined in ISO 25010, specialise the quality definitions to apply particularly to Ontology Design Patterns.
2. Remove any quality characteristics that do not apply to Ontology Design Patterns.
3. Add quality characteristics and indicators defined in *On the Quality of Feature Models* [69] that are relevant to ODPs and that are not already present.
4. Add relevant quality characteristics and indicators from other related literature that are not already present.
5. Add or modify quality characteristics and indicators based on small-scale pre-studies.

The first four steps were carried out sequentially. The studies and prototype development mentioned in the fifth step were carried out in parallel and throughout the development process, but the resulting characteristics and indicators were not integrated with the other results until the end of the process. Each of the steps is described briefly below, and the results of these steps and their contribution to the initial ODP quality model are detailed in Sections 5.2 (steps 1-4) and 5.3 (step 5).

The ISO 25010 [70] quality models are introduced in Section 3.1.4. Conceptually, the quality framework used by these models is compatible with the already developed metamodel, which simplified reuse and adaptation of ISO 25010 for the purpose of developing an ODP quality model. The ISO 25010 standard introduces two quality models, the Quality In Use model, and the Product Quality Model. The Quality In Use model was deemed unsuitable for use within this project due to its emphasis on the quality of a whole Human-Computer System, as opposed to a clearly delineated IT artefact. The granularity of this perspective simply does not match the granularity of Ontology Design Patterns, which are used as building blocks in ontologies, which in turn are used as building blocks in software systems. The Product Quality Model (hereafter PQM) was however considered suitable as a starting point for development, and was subsequently used as an initial building block in the development of the ODP quality model.

The majority of the PQM quality characteristic definitions pertaining to software were possible to specialise for Ontology Design Patterns easily, by replacing references to concepts like “software”, “system”, “requirements”, etc. with concepts specific to ODPs, including “patterns”, “ontology”, “competency questions” and so on. Some required interpretation and rephrasing to be transferrable, but only a few were entirely inapplicable to use with ODPs.

The Thörn quality model for feature models [69] is presented in Section 3.1.2. While such feature models are not built using an as expressive formal language as Semantic Web ontologies and Ontology Design Patterns, the two types of models share certain characteristics (see Section 3.1.2) and intended usages suggesting that quality characteristics for one may be applicable for the other also. In developing the ODP quality model, quality characteristics formalised by Thörn were mapped against the already reused ISO 25010 quality characteristics [70]. Any quality characteristics that could not be mapped in this way, i.e., that were unique to the Thörn quality model were studied as potential candidates for inclusion in the ODP quality model. While some of these candidates were not suitable for use in an Ontology Design Pattern context, others were clearly understandable and useful, and were added to the quality model.

At this stage of development the ODP quality model held only quality characteristics, i.e., abstract concerns or perspectives on ODP quality not directly measurable themselves (*Learnability*, *Reusability*, etc.). In order to make the model more usable by scholars and practitioners, a set of

measurable indicators believed to affect these quality characteristics were developed, drawn primarily from existing literature on ontology quality [88, 89, 100, 118] and ER model quality [76, 77, 78, 119].

Prestudies

In parallel with the above development, two small-scale prestudies with master students were carried out, to develop an understanding of ODP usage, and to provide input into developing the quality model. These studies helped to provide some insight into possibly relevant quality characteristics and indicators to focus attention on. However, they were not large enough or methodologically rigorous enough to qualify as empirical evaluation of the model itself – a partial such evaluation is instead presented in Chapter 6. The two most influential prestudies are described briefly below.

During the early stages of development of the quality model, the author had the opportunity to discuss with and interview a master student performing a master thesis project on semantic search, who employed patterns for ontology development. The project in question involved two master students and was run together with the Jönköping County Council (Jönköpings Läns Landsting), within the context of the urology department at Ryhov County Hospital. For their thesis project the students were tasked with designing and developing a system for finding appropriate manuals and documents for performing particular processes in healthcare, based on the role of the user performing the search.

Early in the thesis project the students were provided with two Ontology Design Patterns developed by the author². One concerned how to model and structure context-dependent information, and one concerned organisational structures, the latter essentially being an encoding of the Organisational Structure analysis pattern presented by Fowler in [120]. The students were also provided links to the ODP portal³ with a suggestion to in particular study some of the patterns therein: Action, AgentRole, ParticipantRole, Role Task, Roles, Tagging, and TaskExecution. The students were not instructed on how to use and apply these patterns, but were simply provided with them and told to use them if they so wished and if they found them applicable to their problem. The interview sessions were held half-way through the project and towards the end of it, matching the conceptual design phase and the implementation phase in terms of actual programming.

Of the two students taking part in the master thesis project, one was interviewed on three occasions (the other did not decline to participate, but was simply unavailable at the times selected). The interviews were prepared with interview question manuscripts, but these manuscripts were not strictly followed – instead, a probing and questioning approach was employed, with the manuscripts used to get back on track when required (i.e., a

²Available at <http://purl.karlhammar.com/data/ph1/master-thesis-project/>

³<http://ontologydesignpatterns.org>

semi-structured interview format was followed, as discussed in Section 4.1.3). The results of these interviews fed into the design of the quality model in terms of particular indicators for primarily usability-related qualities.

In the latter stages of development of the quality model, another small scale experiment was performed in the context of an Information Logistics course held within the master program in Information Engineering and Management at Jönköping University, in late 2011. In this course, a lab session was arranged at which the participating master students were tasked with developing an ontology for a particular purpose, and provided with patterns to use in this development. This experimental lab session was optional, and the students who preferred to do so were free to take another regular lab exercise instead. However, due to the nature of the experimental setup, the optional experiment lab exercise had relaxed grading requirements, essentially a pass/fail grading based on presence during the lab session, and consequently the majority of the students taking the course attended this session rather than the ordinary one.

At the lab session the students were organised into randomised groups of two students each. Half of the groups were then given a certain set of five patterns, and half another set of five. These two sets differed in that one of them contained patterns that were more abstract and general in nature, whereas the other one contained patterns that were more concrete and specific, in some cases derived from the more general ones⁴. Most of the patterns provided came with textual descriptions and example OWL files, while a few had only the example OWL file and no textual description. The students' task was then to model an ontology based on a provided scenario in the domain of security and surveillance, in order to fulfil a set of provided competency questions. Upon completing this task, the students handed in the developed ontology and filled out a followup questionnaire on their experiences of using Ontology Design Patterns for this purpose, and their opinions on useful features of such patterns.

4.2.3 Knowledge Fusion Case Study

The Knowledge Fusion case study was set up with the intention of studying the impact of some of the indicators of the developed quality model, particularly with regard to indicators believed to affect usability and compatibility. By having case participants work together as a group on solving problems using Ontology Design Patterns, observations were made that tested the relevance and perceived impact of some of the previously developed indicators and qualities, and also allowed for the generation of new theory based on observations of how participants actually used ODPs. This approach made use of the proximity to and participation within a real world project, a setting which was not available to the author within the two other, more observational and quantitative evaluations of the quality model described later in

⁴Available at <http://purl.karlhammar.com/data/phl/ilog-lab/>

this chapter. The method employed in this study is described below, and the results of the study are presented in Section 6.1.

The objectives of the case study evaluation were formalised into three research questions:

1. How do users select and make use of ODPs?
2. What ODP characteristics do participants find helpful or harmful in ODP use?
3. What effects of ODP use on ontology engineering performance and resulting ontologies can be observed?

The case study method was selected for two reasons: firstly, its suitability for studying real life phenomena away from an artificial or laboratory setting (the phenomenon in this case being ODP use in ontology engineering for knowledge modelling purposes), and secondly, its suitability in studying context-bounded phenomena (the context in this case being the development of a system for complex event processing using semantic technologies, which the author had the opportunity to be involved with) [114].

In the initial development of the case, the author read and studied case related documentation in order to understand the requirements on the system which was under development at the partner research institute (hereafter RI) that hosted the project which the case study followed. RI was interested in learning how ODPs could be applied in the development of a system for complex event processing, or CEP (a concept introduced in Section 2.2.4), and for this purpose asked the author to help develop an ODP-based variant on their own developed CEP platform. Setting up such a system required developing an understanding of the constraints and requirements on the system as formalised in a large set of requirements documentation, which the author studied extensively.

Observation and data gathering were performed at a two-day modelling workshop at RI. The purpose of this workshop in the context of the project was to present the developments on the proposed system architecture and a prototype of the software to the participants, and to let them develop configurations for it, thereby validating the applicability of the approach to their deployment scenarios. Of course the author also had the personal intention of studying ODP use in practice by observing the participants work. This other motive was also, for the sake of transparency and trust, presented to the participants of the study and they willingly accepted acting as subjects in such observation.

During the modelling sessions data was gathered by way of audio and video recordings of the work in progress, photographs taken of ontology prototypes on the whiteboard, and notes taken on perceived key actions, behaviours, and trends taken independently by two researchers, the author and a professor with extensive experience of this research method. By acting as passive observers of the ongoing ontology development process, the

researchers were able to gain a perspective on real life usage of ODPs, including difficulties and problems in usage that the subjects experienced. Occasionally the subjects asked the researchers questions on ontologies and semantic technologies - these questions were answered insofar as they concerned technical specifics or details (such as the participants might have been able to gather themselves via a web search), but questions regarding modelling practice, how to solve a particular problem or which pattern to use for a given task, were not answered, so as not to interfere with the case.

At the end of the second workshop day a semistructured group interview was held where the participants were queried about a number of different aspects of their experience and opinions on ODP use. The purpose of this more active data gathering activity was to revisit and discuss issues and statements of particular interest observed during the workshop, and to resolve conflicting interpretations by the researchers. However, care was taken not to use this interview to enforce a group consensus in the cases that the subjects expressed diverging opinions. Such situations were instead noted and kept for analysis.

Data Analysis

Upon completing the workshop, the recorded material was transcribed into text. The vast majority of the material was immediately understandable. In the cases where ambiguities required interpretation, markers were put down. Those sections were revisited at the end of transcription, when a greater experience of the participants' voices was established, and in the majority of cases then resolved. The few uncertainties that remained were clearly marked out in the transcribed text, and subsequently ignored in later analysis steps.

The text material (notes and transcripts) was then analysed according to established transcript analysis methods [117, 121], as discussed in Section 4.1.3. All the texts were read through and fragments coded by theme. The texts were read twice, once to establish coding categories in the material, and once to apply codes to the text corpus. The fragments were grouped by code, and the collected material pertaining to each code studied to see what conclusions could be drawn regarding participant experiences, opinions and behaviour.

Validity and Generalisability

As touched upon in Section 4.1.2 and mentioned by [113] and [114], performing case study research in a reliable manner requires triangulating over multiple data points to gain as complete, transparent, and trustworthy a picture of the phenomenon of study as possible. During the modelling workshop described above, two researchers were involved in data collection and note-taking. Multiple data collection methods were employed (audiovisual recordings of modelling sessions, researcher notes, and interview tran-

scripts). Preliminary analyses made at the scene were verified against the case participants' opinions by way of a group interview at the conclusion of the sessions. However, due to resource limitations, coding and analysis was performed by only one person.

As in any single-case study, the generalisability of entirely new findings is limited, for which reason such findings warrant further empirical evaluation in other cases. The generalisability of findings which support existing empirically founded theory (i.e., those developed ODP quality indicators which are grounded in empirical studies) is in this regard higher.

4.2.4 Learnability and Usability Evaluations

The effects observed in the Knowledge Fusion case study, with a few exceptions, concern features relevant to pattern selection, pattern naming, pattern catalogues, etc. The majority of indicators within the initial quality model are however structural or quantitative in nature, and deal with features of the reusable OWL building block file accompanying most ODPs. Understanding the effects of these indicators would be helpful in guiding ODP developers produce patterns that are not only computationally and logically sound, but also easy to use for inexperienced ontology engineers (which is a major *raison d'être* of ODPs in the first place).

In order to attempt to test some of the more quantitative and structural learnability- and usability-related indicators of the developed model, a study was set up in the context of a course in Information Logistics at Jönköping University in 2012 (the results of which are presented in Section 6.2). The study aimed to answer three questions:

1. How do the indicators *Example illustration count* and *Documentation minimalism* affect ODP learnability?
2. How do the indicators *Anonymous class count* and *Class/property ratio* affect ODP usability?
3. How do *Property domain restrictions* and *Property range restrictions* affect ODP learnability?

The design of this evaluation was constrained by two main factors. Firstly, the educational context in which the study was executed necessitated that the study contain some practical lab tasks to be performed by students. Secondly, the number of participants available (initially estimated to 10-15) and the time constraints imposed by the educational context meant that an in-depth qualitative approach such as an interview study would be infeasible. To meet both of these constraints, an approach consisting of both survey questionnaires and practical tasks was designed. Surveys are useful in gaining an understanding of participant opinion of some phenomenon under study. While they lack in explanatory power compared to proper interviews, they take a fraction of the time of an interview to perform and analyse.

The requirement to have participants perform a practical task inspired the application of understandability and modifiability time indicators as proposed by Genero et al. [76], discussed in Section 3.1.3. Additionally, the application of such individually performed tasks have the added benefit of giving the participants immediate real experience of ODP usage in practice, which enables the usage of survey questions on such practical ODP usage issues which the participants would not have been able to answer had they only gotten a cursory overview of ODPs in a pattern repository.

In studying the first two research questions, the indicators under study acted as controlled variables, which were adjusted as discussed below, with the goal towards finding some interesting correlation against independent variables from either the time measurements applied or the survey responses received. This approach adheres to the Basili's [103] perspective on experiments in Software Engineering as discussed in Section 4.1.4. For practical reasons this approach was not feasible for the study of the third research question - varying one more parameter would have required considerably more time than was available with the subjects. Instead, the question regarding usability effects of property domain and range restrictions was put to the subjects through a survey form as described below.

Setting

The study was performed at Jönköping University, within a master course in Information Logistics. This course is located in the second year of the master program in *Information Engineering and Management*, and the students taking it have earlier in the program taken courses on knowledge modelling and knowledge management, database systems, and software engineering methods. They have in these courses studied and used the Semantic Web and ontologies, as well as ER and UML models. Additionally, in order to be accepted for the program, the students must have a bachelor degree in computer science, information systems, or a related field.

The study took place at a scheduled lab session, in a computer lab on campus, during a four hour afternoon session (though the second survey could be filled out also after this session). Attending the lab session was mandatory, but participating in the study was optional. In total 12 students opted in to take part in the study.

Study structure

The study consisted of three parts:

1. Survey 1, measuring ODP learnability effects of documentation-related indicators
2. Tasks, measuring ODP usability effects of structural indicators

3. Survey 2, surveying participant opinion regarding documentation-related and structural indicators

In **Survey 1**, the participants were presented with a randomised order of four ODPs sourced from the ODP portal and asked to answer a number of questions about them, gauging their understanding of the pattern in question. Each pattern displayed was presented using a template mechanism, by which the two controlled variables (i.e., *documentation minimalism* and *example illustration count*) were randomly adjusted for each participant. The understanding questions were of two forms; firstly, the participants were asked to mark which out of five competency questions the pattern was capable of answering, and secondly, a scenario description was provided and the participants were asked which class in the pattern corresponded to a certain term in the scenario text. Simultaneously, the time taken to answer the questions was measured, for the purpose of providing corroborating evidence of the ease or difficulty of understanding associated with each displayed pattern. Additionally, each survey ended with two questions on how concrete or abstract the participant considered the pattern, and how easy to understand they found it.

In the **Tasks** portion of the study, participants were asked to use patterns to help model a number of scenarios. In order to study the usability-related effects of the structural indicators *class/property ratio* and *anonymous class count*, patterns varying over these two indicators and being of non-trivial size (i.e., containing a minimum of 10 classes including imports) were selected for study. For each of these four patterns, the participants were tasked with modelling a certain scenario, using the provided pattern if they so wished. The participants had the option of using either of the two tools Protégé⁵ or TopBraid Composer⁶. They were not given any specific instructions on how to apply the patterns in terms of technology or method. However, they had at a previous lesson in the same course been instructed on the different methods available for this purpose, i.e., using owl:imports to import the pattern OWL file as-is, adapting that OWL file via subclassing or other modifications before importing it, or recreating the pattern from its documentation description into an entirely new OWL file. The time taken to complete this modelling task was recorded, and the resulting OWL files handed in.

Finally, in the concluding **Survey 2** portion of the study, the participants were surveyed on their opinions and impressions of ODPs and their features, now that they had used them for modelling a number of scenarios. The questions in this final survey concerned both the documentation-related indicators and the structure-related indicators under study, and asked the participants whether they found the presence of the features described by these indicators to be very helpful, helpful, neither helpful nor harmful,

⁵<http://protege.stanford.edu/>

⁶<http://www.topquadrant.com/>

harmful, or very harmful in understanding and using the patterns provided to them.

4.2.5 Performance Indicator Evaluation

While the studies described in the previous sections give some insight into usability-related effects of various ODP quality indicators and the usability and characteristics of ODP repositories, neither study captures the possibly substantial performance-related effects of ODP use and ODP structure. In the evaluation described in this section the author attempted to reconcile the previously developed initial ODP quality model with recent findings on ontology reasoning performance. In order to support this work, a literature study on ontology performance was performed, and the prevalence of indicators developed in the initial quality model (and also that found in the aforementioned literature study) in real world published patterns was studied.

The following questions were employed in this evaluation:

1. Which of the proposed performance-related effects of indicators from the initial ODP quality model are supported by existing literature?
2. Which indicators from literature known to affect the performance of reasoning with ontologies are also applicable to ODPs?
3. How do performance-altering indicators vary across published ODPs?

In order to answer these questions, the indicators from the initial quality model asserted to affect resulting ontology reasoning performance were selected from the initial model. A literature review across performance-related ontology research was performed, to find evidence supporting or disproving such performance effects of these indicators, and to add new indicators to the extent that applicable ones were found. Finally, the values of these proposed indicators among patterns “in the wild”, that is, published on the net in ODP portals today, were studied and analysed in order to learn whether these indicators actually do vary in practice, and how. Figure 4.3 presents an overview of the method. In this figure, oval shapes represent artefacts of study, and rectangular shapes represent research activities. The individual research activities are described in more detail in the following subsections, and the results of the evaluation are presented in Section 6.3.

Literature review

The indicators developed in the initial quality model were grounded in literature on or prestudies dealing with ontology quality issues. While such work is very helpful in studying issues like the logical correctness of an ontology or the usability of ontologies for various purposes, it seldom captures

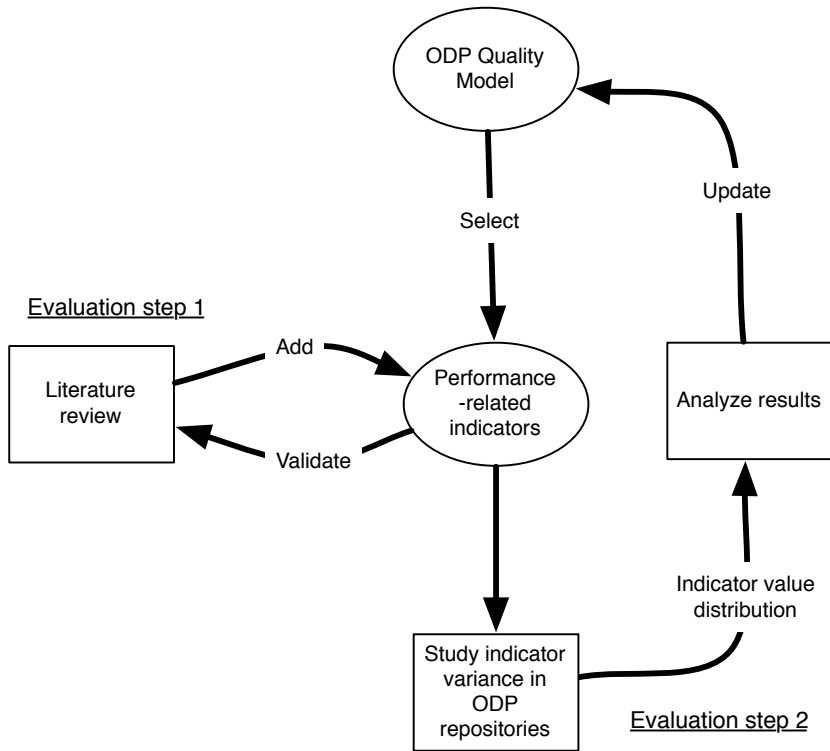


Figure 4.3: Performance indicator evaluation method

performance-related ontology issues, which are instead published and discussed about at conferences and in particular workshops focused on description logic languages and formal reasoning. The first goal of this evaluation was therefore to study the state of research in these fields, in order to learn whether the preliminary indicators developed could be supported by connections to known performance-altering ontology structures, and also whether the addition of new indicators to the ODP quality model based on such known structures was warranted.

For this purpose, publications at the main tracks and the associated workshops of four high-impact conferences dealing with formal knowledge modelling were studied, namely the International and Extended Semantic Web Conferences (ISWC and ESWC), the International Conference on Knowledge Capture (K-CAP), and the International Conference on Knowledge Engineering and Knowledge Management (EKAW). Timespan-wise papers published between 2005 and 2012 were selected and downloaded. This delineation was made based on the initial appearance of Ontology Design Patterns in research in 2005.

All papers matching the above criteria were downloaded, and their abstracts studied. Abstracts mentioning metrics, indicators, language expressivity effects, classification performance improvements or performance analyses (in total, 16 papers) were selected for thorough reading. Of these, eight were found to contain evidence supporting, disproving, or complementing the existing indicators of the initial quality model.

Study of indicator variance in ODP repositories

While getting a better understanding of performance-related effects of ODP indicators is an important goal in itself, if the work is to be applied in practice one needs also to study how these indicators appear in ODPs developed and used by the community, in order to provide guidance regarding which patterns are more or less suitable for different types of computationally expensive reasoning tasks. Such study may also help in validating the need for a particular indicator in practical ODP use and development. For instance, if all known patterns display a very small variance in one known performance-altering indicator, this may indicate that the indicator value results from either an established modelling practice (in which case a useful recommendation for the community may be made) or from some inherent characteristic of ODPs (in which case the use of the indicator provides little value). Accordingly, the second goal of the work presented in this section was to study how these performance-related indicators varied among the patterns available in the pattern repositories used by the community.

To this end, the reusable OWL building blocks of the patterns from two well known ODP repositories, <http://ontologydesignpatterns.org> and <http://odps.sourceforge.net>, were downloaded and studied. A modu-

lar expandable tool for measuring ontology or ODP metrics was developed⁷ specifically for this purpose. The Java-based tool parses an input ontology (or in this case, ODP module) and based on which metric measurement plugins are located in the tool's classpath, measures different aspects of said ontology. It generates as output CSV data suitable for post-processing in a spreadsheet or statistics tool. Plugins for all of the performance related indicators under study (with two exceptions, detailed in Section 6.3) were developed for this tool, and it was then executed over the downloaded pattern set.

Analysis of indicator variance

In analysis of the data from the execution of the indicator measurements, a simple four step process was repeated for each indicator under study:

1. Sort all ODPs by the studied indicator.
2. Observe correlation effects against other indicators. Can any direct or inverse correlations be observed for whole or part of the set of patterns?
3. Observe distribution of values. Do the indicator values for the different patterns vary widely or not? Is the distribution even or clustered?
4. For any interesting observation made above, attempt to find an underlying reason or explanation for the observation, grounded in the OWL ontology language and established ODP usage or ontology engineering methods.

In performing the above analysis, several interesting correlations were discovered and studied, as shown in Section 6.3. In some cases, an explanation for the correlations based on the structure of the OWL language and the constructs within it could also be generated. Since these explanations could be grounded in and motivated by the actual OWL language implementation, they were considered to be sufficiently motivated to warrant updating the quality model accordingly. On the other hand, such explanations for correlations that could only be motivated based on an understanding of established practice in ontology engineering were not considered to be sufficiently trustworthy for inclusion in the updated quality model.

The above approach in studying indicator variance can in part be considered a descriptive experiment in the Basili sense [103], i.e., an attempt to isolate patterns in data through a structured process of evaluation, measurement and analysis. However, since there is no specific treatment or comparison against a control group established, this method does not wholly fulfil the Basili experiment definition. For this to be the case, some task including modification indicators within the ODP set would be required. This is considered future work in order to further evaluate the ODP quality model.

⁷<https://github.com/hammar/OntoStats>

Chapter 5

Initial Quality Model

In order to answer the research questions put forth in Chapter 1, the author has developed a quality model for Ontology Design Patterns. The overall method by which this model was developed has been described in Section 4.2. The following chapter describes the process and results in greater detail.

5.1 Quality Metamodel Development

Ontology Design Patterns are inspired by both traditional software engineering design patterns and reusable software components. Like the former, they can emphasise the logical solution to a type of problem, and express this problem-solution mapping in text and diagrams. Like the latter, they can, and often do, include implementation modules ready to plug in and adapt. A general conceptualisation of ODP quality must cover both of these aspects, allowing for both modelling of design pattern-style qualities that are intangible or difficult to measure using purely quantitative metrics, and of more traditionally quantifiable software component-style qualities. Furthermore, since ODPs are used in the creation of IT artefacts, such a conceptualisation needs also to allow for the modelling of the IT artefact construction contexts in which the patterns are used.

The metamodel developed within this thesis project fulfils the above requirements, and is believed to represent a suitable and relevant understanding of ODP quality in the general case. In it, a differentiation is made between abstract quality characteristics like usability or performance (the importance of which depend on the specifics of the ODP use case) and the more concrete and measurable indicators which affect said characteristics, for better or worse. This understanding is conceptually compatible with the existing quality frameworks introduced in Chapter 3.

The developed metamodel is displayed in Figure 5.1. The topmost half of Figure 5.1 displays the relation $R(D, ODP, OU)$ where D denotes a domain, ODP denotes an ODP Use (*ontology engineering, ontology matching,*

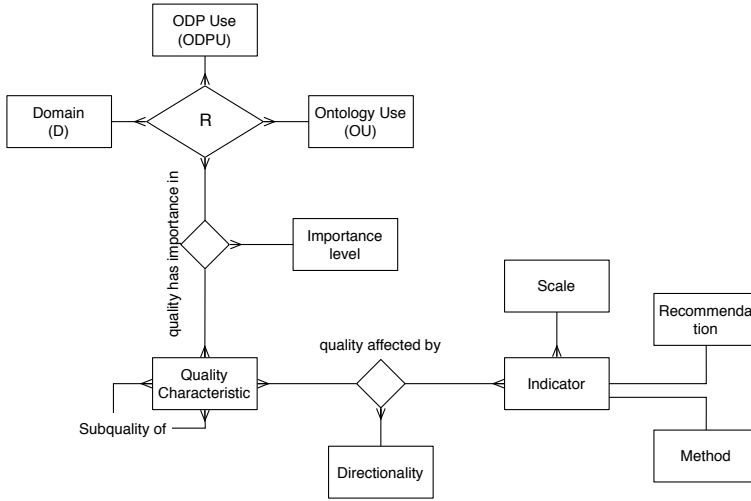


Figure 5.1: Quality Metamodel

transformation, etc.), and *OU* denotes a use to which the result of the ODP usage, e.g. an ontology, is put. This last concept can be exemplified in the case that the deliverable is an ontology by *competence modelling*, *event scheduling*, etc. or in the case that the deliverable is an alignment, by for instance *web service composition*. An instantiation of this relation *R* captures a concrete usage example of ODPs. For each such instantiation of *R* a number of quality characteristics affect the result. The term *quality characteristic* is here used synonymously with the definition in the ISO 25010 standard [70] on software quality.

As an example, *ease of use* may be a quality characteristic which is beneficial to the ODP usage situation *R(zoology, ontology engineering, resource supply estimations)*. Some quality characteristics are believed to be more abstract and consist of compositions of other quality characteristics. For instance, *ease of use* could be composed of *documentation quality*, *format compatibility*, etc. Quality characteristics are affected (positively or negatively) by indicators that are measurable using some metric. Indicators are here used similarly to the concept *quality property* in ISO 25010, as concrete properties of an IT artefact that affect some quality characteristic. For instance, the quality characteristic *documentation quality* could be mapped to depend on a number of indicators, one of which is *has graphical illustration*.

Indicators have scales of metrics, in accordance with the definitions established by Stevens [122] on nominal, ordinal, interval, and ratio scales. They also have recommendations that give guidance regarding what are relevant values for indicators to assume, to the extent that such can be given, and method definitions for how indicator measurements are to be sampled,

to the extent that it is not obvious from the indicator name itself.

The concepts used in the metamodel are defined as follows:

- **Domain** – A classification of the social or business environment or area in which the developed artefact is to be deployed.
- **ODP use** – A classification of the use to which an ODP is to be put.
- **Ontology use** – A classification of the use to which the delivered artefact generated by ODP usage is to be put.
- **Importance level** – The importance of a quality characteristic to a particular ODP usage situation.
- **Quality characteristic** – Denotes a particular aspect of ODP quality. Quality characteristics may be decomposed into sub-characteristics. Quality characteristics represent concerns or perspectives on quality on an abstract level. They are not themselves directly measurable using some metric or method.
- **Directionality** – Denotes the effect that an interpreted “high” value of an indicator has on a quality characteristic. Expressed as *positive*, *negative*, or *null* (i.e., no or unknown value).
- **Indicator** – Individually measurable properties of an ODP that contribute to some quality characteristic(s).
- **Scale** – The type of scale whereby an indicator is measured. One of *interval*, *nominal*, *ordinal*, or *ratio* scales.
- **Recommendation** – A textual recommendation on what values for an indicator that are relevant and beneficial to aim for in different ontology engineering contexts. Is optional.
- **Method** – A description of how to perform measurements of an indicator. Is optional.

5.2 Quality Model Development

As described in Section 4.2.2, the development of the initial ODP quality model consisted of adapting and reusing existing work on information systems quality, conceptual model quality, ER model quality, and ontology quality, as well as two smaller pre-studies. ISO 25010 [70] and Thörn [69] provided quality characteristics for the ODP quality model, and several other works provided indicators. The following sections describe each of the development steps and how the reused research contributed to the initial quality model in more detail. The resulting quality model is described in Section 5.4.

5.2.1 ISO 25010 Adaptation

The majority of the quality characteristics of the ISO 25010 Product Quality Model [70], described in Section 3.1.4, (hereafter PQM) were suitable for describing ODP quality as well, requiring only slight changes to quality definitions to replace software- or information systems-specific terminology or uses with ontology-specific equivalents. For instance the quality characteristic *functional completeness* from ISO 25010 is defined as *degree to which the set of functions covers all the specified tasks and user objectives*. The same quality characteristic, if translated to an ontology engineering context, would still be relevant, but the terms used in its definition would be different: *degree to which the ODP knowledge modelling ability meets expressed requirements*.

However, certain quality characteristics required more adaption to be reused. One example of this is the characteristics related to performance efficiency – since an Ontology Design Pattern is rarely, if ever, used on its own, these quality characteristics had to be rephrased to refer to the performance efficiency of the resulting ontologies created using patterns. Another example is the quality characteristic *compatibility* and its corresponding sub-characteristics, originally dealing with how well a system can co-exist with other systems in terms of resource allocation and message passing. In an ODP context, compatibility relates closer to interoperability in terms of shared base concepts and lack of definition duplication, and the quality characteristic definitions were revised accordingly.

Some quality characteristics were deemed too tightly coupled to the concepts of software and systems, and inapplicable in an ODP context. Such characteristics include:

- *Functional correctness* – In a software context it makes sense to speak of functional correctness and functional completeness as disjoint qualities - a system can perform only part of a specified task, but perform that part correctly. However, in an ODP context, a pattern can be considered correct only if it fulfils the requirements by which it is defined. A pattern which completely covers its defined requirements and models everything that it is required to model in the way that it is required to be modelled expressed per those requirements, is by definition correct. By this understanding, discussing functional correctness separately from functional completeness is unnecessary, making this quality characteristic redundant.
- *Security* – This characteristic and its sub-characteristics deal with the behaviour of a system in terms of authentication, authorisation, logging, etc. As both ontologies and ODPs are inactive non-executable components, such behaviour is not exhibited by them. Authentication, logging, auditing, etc. can be performed by software that is built on and operates using an ontology or knowledge in which information governing security behaviour is stored, but this does not imply that

the ontology exhibits any security characteristics in itself – rather, it means that the ontology is designed to support usage in this domain, which goes toward functional suitability.

- *Reliability* – As with the security characteristics, reliability and its sub-characteristics deal primarily with executable behaviour at runtime – how many hardware/software faults can the system sustain without failing, to how large a degree is the system operational when required, how well can the system recover from failure, etc. Again, ontologies and ODPs being passive components do not exhibit system behaviour and this characteristic is therefore not applicable to them.
- *Capacity* – While the other characteristics relating to performance efficiency can be rephrased to cover resulting ontologies as discussed above, this sub-characteristic dealing with maximum capacity (exemplified in terms of number of users, communication bandwidth, and transaction throughput) relates closer to executable programs, and makes little sense in an ODP context.

Finally, the three ISO 25010 [70] PQM quality characteristics *maintainability*, *portability*, and *compatibility* display a certain overlap when translated to apply to ODPs. This is likely due to the intended usage of these artefacts. The software systems that the ISO standard supports are normally deployable on their own, and thus it makes sense to differentiate between qualities pertaining to where and how they can be deployed (*portability*), how well they integrate with other systems (*compatibility*), and how they can be reused in construction of other systems (*maintainability*). However, ODPs are from the beginning intended to be standards-compliant (in the sense that they adhere to the RDF/RDFS/OWL standards, not necessarily to some specific conceptualisation) modules that are used and reused together in constructing ontologies. Consequently, portability and compatibility are difficult to untangle for ODPs - a pattern that is conceptually compatible with other patterns is also portable in the sense expressed in the ISO standard, i.e., that it can be transferred between different usage environments. Patterns which are easy to adapt and replace are not only portable, but also support maintainability.

In studying the definitions of these qualities present in the ISO standard, it was decided that the ones sorting under the top-level quality *compatibility* were most applicable to ODPs, and that the portability qualities from the standard were either inapplicable to ODPs or, in this context, could be seen as specialisations of other existing qualities (adaptability of modifiability, installability of operability, and replaceability of interoperability). Consequently, *portability* and associated sub qualities were removed from the model. Since in an ODP context improved pattern reusability does not necessarily contribute to the maintainability of a pattern or a pattern-based ontology, but does imply that the pattern can be reused and integrated

Table 5.1: ODP Quality Model after ISO 25010 adaptation

Quality characteristic	Subcharacteristic
Functional suitability	Functional completeness Functional appropriateness
Performance efficiency	Time behaviour Resource utilisation
Compatibility	Co-existence Interoperatbility Reusability
Usability	Appropriateness recognisability Learnability Operability User error protection User interface aesthetics Accessibility
Maintainability	Modularity Analysability Modifiability Testability

Table 5.2: Mapping of Thörn quality characteristics to ISO 25010

Thörn	ISO 25010
Usability	Usability
Reusability	Compatibility
	Performance efficiency
Correctness / Formalness	Functional suitability
Changeability	Maintainability
Mobility	Portability
	Security
	Reliability

more with other ontologies (i.e., is more compatible with other ontologies) the sub-quality *reusability* was moved from being a sub-quality of *maintainability* to being a sub-quality of *compatibility*. The resulting initial quality model based on ISO 25010 is displayed in Table 5.1.

5.2.2 Thörn’s Qualities

Thörn’s model [69] includes six quality characteristics; changeability, reusability, formalness, mobility, correctness, and usability. The majority of these quality characteristics are easily translatable to match the top-level quality characteristics in the ISO 25010 standard as illustrated in Table 5.2.

While the Thörn quality model does not include lower level quality characteristics, the preliminary first iteration of that model also presented in [69] does define and argue for the existence of a set of such more specific characteristics. These lower level characteristics also match the characteristics already present in the ODP quality model based on the ISO 25010 standard to a large degree, as evidenced by Table 5.3. However, some attributes pro-

posed by Thörn do not have suitable matches in the initial ISO 25010-based ODP quality model. Of these attributes a subset deal with issues that make them strong candidates for adaptation and inclusion in the ODP quality model, as suggested below:

- *Accuracy* – Degree to which the pattern represents the domain being modelled. While a pattern may be functionally complete and correct simply by fulfilling its design criteria (no matter what those criteria are), accuracy is here intended to reflect how consistent the pattern is with regards to generally accepted understanding of the domain in question. In other words: is the pattern design criteria reasonable in the real world?
- *Consistency* – The characteristic that determines absence of contradictions in the ODP. A pattern which holds or suggests conflicting axioms is obviously going to be difficult to apply in a real world case including any reasoning requirements, but it could still be useful for simple vocabulary tasks.
- *Stability* – Denotes perceived change expectation. A stable ODP is developed with the intention of covering the foreseeable evolution of the modelled concepts with relatively few changes to the pattern.

These characteristics and their definitions were added to the ODP quality model, resulting in the updated set of quality characteristics displayed in Table 5.4.

5.2.3 Reuse of ER Model Quality Research

Several methods of evaluating ER models are introduced in Section 3.1.3. Some of these metrics and methods are strong contenders for inclusion in an ODP quality model also.

In [76] Genero et al. study the learnability and modifiability effects of a number of metrics on ER models. While the specific metrics studied in this experiment are to a large degree specific to ER models (including for example metrics like *number of N:N-relationships* or *number of composite attributes*), the method in which the understandability and modifiability of models is gauged via measuring the time taken to respond to an understandability questionnaire, and the time needed to update said models, are easily transferrable to an ODP context. Consequently, understandability and modifiability time measures are included in the ODP quality model, as indicators for the corresponding sub-characteristics.

In [119], the same authors study and summarise existing data model quality metrics. Upon reading this summary it is apparent that a great deal of the existing work on data model metrics is specific to ER models. However, some of the measures mentioned and proposed can be translated into indicators applicable to ontology and Ontology Design Pattern qualities

Table 5.3: Thörn initial quality model mapping to ODP quality characteristics

Thörn quality model quality attribute	ODP quality model quality subcharacteristic
Acceptability	Appropriateness recognisability
Accessibility	Accessibility
Accuracy	
Adaptability	Modifiability
Analysability	Analysability
Communicativeness	User interface aesthetics
Completeness	Functional completeness
Complexity	
Conformance	Interoperability
Consistency	
Extensibility	Reusability
Installability	
Interoperability	Interoperability
Learnability	Learnability
Modularity	Modularity
Portability	
Redundancy	
Reliability	
Robustness	User error protection
Self-containedness	Co-existence
Structuredness	
Testability	Testability
Understandability	Learnability
Visibility	

Table 5.4: ODP Quality Model after reusing Thörn [69]

Quality characteristic	Subcharacteristic
Functional suitability	Functional completeness
	Functional appropriateness
	Accuracy
	Consistency
Performance efficiency	Time behaviour
	Resource utilisation
Compatibility	Co-existence
	Interoperatbility
	Reusability
Usability	Appropriateness recognisability
	Learnability
	Operability
	User error protection
	User interface aesthetics
Maintainability	Accessibility
	Modularity
	Analysability
	Modifiability
	Testability
	Stability

also. These translated indicators deal primarily with the size or complexity and includes the number of nary relations and the number of redundant axioms, both of which are believed to affect the usability and performance characteristics of an ODP and are therefore included in the quality model.

Both Moody and Shanks [77] and Lindland et al. [78] focus on the importance of reducing unnecessary content (that is, such content which is not required for the model to be functional) in conceptual models. By this understanding, redundant axioms are not only axioms which can be inferred from the rest of the model (as the more technical perspective on the issue presented in [119] might suggest), but also axioms which are not required by ODP design criteria or competency questions. This type of minimalism with regards to competency questions was also added to the model as an indicator.

Moody and Shanks [77] also suggest evaluation of ER model understandability via user ratings, and recommend that three user categories be considered in such rating. These three categories have been mapped to ODP usage categories (knowledge engineers, system operator/configurator, and end-user/input clerk) and are added as indicators contributing to the *usability* quality sub-characteristics.

5.2.4 Reuse of Established Ontology Quality Research

As has been covered in Section 3.2, a great deal of research has already been developed on formalising the quality of ontologies. Obviously, a lot of this work is applicable to Ontology Design Patterns. Rather few publications deal with qualities and quality characteristic in the senses described and defined in the previous sections, so the contributions of existing work to this portion of the quality model is limited. However, much work deals with concrete and measurable qualities mapping well to the indicator definition given in Section 5.1.

The combination of O^2 and *oQual* by Gangemi et al. [88, 89] presented in Section 3.2.1 provides for a very comprehensive ontology evaluation and selection method. Unfortunately the method is also very complex and requires a high degree of familiarity with description logic, knowledge modelling, and semiotics to learn and use, skills which the intended users of Ontology Design Patterns are rather unlikely to possess. For this reason, while the entirety of the method has been studied, only the presented measures have been studied and in some cases selected for reuse.

The set of quantitative indicators from [89] measuring *structural* dimensions of ontology graphs are interesting and potentially useful. Some of the indicators here are likely unsuitable for use with ODPs, as they measure the presence of features that are unlikely to occur in small ontology modules such as ODPs (example indicators include subsumption fan-outness, density, degree distribution, etc). Other indicators measure features that are unlikely to occur frequently in an ontology module which is intended to be

used as a design pattern (for instance, indicators that concern the usage of instances). However, after filtering out a number of indicators from [89] considered to be unsuitable for ODPs, several indicators remain that are likely directly applicable to ODP evaluation also, and which were therefore added to the model. Listed by the quality characteristic that they are believed to affect (based on expected effects on quality as defined in [89]), those are¹:

- *Usability* – subsumption hierarchy depth, subsumption hierarchy breadth, tangledness, anonymous classes, class to property ratio, annotation ratio
- *Analysability* – size, axiom/class ratio
- *Resulting performance efficiency* – disjointness ratio
- *Compatibility* – tangledness, annotation ratio

The importance of good amount of integrated pattern documentation in the form of pattern comment lines (roughly translatable to a high annotation ratio) on usability is also expressed by Prechelt et al. [84], as described in Section 3.1.5. Prechelt et al. also report an experiment showing that this measure has an effect on the maintainability of produced solutions. Accordingly, the effects of annotation ratio on both the above quality characteristics, and maintainability, were added to the ODP quality model.

Some of the measures of *functionality* presented by Gangemi et al. [89] are also interesting and potentially useful, particularly the *precision* and *recall* measures (borrowing from established Information Retrieval theory). Unfortunately however, as formalised and expressed in [89], these measures depend on calculating the precision and recall in terms of correct conceptualisations of the world or domain of discourse vis-à-vis all possible interpretations of an ontology. While this may be a formally and philosophically correct model, it is unfortunately quite impossible to apply, for which reason these measures are not reused here. The *usability* measures from the same work are likewise interesting, but difficult to concretise and apply, for which reason they are also excluded.

The findings on performance related indicators associated with the use of certain types of design patterns by Lefort et al. in [100] discussed in Section 3.2.4 are natural candidates for inclusion in an ODP quality model. Those indicators (the number of terminological cycles present, and the complexity of the description logic language used) were added to the initial quality model indicators affecting resulting performance efficiency.

The author has in a previous work [118] studied the benefits and negative effects of aligning an existing ontology for modelling academic competencies and achievements to both well-established Semantic Web ontologies, and Ontology Design Patterns. Experiences from this project indicate that the

¹For full definitions of the referenced quality characteristics and indicators, see Section 5.4.

number of import statements (i.e., references to other ontologies, the logic statements of which are considered to be part of the importing ontology) in an ontology or Ontology Design Pattern can be an indicator for how easily adaptable and compatible this ontology or ODP is. Since the OWL language lacks features for partial import, and since imports are transitive, the total import closure of even a relatively small pattern or ontology can be very significant. Because of this, an ontology or ODP that imports many other ontologies will likely take significant resources to classify using a DL reasoner. Furthermore, due to tooling limitations in managing the display of imported concepts, such an ontology is difficult to visualise and work with. Consequently, the effects of import count on usability and computational performance were added to the quality model.

As mentioned in Section 3.2.3, the OntoClean [93] methodology is an established method for ontology evaluation. While applying OntoClean to larger ontologies is a possibly very time-consuming process, for Ontology Design Patterns (which are in one sense small scale ontologies) this ought not be as big a problem. OntoClean validation would likely be a suitable indicator for pattern accuracy, and this indicator was therefore added to the quality model.

A master thesis by Lodhi and Ahmed [101], introduced in Section 3.2.5 finds that certain fields in ODP documentation are considered to be of more importance than others in supporting the learnability of the patterns. The author’s intuition is that if this is the case, then possibly those relatively unimportant fields (of which there are quite a few) are a distraction for ODP users, making it difficult for them to quickly understand the ODP when first exposed to it. Consequently, the indicator *Documentation minimalism* is defined as a limitation on the data fields displayed in the ODP documentation in accordance with the fields found to be most important by Lodhi and Ahmed [101].

It should be noted that while the results of Lefort et al. [100], Lodhi and Ahmed [101], and the author’s own work [118] mentioned above are grounded in some empirical findings, the papers by Guarino and Welty [93] and Gangemi et al. [89] lack empirical evaluation of the proposed methods and metrics. Such an evaluation would need to be performed to validate whether these indicators are indeed associated with the quality characteristics as asserted here.

5.3 Empirical Pre-studies

In parallel with the development of the quality model described above, two small studies on using Ontology Design Patterns were performed and evaluated, providing input to the development process. The results of those studies and their effects on the developed quality model are presented below.

5.3.1 ODP Documentation Structure Interviews

As introduced in Section 4.2.2, this pre-study consisted of interviews with a master student performing a master thesis project using ontologies and Ontology Design Patterns in the healthcare domain. The student had been provided with two such patterns and references to several more, and the interview was performed to gauge his understanding and opinion of these patterns.

The interviewed student was at the time on the final year of his two year master program. He had previously obtained a bachelor in software engineering and computer science in his home country, before migrating to Sweden to pursue a master's degree there. Apart from his academic background, he had some experience of software engineering in industry, having worked with GIS technology and databases for the armed forces in his home country, and with web development and programming in the private sector. He had, prior to enrolling in the master program at Jönköping University never used semantic technologies or ontologies (the program does however contain courses giving overviews of these technologies). He had also not to his knowledge previously used OOP design patterns.

During the interviews, it became apparent that of the provided patterns, all had been studied, but relatively few were deemed relevant to the problem at hand. The ones that were seen as relevant were however very much appreciated and believed to be very helpful in designing an ontology. The key selection criterion employed by the students was the immediately perceived fitness for purpose, i.e., whether the pattern could be seen to match the perceived problem more or less directly. The provided competency questions were not mentioned by the student as a criterion in selecting a pattern for use, nor was the use of a pattern as a best practice recommendation for how to frame and understand the problem considered. Instead, patterns (when applicable) were used simply as a shortcut for implementing a solution compatible with how a problem was already understood. In this pattern usage case, understandability and learnability were the most important qualities.

The interviewed student strongly preferred task-oriented documentation in performing software and ontology engineering tasks, preferably with clear and pedagogic examples included, as opposed to typical API documentation formats or long and extensive white papers on whole technology stacks. He emphasised this preference also regarding patterns. He also suggested that more than one usage example and corresponding context be included in pattern documentation, with reference to how such pattern usages often are case dependent and how an understanding of several types of application scenarios would be beneficial.

In terms of the structure of the pattern documentation, the availability of graphic illustrations was also heavily emphasised. The student expressed a preference for himself doodling architecture diagrams when developing software, to help structure and understand problems, and found the same type of conceptual diagrams very helpful in understanding patterns and the

proposed solutions to problems they address. He recommended that both the pattern structure itself and one or more examples of pattern application should be presented graphically. However, the graphic illustrations need also to be backed by descriptive texts referencing their content (both in the case of illustrations describing pattern structure and pattern usages), rather than be entirely disconnected artefacts.

5.3.2 ODP Usage Experiment and Survey

This second pre-study took place in an Information Logistics course at Jönköping University, where a group of students were tasked with building an ontology using patterns, and then filled out a survey regarding that pattern usage experience.

The 29 students who attended the session were given a background questionnaire to gauge their existing knowledge and understanding of conceptual modelling, ontology engineering and Ontology Design Patterns. The majority reported having some prior experience with RDF (55 %) and OWL (90 %), primarily from previous courses. Only around 10 % had ever used Ontology Design Patterns for creating such ontologies however. Academically all respondents reported having bachelor's degrees or higher in Computer Science or related areas.

Judging from the results of this survey and from the ontologies handed in by the students, the usefulness of Ontology Design Patterns was apparent. Out of 27 respondents, 24 responded that at least one of the provided patterns was helpful in solving the prescribed task. Most groups used one or two patterns, with the average number used at approximately 1.5. While the initial idea was to get an understanding of user preferences with regards to the abstraction level and size of patterns, the followup survey and handed in solutions provided too little indication to say anything of consequence regarding such qualities or measures. There was a slightly higher number of patterns used that the author classified as specific patterns than those classified as general patterns, but not enough to call it a significant difference. The same was true for the survey, which indicated a slight preference for larger and more concrete patterns, but not enough to be significant.

In terms of the pattern usage process, approximately 60 % of the students reported having studied the accompanying OWL files to get an understanding of how to apply the patterns, whereas the remaining approximately 40 % did not look at the OWL file. Supporting such a usage method requires that the textual descriptions and graphical illustrations of the pattern be sufficient, so that the pattern can be understood on the basis of them. When asked to grade the understandability of the patterns that did not come with an accompanying description, 35 % of respondents reported poor or no understanding of those patterns.

For those patterns that did have accompanying descriptions, the majority of students responded that the competency questions and the graphical

illustrations were the most important in understanding how to use and apply the patterns. The former is in line with general consensus in the ODP research community, but the importance of the latter has in the author's opinion not been as emphasised. When asked to suggest improvements to the patterns used, the most common recommendation from the participant students was to add example scenarios. This is in line with the findings reported in the Section 5.3.1 emphasising the importance of more and better described example usages in pattern documentation.

Finally, a somewhat unexpected result of the followup survey was the clearly expressed preference among the participants (82 % of respondents) for object properties encoded in patterns to be restricted by having defined domains and ranges, as opposed to not having such domains and ranges defined. In comments given in a free form text field accompanying this question, respondents indicate that they think this makes the pattern OWL files a lot easier to learn and understand. The author shares this opinion, but would also add a plausible negative side-effect of such restrictions: it is likely to lead to a lower reusability of patterns, if the properties are confined to work on and with instances of classes defined within the pattern itself.

5.4 The Developed Initial Quality Model

The initial Ontology Design Pattern quality model developed is presented in the following sections. It is not exhaustive or complete – there are some blank spots, particularly in regard to concrete measurement methods for indicators, recommendations for reasonable values, and the directionality of indicator effects. Parts of the model are evaluated in Chapter 6, and certain of these blank spots filled, but a complete evaluation of the whole model remains to be done in a future PhD thesis.

5.4.1 Quality Characteristics

As defined in the quality metamodel in Section 5.1, quality characteristics represent aspects of ODP quality that affect different ODP usage situations. They are not directly measurable, but are indicated through measurable indicators. The quality characteristics of the model are presented below, grouped by the top-level quality characteristics to which the lower-level quality characteristics contribute. Each quality characteristic is given with an accompanying definition.

Functional Suitability

Degree to which an ODP meets stated or implied needs.

- *Functional completeness* – Degree to which the ODP meets expressed knowledge modelling requirements (i.e., competency questions and other design requirements).

- *Functional appropriateness* – Degree to which the ODP facilitates simple storage and retrieval of knowledge formalised according to its definitions (e.g., does the ODP require simple or complex SPARQL queries to retrieve knowledge).
- *Consistency* – Degree to which the ODP is internally logically consistent.
- *Accuracy* – Degree to which the ODP accurately represents the real world domain being modelled (e.g., whether it adheres to established industry standards and protocols, or legislation).

Resulting performance efficiency

Reasoner or system performance efficiency over ontologies created using the pattern.

- *Time behaviour efficiency* – Response or processing times when reasoning over or using resulting ontologies in a system.
- *Resource utilisation efficiency* – Amounts and types of system resources used when reasoning over or using resulting ontologies in a system.

Usability

Degree to which an ODP can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction.

- *Appropriateness recognisability* – Degree to which users can recognise whether an ODP is appropriate for their needs.
- *Learnability* – Degree to which an ODP's structure, and intended usage can be understood by users new to it, such that they can thereafter apply the ODP successfully and efficiently.
- *Operability* – Degree to which an ODP has attributes that make it easy to apply and use.
- *User error protection* – Degree to which an ODP prevents users from making modelling errors.
- *User interface aesthetics* – Degree to which the ODP's documentation (text, graphics, etc.) is pleasing for the user.
- *Accessibility* – Degree to which the ODP's documentation can be used by people with the widest range of characteristics and capabilities.

Maintainability

Degree of effectiveness and efficiency with which an ODP (and consequently, ontologies built using that ODP) can be adapted and modified by maintainers after deployment in some usage scenario.

- *Modularity* – Degree to which the ODP is composed of discrete components such that a change to one component has minimal impact on other components.
- *Analysability* – Degree of effectiveness and efficiency with which it is possible to assess the impact on an ODP of an intended change to one or more of its parts, or to diagnose an ODP for deficiencies or causes of failures, or to identify parts to be modified.
- *Modifiability* – Degree to which an ODP can be effectively and efficiently modified without introducing defects or degrading ODP quality.
- *Testability* – Degree of effectiveness and efficiency with which test criteria can be established for an ODP and tests can be performed to determine whether those criteria have been met.
- *Stability* – Perceived change expectation on the ODP - high stability denotes a low degree of change is expected, and vice versa.

Compatibility

Degree to which an ODP can successfully be reused and integrated with other ODPs or IT artefacts in the construction of ontologies or systems.

- *Reusability* – Degree to which an ODP can be used in more than one system, or in building other assets.
- *Co-existence* – Degree to which an ODP can coexist with other ODPs as modules in an ontology, i.e., without detrimental impact on other ODP modules.
- *Interoperability* – Degree to which two or more ODPs share definitions of co-occurring concepts.

5.4.2 Indicators and Effects

Table 5.5 lists the indicators contributing to the aforementioned quality characteristics, along with research article(s) from which each indicator was sourced, or, in a few cases, the section in this thesis describing the pre-study in which it was developed or observed. Additionally, a more exhaustive listing of the same indicators is provided, including indicator measurement

methods, scales of measure compliant with said methods, and indicator effects on quality characteristics. The latter are also for the majority of effects associated with a directionality, expressed as *positive* or *negative*. Directionality is interpreted such that a higher value for the indicator is associated with a increase in the affected quality characteristic if the directionality is positive, and a decrease in the affected quality characteristic if the directionality is negative. Indicators that are expressed as boolean values should when ascertaining effects on quality characteristic in this manner be understood as representing either zero (false) or one (true).

Table 5.5: Initial model indicator summary.

Nr	Name	Source(s)
I-1	Accompanying text description	Section 5.3.2
I-2	Annotation ratio	[84], [88], [89]
I-3	Anonymous class count	[88], [89]
I-4	Axiom/class ratio	[88], [89]
I-5	Class disjointness ratio	[88], [89]
I-6	Class to property ratio	[88], [89]
I-7	Competency question count	Sections 5.3.1 and 5.3.2
I-8	Direct import count	[118]
I-9	DL Complexity	[100]
I-10	Documentation Minimalism	[101]
I-11	Example text count	Sections 5.3.1 and 5.3.2
I-12	Example illustration count	Sections 5.3.1 and 5.3.2
I-13	Functionality questionnaire time	[76]
I-14	Minimalism	[77], [78]
I-15	Modification task time	[76]
I-16	Nary relation count	[119]
I-17	OntoClean adherence	[93]
I-18	Property domain restrictions	Section 5.3.2
I-19	Property range restrictions	Section 5.3.2
I-20	Redundant axiom count	[119]
I-21	Size	[88], [89]
I-22	Structure illustration	Sections 5.3.1 and 5.3.2
I-23	Subsumption hierarchy breadth	[88], [89]
I-24	Subsumption hierarchy depth	[88], [89]
I-25	Tangledness	[88], [89]
I-26	Terminological cycle count	[100]
I-27	Transitive import count	[118]
I-28	User evaluation ranking	[77]

I-1 Accompanying text description

Method: Check that the ODP OWL file is associated with a textual description document or webpage.

Scale: Nominal (boolean)

Affects characteristics: Usability (positively)

I-2 Annotation ratio

Method: Divide the cardinality of the set of OWL annotation property usages in the associated OWL file with the cardinality of the union of all class, property, and instance nodes.

Scale: Ratio

Affects characteristics: Usability (positively), Compatibility (positively), Maintainability (positively)

I-3 Anonymous class count

Method: Count the cardinality of the set of anonymous classes in the associated OWL file.

Scale: Ratio

Affects characteristics: Usability (negatively)

I-4 Axiom/class ratio

Method: Divide the number of axioms in the associated OWL file by the number of named classes.

Scale: Ratio

Affects characteristics: Analysability (positively)

I-5 Class disjointness ratio

Method: Take the ratio of the number of disjointness axioms in the ODP to the possible number of disjointness axioms given the number of classes (if C is the set of classes in the ODP, the latter is given by $(|C| - 1)!$). Add to this the ratio of all classes involved in a disjointUnion axiom to the total number of classes in the ODP.

Scale: Ratio

Affects characteristics: Resulting performance efficiency (positively)

I-6 Class to property ratio

Method: Divide the cardinality of the set of named classes in the associated OWL file by the cardinality of the set of properties.

Scale: Ratio

Affects characteristics: Usability (positively)

I-7 Competency question count

Method: Divide the number of competency questions expressed in the pattern documentation by the size (I-21) of the ODP.

Scale: Ratio

Affects characteristics: Learnability (positively)

I-8 Direct import count

Method: Count the owl:imports statements in the associated OWL file.

Scale: Ratio

Affects characteristics: Usability (negatively), Resulting performance efficiency (negatively), Reusability (negatively)

I-9 DL Complexity

Method: Ascertain the DL complexity of the axioms included in the ontology, e.g., by checking using the DL metrics tab in the Protégé tool.

Scale: Ordinal

Affects characteristics: Resulting performance efficiency (negatively)

I-10 Documentation minimalism

Method: Assert that the ODP documentation contains only the minimum documentation fields required to enable use of the ODP (e.g., graphical representation, examples, pattern intent, OWL building block link, OWL example file, competency questions, common pitfalls, and consequences of use).

Scale: Nominal (boolean)

Affects characteristic: Learnability (positively)

I-11 Example text count

Method: Count the number of written examples of ODP usage in the associated description.

Scale: Ratio

Affects characteristics: Appropriateness recognisability (positively), Learnability (positively)

I-12 Example illustration count

Method: Count the number of illustrations of example ODP usage in the associated description.

Scale: Ratio

Affects characteristics: Learnability (positively)

I-13 Functionality questionnaire time

Method: Apply a questionnaire on ODP functionality and usage, and have a set of representative users answer this questionnaire, measuring the time required for them to do so. In the case that the same participants take multiple surveys for different ODPs, ensure sufficient randomisation in survey ordering to avoid learning effects affecting the results.

Scale: Ratio

Affects characteristics: Learnability (negatively)

I-14 Minimalism

Method: Compare ODP axioms against its competency questions and other design restrictions, ensuring that no extraneous axioms, not required by design requirements, exist.

Scale: Nominal (boolean)

Affects characteristics: Learnability (positively), Operability (positively), Compatibility (positively)

I-15 Modification task time

Method: Define a set of modification tasks for an ODP, and have a set of representative users perform these tasks. Measure the time required to perform said modifications. In the case that the same participants perform multiple modification tasks for different ODPs, ensure sufficient randomisation in ODP ordering to avoid learning effects affecting the results.

Scale: Ratio

Affects characteristics: Modifiability (negatively)

I-16 Nary relation count

Method: Count the number of classes defined in the ODP associated OWL file that exist solely to mitigate the OWL binary property limitation, i.e., classes that would not be naturally expressed as their own nouns in natural language as expressed by representative domain-knowledgeable users.

Scale: Ratio

Affects characteristics: Resulting performance efficiency (negatively), Usability (directionality unknown)

I-17 OntoClean adherence

Method: Employ the OntoClean method to tag the ODP classes and properties with OntoClean metaproperties. Assert that the taxonomic structure of the ODP is compliant with the constraints imposed by the applied metaproperties.

Scale: Nominal (boolean)

Affects characteristics: Functional suitability (positively)

I-18 Property domain restrictions

Method: Divide the cardinality of the set of properties that have defined domain restrictions with the cardinality of the set of all properties in the ODP.

Scale: Ratio

Affects characteristics: Learnability (positively), Reusability (negatively)

I-19 Property range restrictions

Method: Divide the cardinality of the set of properties that have defined range restrictions with the cardinality of the set of all properties in the ODP.

Scale: Ratio

Affects characteristics: Learnability (positively), Reusability (negatively)

I-20 Redundant axiom count

Method: Calculate the number of asserted axioms that can be removed from the ODP OWL file without affecting the inferred model.

Scale: Ratio

Affects characteristics: Resulting performance efficiency (negatively), Usability (directionality unknown)

I-21 Size

Method: Add the cardinality of the set of OWL classes to the cardinality of the set of OWL properties in the ODP.

Scale: Ratio

Affects characteristics: Learnability (negatively), Analysability (negatively)

I-22 Structure illustration

Method: Assert that the ODP documentation includes at least one illustration of the classes and properties proposed by the pattern and how they relate.

Scale: Nominal (boolean)

Affects characteristics: Learnability (positively)

I-23 Subsumption hierarchy breadth

Method: Define a level as the set of all classes in an ODP that have the same number of hops via asserted subclass links to the top-level concept `owl:Thing`. Define the breadth of a level as the cardinality of that level. The average breadth of the ODP is then the sum of all breadths in an ODP divided by the cardinality of the set of levels.

Scale: Ratio

Affects characteristics: Usability (directionality unknown)

I-24 Subsumption hierarchy depth

Method: Define an ancestor path as a path through the asserted subsumption hierarchy linking a leaf node concept to the top-level concept `owl:Thing`. Define the depth of an ancestor path as the cardinality of that path. The average depth of the ODP is then the sum of all depths in the ODP divided

by the cardinality of the set of ancestor paths.

Scale: Ratio

Affects characteristics: Usability (directionality unknown)

I-25 Tangledness

Method: Divide the cardinality of the set of named classes which are asserted to have more than one named superclass by the cardinality of the set of all classes in the ODP.

Scale: Ratio

Affects characteristics: Usability (negatively), Compatibility (negatively)

I-26 Terminological cycle count

Method: Execute a DL reasoner over the ODP. Then calculate the number of occurrences of terminological cycles that occurs in it, i.e., concepts that occur on both sides of a DL equivalency definition and are therefore wholly or partially defined in terms of themselves.

Scale: Ratio

Affects characteristics: Resulting performance efficiency (negatively)

I-27 Transitive import count

Method: Calculate the cardinality of the set of OWL files found through a recursive search over the import hierarchy of the original reusable OWL building block associated with the ODP.

Scale: Ratio

Affects characteristics: Usability (negatively), Resulting performance efficiency (negatively) , Reusability (negatively)

I-28 User evaluation ranking

Method: Develop a survey on the usability of an ODP, expressing quantifiable rating-style questions on for instance the understandability of the pattern documentation components, the suitability of the pattern naming, and the aesthetics of the pattern documentation. Define each question metric clearly enough that ambiguity and interpretation can be minimised. Let ODP users of different categories (knowledge engineers, system developers, data input clerks) answer the survey. Pay particular notice to results from your intended ODP usage group (which may depend on context, but often is the knowledge engineers). In the case that fewer people are available to reply, consider employing delphi techniques [123] to arrive to one joint evaluation as opposed to several individual ones.

Scale: Interval

Affects characteristics: Usability (positively)

Chapter 6

Quality Model Evaluations

The quality model presented in the previous chapter is an initial but well-founded theory on how one may understand and structure the quality of Ontology Design Patterns. As any theory, it must be tested and validated, to ensure that it is in actual fact consistent, relevant and useful. The following chapter presents the work that the author has undertaken in testing different aspects of said model, updating it when so required. These tests are by no means exhaustive, and more work definitely remains to be done if one is to be able to say that the proposed ODP quality model has been truly and completely vetted. However, as we shall see, preliminary results indicate that quite a few of the suggested quality characteristics and indicators are useful in selecting and applying Ontology Design Patterns for ontology engineering cases.

6.1 The Knowledge Fusion Case Study

This section presents a case study of content pattern usage in configuration of an event processing system. The focus of this case study has primarily been on usability and compatibility related aspects of Ontology Design Pattern quality. A number of the hypothesised qualities and indicators were found to be relevant and helpful for the involved practitioners, but a number of possible new indicators were also seen. The case study setup and method issues associated with it are discussed in Section 4.2.3. The study has previously been published in [124].

6.1.1 Case Characterisation¹

The project framing this case study was a small spinoff project from a larger project on threat detection using sensor systems where the partner research

¹For reasons of integrity and confidentiality, the case description and published data has been anonymised.

institute (hereafter denoted RI) was involved. The work at RI focused on development of a rule-based complex event processing subsystem intended to help isolate and correlate critical situations and threats based on incoming data, to support human operators in decision making and response force deployment guidance.

Within the spinoff project that the case study covered, the aim was to develop the same functionality as the one in the Knowledge Fusion subsystem, but using Semantic Web-based technologies like ontologies and description logic reasoners. The motivation for this project from RI's perspective was twofold: they wanted to see if they could achieve higher flexibility of knowledge modelling and reasoning by using description logic languages as opposed to the more low-level pre-compiled rules used in the existing system. They also believed that using ODPs as preconfigured modules of functionality to plug in and out of the system could support reconfigurability, particularly by non-expert users.

Workshop tasks

Data collection was performed at a two-day workshop, at which the case participants from RI developed configurations for the ODP-based variant of the complex event processing system. Two scenario descriptions developed within the project were used to describe system deployment contexts². The participants then attempted to model some typical relevant critical situations associated with each of these scenarios. Two examples of such critical situations are listed below:

- A gang is four or more people who have been seen together via at least three cameras over at least fifteen minutes and who are all wearing the same colour clothing. A critical situation occurs when a gang of five or more football fans are loud and have within the last hour been spotted by a camera at a bar.
- Two vehicles are the same if they have the same license plate number *or* have the same brand, model and colour and are observed by two cameras located at the same physical place within five seconds. A vehicle is behaving oddly if observed driving less than 15 km/h in three different cameras.

To their aid, the participants had a set of twenty Ontology Design Patterns, of which fourteen were selected from the ODP community portal, and six were selected from other research projects (Table 6.1). They were not provided with any training in pattern use, and were not recommended any particular development method, on the basis that providing such recommendations or training would restrict the participants' behaviour and interaction with the patterns and the possibility of learning from their work.

²Downloadable from <http://purl.karlhammar.com/data/phl/odpcep/>

Table 6.1: ODPs provided for case study participants

ODP name	Source ³
Access Rights	Section 5.3 pre-studies
Accountability	Adapted from [120]
Action	ODP Portal
Action Distance	Section 5.3 pre-studies
Agent Role	ODP Portal
Authorised Position	Section 5.3 pre-studies
Basic Plan	ODP Portal
Communication Event	ODP Portal
Description and situation	ODP Portal
Nary Participation	ODP Portal
Observation	ODP Portal
Part Of	ODP Portal
Participant Role	ODP Portal
Participation	ODP Portal
Path	Section 5.3 pre-studies
Place	ODP Portal
Route	Section 5.3 pre-studies
Time Indexed Participation	ODP Portal
Time Indexed Situation	ODP Portal
Time Interval	ODP Portal

Participants

Three participants attended the modelling workshops, participants A, B, and C. They were all male, and were at the time of the study all in the age bracket from 35 to 55 years. All three were researchers (two PhDs, one MSc) in software engineering or conceptual and data modelling within RI, and all three had some experience in such modelling. B and C had little or no prior knowledge of Semantic Web ontologies and semantic technologies, whereas A had worked on these topics quite extensively, among other things researching rule languages for reasoning over Semantic Web ontologies. Their respective specialities were as follows:

- A had published on ontology matching, rule languages, model transformations, semantic technology use cases, etc.
- B had published on information logistics, mobile computing, context- and task-aware computing, etc.
- C had published on component based software engineering, middlewares, service orientation, system architectures, garbage collectors, etc.

6.1.2 Data

The resulting dataset comprises some 21600 words, or approximately 85 pages of text. Of these, 16 pages are researcher notes, and 69 pages are audio or video transcriptions. The participants were initially skeptical about being

³ODP Portal denotes <http://ontologydesignpatterns.org>. All patterns are downloadable from <http://purl.karlhammar.com/data/phl/odpcep/>

recorded on film, and their behaviour changed noticeably when cameras were present, becoming quite a lot more formal and tense. In order to promote a good natural working environment for observations, the researchers chose to turn off the recording equipment initially, turning it back on only when the participants had gotten warmed up to the task and seemed less concerned about this. Due to the triangulation in analysis, this is believed to have little effect on the reliability of the results however.

Additionally, six whiteboard illustrations were photographed. In analysing the gathered data, qualitative text analysis methods were applied as described in Section 4.2.3: the text was split into fragments based on content and each fragment tagged with a thematic code. In total there were 187 applications of such codes to fragments, with the distribution of fragments over codes shown in Table 6.2.

Table 6.2: Distribution of fragments to codes.

Code label	Fragments	Code label	Fragments
DL/semantics limitations	8	ODP structure	1
Efficiency	11	ODP usage prerequisites	11
Implicit ontology effects	1	ODP-attributable errors	8
Method/metamodel adequacy	6	ODPs-as-error-control	7
Modelling errors	1	ODPs-as-ground ontologies	8
ODP catalogue and selection	28	ODPs-as-guidance	21
ODP complexity	1	OE method observations	8
ODP effects	3	Pattern insufficient	12
ODP imports	10	Top-down/bottom-up choices	7
ODP method observations	19	Usefulness	13
ODP size	3		

6.1.3 Findings

The following section presents some observations and analyses pertaining to the research questions developed from the gathered data. Some of the analyses are also accompanied by reflections on how these observations support or weaken the initial quality model hypothesis presented in Section 5.4.

Important ODP Features

During the modelling and subsequent interviews, the issues of ODP *size* and ODP *import count* were brought up. The participants initially expressed divergent opinions regarding the effect of OWL import statements in ODPs. Participant A considered imports quite helpful in that the reconciliation of imported more general base concepts with one’s own model provided a good opportunity for validating the soundness of one’s own design. He also emphasised the advantage of getting a foundational logic “for free” that one would not otherwise have had time to develop. Participant C expressed an understanding of the tension between reuse and applicability presented by the import feature and large import closures, comparing it to discussions

in the object oriented design pattern community in the nineties. Participant B criticised the use of imports, on the grounds that the expansion of ODP size that such imports imply negatively affects ODP usability, and on the grounds that the base concepts included by imported patterns may be incompatible with one's own world view, being written for some other purpose:

“I really have to know what is there and what does it mean. And maybe it's written with some other focus, some other direction, some other goal. And I don't believe in this general modelling of the universe that fits all purposes.” – Participant B

Participant B also indicated that he would use the idea of a pattern as presented in a pattern catalogue and reimplement it, rather than reuse an existing OWL building block, if that block contained too many imports or dependencies. After some discussions Participant A agreed to the soundness of such a method in the case of a large import closure not directly relevant to the problem at hand. Both participants A and B proposed that a better solution would be to add support for partial imports to tools and standards.

In terms of the size of patterns, the participants emphasised during the interview session the importance of patterns being small enough to be easily understood in a minute or two of study. They considered an appropriate size to be three-four classes and the object- and datatype properties associated with them. They drew parallels to object oriented design patterns which are frequently of approximately this size. This expressed preference is consistent with the patterns they selected during modelling, all of which contained three or fewer classes, excluding imports.

Pattern Selection

It was observed that the single most important variable in ODP selection from the pattern catalogue seemed to be pattern naming. If a name “rang a bell” the participants proceeded with studying the pattern specifics to see whether the pattern was suitable in their case. This observation is supported by participant feedback at the interview session. The participants also suggested that description texts and competency questions were important selection criteria that should be emphasised in an ODP catalogue. Additionally, they considered the possible negative consequences of applying a certain pattern to a problem to be of particular importance in selecting and applying patterns.

On the subject of pattern catalogues, the participants indicated that they considered the two catalogues to which they had been exposed (the ODP community portal and the one developed for these sessions) to be unordered and unintuitive, holding patterns of varying completeness, abstraction level and domain, all mixed in one long list. The participants suggested that they would find it easier to navigate a catalogue that was structured according to topic, architecture tier, abstraction level, or some other hierarchy:

“You also know the old classification of upper ontologies, domain ontologies, and task ontologies. You know this old picture. This, at least this structure should be present.” – Participant A

Further participant suggestions for improvements to ODP catalogue usability included the addition of graphical illustration of pattern dependencies, and providing a semantic search engine across ODPs held in the catalogue. The former suggestion was inspired by an illustration from the Core J2EE Patterns web page⁴ that the participants found helpful in deciphering pattern intent, and which Participant C in particular argued would be helpful in understanding the structure of a set of ODP patterns. The latter suggestion was that a search engine be added allowing users to search through concepts and properties present in ODPs in the catalogue, ideally including NLP techniques to match for synonyms and related terms.

ODP Usage Method

The participants initially developed their designs on a whiteboard rather than on their computers. They used the patterns as guidance in development, rather than as concrete building blocks to be applied directly. When questioned on why this method of working was preferred, they stated that it was more flexible and required less commitment to a design in progress than immediately formalising to OWL code. The participants would build a prototype solution to a whole problem in one go, rather than tackle one part of the problem at a time. This method is contrary to eXtreme Design (as described in Section 2.4.2), which emphasises modular development and unit testing. However, the individual critical situations being modelled were rather small and self-contained, and it is uncertain whether this way of working would scale to larger and more complex problem spaces.

The guidance that the participants got out of the patterns appears to be of two types. To begin with, to the extent that patterns provided reasonable solutions to difficult to model problems, the pattern solutions were used as archetypes for own solutions on the whiteboard. This was the most common usage of patterns observed. In the second case, patterns were used to verify the correctness of modelling, by ensuring that the developed solution was consistent with the patterns selected:

Participant B: *“Is a vehicle an agent?”*

Participant A: *“Let’s check the pattern!”*

The latter usage was observed both on the whiteboard and later on when attempting to formalise results into OWL files on a computer. In usage, the selected patterns were seen as optimal solutions to problems, and no reflections on the suitability of the patterns in question were observed. On the contrary, in some situations the participants attempted to realign their

⁴<http://www.corej2eepatterns.com/>

solutions to available design patterns even when this needlessly significantly increased the complexity of their solution. One example of this is the observed use of the *AgentRole* pattern in categorising different types of staff, which in the scope of the problem could just as easily have been done via subsumption.

During modelling there were occasions when the work process slowed down, and the participants got caught up in discussions on how to define some very fundamental concepts such as situation, time, event, etc. When questioned, participants expressed a strong preference for such foundational concepts being available as patterns. While a few such foundational patterns have been extracted from top-level ontologies and made available in the community portal [8], their documentation is at the time of writing limited.

Effects of ODP Usage

Across the two days of working, a noticeable improvement in modelling speed among the participants could be observed. Tasks that in the morning took an hour to complete were in the afternoon performed in fifteen-twenty minutes. While this learning effect cannot be solely attributed to pattern use, the participants indicated that a certain efficiency gain is certainly due to them:

“I think it was helpful, it makes it clearer and furthers reuse, saving time.” – Participant A

This efficiency gain was most pronounced when the participants reused patterns which they had already tried once or twice on other problems. The participants also indicated that in order to get the most out of the design patterns, a practitioner needs to have developed some degree of familiarity with them:

“For me it’s a new type of modelling [...] but it’s understandable, and I can imagine if you know patterns, you are quite faster at inventing everything.” – Participant C

As has been mentioned in Section 6.1.3, the effects of ODP use on the process and resulting ontologies were not all beneficial. In some cases, over-dependence on patterns complicated the resulting ontologies needlessly, and misunderstanding of pattern documentation led to generally strange results. An example of the latter is the modelling of the characteristic “loudness”, where the resulting model had time being loudness-indexed rather than the other way around. On the whole however these problems were minor compared to the observed and perceived benefits of ODP usage in guiding modelling.

6.2 Learnability and Usability Evaluations

In this study, two structural indicators from the initial model were selected for study, namely *Anonymous class count* and *Class to property ratio*. These indicators share the characteristics that they are easy to formalise, measure, and vary as needed for the sake of the study, and they also per the initial model affect the usability related quality characteristics. Additionally, two indicators dealing with ODP documentation, *Example illustration count* and *Documentation minimalism*, believed to affect ODP learnability, were selected for study. Like the structural indicators, these indicators are easy to vary as needed through the use of adjustable templates for displaying ODP documentation. Finally, as the study design was being developed, the opportunity to also query participants about the perceived learnability effects of a few more structural features of ODPs was also realised and taken. The use of *Property domain restrictions* and *Property range restrictions* were selected for such study, based on the fact that these features are naturally used in most ontology modelling situations, meaning that the study participants are very likely to come across them and form an opinion and understanding about them. Accordingly, the following research questions were established for the study:

1. How do the indicators *Example illustration count* and *Documentation minimalism* affect ODP learnability?
2. How do the indicators *Anonymous class count* and *Class to property ratio* affect ODP usability?
3. How do *Property domain restrictions* and *Property range restrictions* affect ODP learnability?

In order to answer the above questions, a study consisting of both surveys and timed tasks, was set up, as described in Section 4.2.4. Based on the existing initial ODP quality model, the following hypotheses were established:

- Patterns *including examples* (i.e., “Scenarios” as expressed in ODP portal) with *illustrations* are superior to ones having examples simply written in text, in terms of learnability.
- Patterns displaying *documentation minimalism* (i.e., a limitation of the fields displayed per pattern to a sane minimum) are superior to patterns not displaying this feature, in terms of learnability.
- Patterns that contain a *high number of anonymous class definitions* (i.e., restrictions) are more difficult to apply than patterns that contain low numbers of such definitions.
- Patterns that have a *high ratio of classes to properties* are easier to apply than patterns that display a low such ratio.

- *Property domain restrictions* and *Property range restrictions* in ODPs are beneficial to the learnability of said ODPs.

The patterns used in this evaluation and provided to the participants are listed in Table 6.3. They were selected from all non-trivial (i.e., larger than 10 classes including imports) and general purpose patterns patterns in the `ontologydesignpatterns.org` repository, and were chosen so as to provide variation across the two structural indicators under study, as indicated in the table. Of all non-trivially sized ODPs in the ODP portal, the mean anonymous class count was 23, and the mean Class to Property ratio was 0.5, with a variation between 0 and 56 for the former value and between 0.33 and 0.83 for the latter.

Table 6.3: ODPs used in learnability and usability evaluations.

ODP name	C/P ratio	C/P group	AC count	AC group
Basic Plan	0,4	Low	33	High
Communication Event	0,55	High	41	High
Reaction	0,45	Low	16	Low
Invoice	0,65	High	0	Low

6.2.1 Results

Survey 1

The most interesting results from the first survey, regarding documentation-related effects on learnability, are summarised in Tables 6.4, 6.5, and 6.6. In all of these tables, the participant responses have been averaged, grouped by the indicator under study, such that the responses given by participants seeing the patterns exhibiting documentation minimalism can easily be contrasted against those given by participants seeing non-minimal patterns (and likewise for example illustrations).

Table 6.4: Survey 1: competency question recognition correctness ratio.

Group	Com. Event	Reaction	Invoice	Basic Plan
Minimal	70 %	89 %	88 %	90 %
Non-minimal	54 %	80 %	100 %	80 %
Illustrated	63 %	75 %	100 %	75 %
Non-illustrated	53 %	91 %	90 %	91 %

Table 6.4 indicates how well participants were able to match a pattern to five given competency questions. The values in the given table are averaged across the participant/pattern pairs for each group of ODPs. Of interest here is that for three out of four patterns (Basic Plan, Communication Event, and Invoice), the group seeing documentation minimal patterns scored higher than the non-minimal group. The presence of an illustrated example does not seem to correspond to any increase in correct responses.

Table 6.5: Survey 1: class recognition correctness ratio.

Group	Com. Event	Reaction	Invoice	Basic Plan
Minimal	50 %	57 %	80 %	67 %
Non-minimal	29 %	50 %	100 %	60 %
Illustrated	50 %	25 %	80 %	75 %
Non-illustrated	0 %	71 %	100 %	57 %

Table 6.5 indicates how well participants were able to match a term in a given scenario description text to a class name in the ODPs. Interesting to note here is that, just as in the previous table, for three out of four patterns, the participants seeing documentation minimal variants scored better than those seeing non-minimal ones. Presence or absence of illustrated examples do not correlate with any such results.

Table 6.6: Survey 1: time required to answer questions.

Group	Com. Event	Reaction	Invoice	Basic Plan
Minimal	14,75	7	4	13
Non-minimal	11,29	10	8,83	13
Illustrated	13,13	7,75	8,4	11,75
Non-illustrated	11	8,29	5,17	13,71

Table 6.6 averages the time (in minutes) taken by participants to answer the questions underlying the above tables. Again, we see a slight advantage for the documentation minimal variant group in two cases, when compared with the non-minimal variant group, and no effect of illustrated examples.

Tasks

Table 6.7: Task times (in minutes) and learnability metrics.

ODP name	Time taken	Concreteness	Difficulty
Communication Event	57	2.55	3.27
Reaction	67.54	2.55	2.82
Invoice	47.82	3.55	2.3
Basic Plan	66	2.91	2.73

The times required to model the provided scenarios using the given pattern are detailed in Table 6.7, along with the average participant responses for concreteness/abstraction and difficulty of understanding from the first survey. The latter two measures were taken using five point scales, e.g. ranging between “very easy” (a score of 1) and “very difficult”, (a score of 5). When cross-referencing this table against the ODP characterisations in Table 6.3, we can see that the two patterns having the lowest class to property ratio (Communication Event, Invoice), are the ones for which the associated modelling exercise took least time to complete. We can also see from the concreteness and difficulty of understanding measures that the Invoice

pattern was considered by far the most concrete and easiest to understand, whereas Communication Event was considered quite difficult to understand and quite abstract.

Survey 2

Table 6.8 shows the most interesting results of the second survey on how the participants perceived indicators and their effects on the usability and learnability of the patterns in question. The indicators queried about (abbreviated in the table) were the presence of example illustration in pattern documentations, the restrictions on ranges or domains of properties asserted in the patterns, and the existence of class restriction axioms in the pattern (the latter being the main source of anonymous class definitions in an ontology). For the first three indicators, participants were questioned how they found the presence of the related features helped or harmed in *understanding* the patterns, whereas for the last indicator, they were questioned how they found the presence of this feature helped or harmed in *using* the patterns.

Table 6.8: Perceived usability and learnability effects of indicators from survey 2.

Effect	Illustrations	Range res.	Domain res.	Class res.
Very helpful	50 %	25 %	17 %	17 %
Helpful	42 %	67 %	50 %	58 %
Neither	0 %	0 %	8 %	0 %
Harmful	0 %	8 %	8 %	17 %
Very harmful	8 %	0 %	0 %	0 %
No opinion	0 %	0 %	17 %	8 %

6.2.2 Findings

The data gathered gives some indications regarding the possible validity of the original hypotheses, though it does not in its own right clearly prove or disprove any one of them.

The hypothesis that *patterns including examples with illustrations are superior to ones having examples simply written in text in terms of learnability* is partially supported by the observation that a large majority of study participants themselves rank the presence of illustrated examples as *Helpful* or *Very helpful* (Table 6.8) with regards to understanding how to use a pattern. However, the data collected on how quickly and how well they were actually able to learn the pattern (Tables 6.4, 6.5, and 6.6) give no such support to the hypothesis.

The hypothesis that *patterns displaying documentation minimalism are superior to patterns not displaying this feature, in terms of learnability*, is partially supported by the observation that for three different measures (competency question recognition, class recognition, and time required to respond to survey one), the participant groups seeing pattern variants display-

ing documentation minimalism performed better than participant groups seeing non-minimal variants. However, the small number of participants and patterns tested severely limits the validity of the figures presented, so this should rather be considered an indication than evidence.

That *patterns that contain a high number of anonymous class definitions are more difficult to apply than patterns that contain low numbers of such definitions* is not supported by the gathered data. In fact, the data from survey 2 presented in Table 6.8 indicates that the opposite may be more correct, and that anonymous class definitions improve usability. As that table shows, a large majority of study participants found the presence of class restrictions (which result in anonymous classes) in an ODP to be *Helpful* or *Very helpful* in terms of usability. The question on class restriction effects was also associated with an open question where several participants expressed how they found these restrictions helpful in ascertaining the purpose of a class in the case that labels, comments, and the subsumption hierarchy were not sufficient for this.

The hypothesis that *patterns that have a high ratio of classes to properties are easier to apply than patterns that display a low such ratio* is partially supported by the data exhibited in Tables 6.7 and 6.3, which shows that the two patterns displaying the highest such ratio are the ones that were fastest to complete modelling tasks with. However, for at least one of the patterns (Invoice), it is quite possible that other characteristics of the pattern displayed in the former table (its ease of use and concrete nature) affect the resulting time more than the class to property ratio, so the support for this hypothesis is rather weak, and more study of it would be required.

The hypothesis that *Property domain restrictions and property range restrictions in ODPs are beneficial to the learnability of said ODPs* is supported by the results shown in Table 6.8. As illustrated there, a clear majority of the participants express that domain and range restrictions on properties are *Helpful* or *Very helpful* in terms of usability of patterns. Associated with the questions on these indicators were open questions, where participants indicated the reason for these responses being that restrictions helped clarify the intended usage of properties. As the presence of such restrictions are likely to have constraining effects on the reusability of patterns, it is an important issue to study whether the usability gains given by their presence could instead be achieved by other means, for instance improved pattern documentation pages or RDFS labelling and comments.

In addition to the original hypotheses, an unforeseen effect relating to indicator effects was observed that is relevant to the ODP quality model development. As seen in Table 6.7, participant-reported values for concreteness and difficulty of use seem to be inversely related. While the issue of the abstraction/concreteness of patterns has been avoided in the initial quality model (largely due to difficulties in finding proper measurement methods for such indicators), this finding implies that more study on the topic is warranted. Another unexpected observation along similar lines concerns

the fact that study participants rated the Communication Event pattern as more abstract than the Basic Plan one. While Communication Event is rather complex and contains many classes, it is in the author's opinion actually an example of a quite concrete ODP. Clearly more work on establishing how to measure and understand the abstraction or concreteness levels of ODPs is required.

6.3 Performance Indicator Evaluation

For the purpose of evaluating and refining performance-related indicators from the initial quality model, a literature survey and a study of existing indicator variance in published ODPs was performed, as detailed in Section 4.2.5. To reiterate, the following research questions were employed in the evaluation:

1. Which of the proposed performance-related effects of indicators from the initial ODP quality model are supported by existing literature?
2. Which indicators from literature known to affect the performance of reasoning with ontologies are also applicable to ODPs?
3. How do performance-altering indicators vary across published ODPs?

The following sections detail the results of this evaluation work.

6.3.1 Literature Study

In the studied papers, three main types of indicators and corresponding effects could be identified, namely expressivity profile indicators (i.e., indicators related to profiles or constraints of ontology language structures available for use), inheritance hierarchy structural indicators (i.e., indicators related to the structure of the subsumption tree), and axiom usage indicators (i.e., general indicators related to the logical axioms employed in an ontology). Each of these categories and the indicators found to be associated with them are discussed in the following subsections.

Profile indicators

In the initial ODP quality model it was hypothesised that the use of a more expressive description logic language would be detrimental to reasoning performance, based on the fact that a more expressive language allows for a much larger set of possible inferences to be drawn. This idea is supported by the findings of several papers in the studied set, e.g., by Urbani et al. [125, 126] and Horridge et al. [127], not to mention the recent recommendations by the W3C specifying OWL 2 Profiles [128].

Urbani et al. discuss the issue of scaling out description logic reasoning on parallel computing clusters using the MapReduce framework. They

show in [125] that materialising the closure of an RDF graph using RDFS semantics can be performed using MapReduce, due to certain characteristics of the RSFS semantics. As shown in [126], the increased expressivity of OWL means that implementing such parallelisable scalable reasoning over datasets based on OWL ontologies is significantly more difficult than when using RDFS. OWL, when expressed as transformation rules, allows for multiple instance triples in the antecedent, and it is impossible to define an ordering of rules whereby each rule need only fire once, which greatly complicates the distribution of an OWL-based dataset across the nodes of a MapReduce cluster. By limiting themselves to the OWL Horst fragment of OWL, the authors manage to work around these issues and present a resulting solution (WebPIE) that enables reasoning with OWL Horst significantly faster than previous solutions. Even so, the solution is many times slower on real data than the simpler RDFS reasoning from [125], and adapting it for higher levels of expressivity (i.e., OWL DL) would (if at all possible) most likely increase execution time further.

In [127] Horridge et al. analyse the characteristics of the three OWL 2 profiles, OWL 2 RL, OWL 2 EL, and OWL 2 QL, and study the adherence to these profiles among ODPs published on the Web. The three profiles are subsets of OWL 2 intended for particular usages. By limiting the semantics used, both in terms of actual axioms allowed and the positioning and use of those axioms, computational properties suitable to different uses (reasoning over many classes and properties, query answering over large sets of instance data, or reasoning using predictably performant rule-based systems). Horridge et al. [127] find that relatively few ODPs fit in these profiles, and that this may in part be due to modelling practices and recommendations (e.g., to always declare an inverse for an object property, or the use of cardinality restrictions where existential restrictions could be used instead).

The initially proposed quality model only declared one indicator pertaining to language expressivity (DL Complexity, I-9), where a higher expressivity was hypothesised to be more detrimental to performance than a lower one for all types of tasks. The above indicates that it would be useful to instead consider the problem from a perspective of multiple indicators contributing to different types of performance-related quality characteristics (corresponding to the aforementioned profiles and use cases).

Structural indicators

The initially proposed quality model presents a number of hypotheses regarding the performance impact of indicators relating to the subsumption hierarchy structure of ODPs. The studied literature adds support to at least one of these hypotheses, the effect of tangledness, but it also supports the addition of a performance-altering effect to the previously defined usability-related indicator on subsumption hierarchy depth, as well as new anonymous class-aware versions of existing structural indicators.

Kang et al. [129] perform a thorough evaluation of the effects of a number of different ontology metrics on performance in different commonly used reasoners. While most of their observations are on effects of axiomatic indicators (as discussed in the following section), one interesting finding concerns the subsumption hierarchy. Kang et al. find that the indicator that they denote *tree impurity* has a measurable impact on reasoner performance, such that a high degree of tree impurity in an ontology correlates to slower reasoning over that same ontology. This tree impurity metric measures how far the ontology’s inheritance hierarchy deviates from being a tree, by calculating how many more `owl:subClassOf` axioms are present in the ontology than are needed to structure a pure tree. This is simply a different way of measuring the same indicator that in the initial quality model of this thesis is denoted *tangledness*. By showing that tree impurity contributes to reduced computational efficiency, Kang et al. [129] thereby also validate that tangledness contribute to lowering the same quality characteristic.

In [130], LePendou et al. study the characteristics of both ontologies and data in the biomedicine domain, with a goal towards improving performance by way of optimisation of data and data structure as opposed to purchasing more hardware. For this purpose, a number of synthetic ontologies and data sets were generated and classified. One of the metrics studied, and found to have a high impact on materialisation performance, is the depth of the subsumption hierarchy. The explanation given for this is that for every asserted instance of a subclass, all of the logic axioms pertaining to each and every superclass must also be calculated. For a shallow ontology, this may be a matter of just a few classes before the top level class is reached, and not a lot of work. For a deeper ontology however, this may be a quite significant amount of entailments that need to be computed. Kang et al. [129] also study the depth indicator, and like LePendou et al. find that it contributes to slower reasoning performance.

The majority of metrics whose effects are studied by Kang et al. [129] are first defined by Zhang et al. in [131]. This paper holds an interesting reflection on the importance of including anonymous classes when computing values for structural indicators affecting reasoning performance, since these anonymous classes are obviously reasoned over just as named classes are. However, anonymous classes are likely to be less important when considering for instance usability-related effects of those same structural indicators. Consequentially, both anonymous class-aware and non anonymous class-aware variants of such indicators would need to be included in the ODP quality model.

Axiomatic indicators

The majority of performance-affecting indicators discussed and tested in the studied literature concerns the use of particular types of axioms or structures in an ontology. Two papers in particular contribute to this knowledge,

namely Goncalves et al. [132] and the aforementioned work by Kang et al. [129].

Goncalves et al. [132] investigate performance variability in ontologies, and details a developed method for isolating performance-degrading sections of ontologies, by the authors denoted “hot spots”, for different reasoners. In the paper some methods for approximate reasoning over ontologies where such hot spots have been removed are presented and compared. The use of such methods may partially mitigate the loss in information caused by the hot spot removal. The removal of hot spots were found to cut reasoning times by between 81 and 99 %. As a side effect of their work, the authors notice that the removal of hotspots correlate with the removal of General Concept Inclusions, GCIs, from the ontologies. GCIs are subclass or equivalency axioms that have a complex class expression on their right hand side, for instance `(HeartDisease and hasLocation some HeartValve)SubClassOf: CriticalDisease`. While this effect was particularly obvious for the HermiT reasoner, where every single GCI was found within a hot spot, it also occurred to a lesser degree within other studied reasoners. These results suggest that the number of GCI axioms in an ontology, and ODP, are useful as indicators of reasoning performance, particularly for the HermiT reasoner.

As mentioned above, [129] evaluate performance effects of a number of metrics (most of which are presented by Zhang et al. in [131]). They find four indicators that show a measurable performance-altering effect and that can be applied to ODPs also:

- Existential quantifications – the number of existential quantification axioms in an ontology or ODP. This is easiest measured by counting the number of `ObjectSomeValuesFrom` axioms in the ontology.
- Cyclomatic complexity – inspired by the same metric as used in software engineering complexity calculations, this indicator measures the number of linearly independent paths through the RDF graph, including not only subclass relations but any directed edges, which a reasoner needs to traverse in classifying said graph.
- Class in-degree – the average number of incoming edges to classes in the ontology. This gives an indication as to how interconnected an ontology or ODP is.
- Class out-degree – the inverse of the above indicator, i.e., the average number of outgoing edges to classes in the ontology.

6.3.2 Indicator Variance in ODP Repositories

The results of the study of indicator variance (the setup of which is introduced in Section 4.2.5) are detailed below, with the exception of a few indicators that were not included, namely cyclomatic complexity, and the

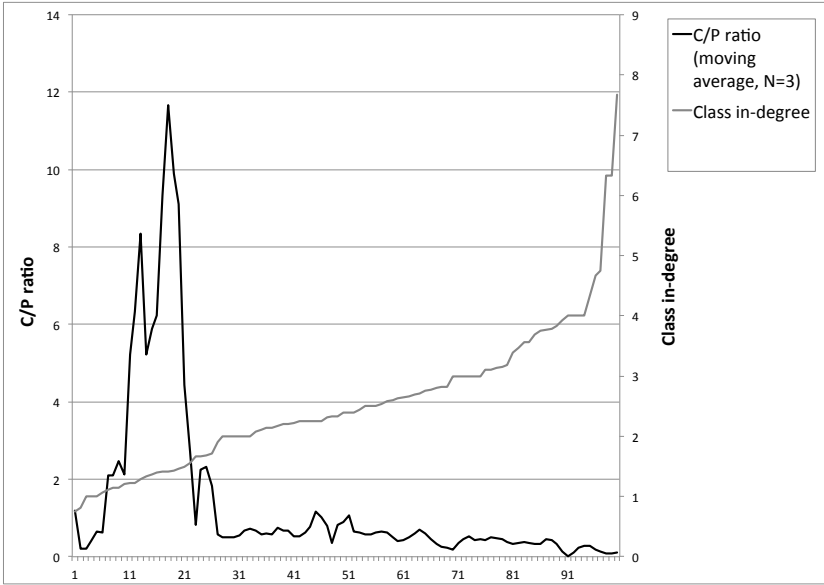


Figure 6.1: Class in-degree and Class to property ratio distributions

set of OWL profile adherence indicators. It proved practically infeasible to develop software for reliably measuring the former, and rather than make assertions based on possibly inexact data, it has been left out of this analysis. However, as it was proven by Kang et al. [129] to be of importance for reasoner performance, it is considered relevant for inclusion in the ODP quality model, despite our current lack of knowledge regarding its variance among published patterns. The latter set of indicators has as mentioned above been discussed extensively in Horridge et al. [127], to which the interested reader is referred. The findings in [127] clearly support the interpretation of these indicators as important for ODP developers to be aware of and keep in mind in ODP work, a fact which in turn motivates the inclusion of said indicators in the ODP quality model.

Average class in-degree

The values for the average class in-degree indicator vary between 0.75 and 8, with a median value of 2.39 and an average value of 2.6. The distribution of indicator values over the whole pattern set is shown in Figure 6.1. As illustrated in the figure, the large majority of patterns (93 %) have a class in-degree of less than four, whereas a small group of patterns differ quite significantly and have an average value of around six.

Upon comparing some of the patterns exhibiting high and low values for the average class in-degree indicator, it was observed that they tended

to differ in terms of the number of object properties contained within the patterns. The patterns exhibiting a high level of class in-degree seemed to contain a larger number of object properties than those patterns displaying a low level of this indicator. To test whether this held for the entirety of the pattern set, the values of the class to property ratio indicator from the initial ODP quality model were mapped against the values of the class in-degree indicator. The inverse of the former being a size-wise normalised measure of the number of properties in the ontology, such a mapping should if the observation holds indicate the existence of an inverse correlation between the two mapped indicator value series.

The results, as shown in Figure 6.1, while not indicating a clear inverse correlation of the indicators across the entire studied pattern set, does indeed indicate that the patterns displaying highest class in-degree have a lower class to property ratio (i.e., contain more properties per class as posited above), and that many of the patterns displaying low class-in degree have a higher than average class to property ratio (i.e., contain fewer properties).

One possible explanation for this observation is the existence of domain and range definitions on the many object properties in patterns with high average class in-degree. It is generally considered good practice to establish such definitions for properties one adds to an ontology. However, each such domain or range definition gives rise to one incoming RDF edge to the class in question, raising the average class in-degree indicator. Based on this observation, a recommendation to the effect of limiting the number of domain and range definitions used in performance-dependent ontologies can be made, and the initial ODP quality model was revised accordingly. However, there may also be other as yet unknown causes beside domain and range definitions that that give rise to high average class in-degrees, and given the observation on variability in this indicator in the observed patterns, including the indicator itself it in a revised ODP quality model is also warranted.

Average class out-degree

The values for the average class out-degree indicator vary between 1 and 3.83, with a median value of 2.75 and an average of 2.64. The distribution of indicator values over the whole pattern set is shown in Figure 6.2. The reason why all patterns exhibit a value of at least one is simply that all defined classes by definition have at least one outgoing `subClassOf` edge to another class.

In studying some patterns displaying low or high values, it was observed that the patterns displaying a higher value seem to be patterns in which class restrictions are used extensively. Class restrictions are written as the class being asserted to be either a subclass of or equivalent to a restriction axiom, which would explain this observation – each `subClassOf` or `equivalentClass` axiom adds an outgoing edge, increasing the value of the indicator.

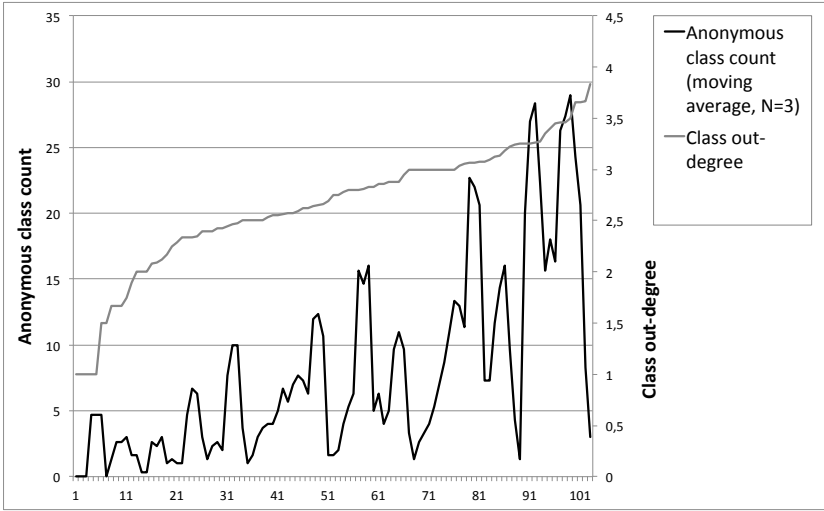


Figure 6.2: Class out-degree and Anonymous class count distributions

To test whether this explanation is supported by further evidence, the number of anonymous class definitions (i.e., restrictions) were plotted against the value of the class out-degree indicator. The results are presented in Figure 6.2 which indicates a possible correlation between class out-degree and anonymous class count.

Since the number of class restrictions has in earlier chapters been shown to be helpful in guiding users of an ODP, this unexpected performance-related effect of using such restrictions is of particular interest. Also, given the variation of this indicator's values over the studied ODPs, inclusion of the indicator in the ODP quality model is well motivated.

Depth of inheritance

As mentioned in Section 6.3.1, it can be important to measure both asserted and inferred versions of structural indicators. Due to the difficulty of measuring the inferred indicators across the transitive import closure graph of an ODP using the tools and APIs available at the time of writing, the values below were only calculated over the asserted depths of patterns, excluding imports. Moreover, as even this is quite a difficult task (due to different practices on how subclass relations to the top-level `owl:Thing` class are modelled), certain simplifications had to be made. These simplifications include the assumption of a subclass relation to `Thing` if no other superclass is asserted within the particular OWL file⁵.

⁵To study the code used to calculate the depth metrics, the reader is referred to <https://github.com/hammar/OntoStats/tree/master/plugins-structural>

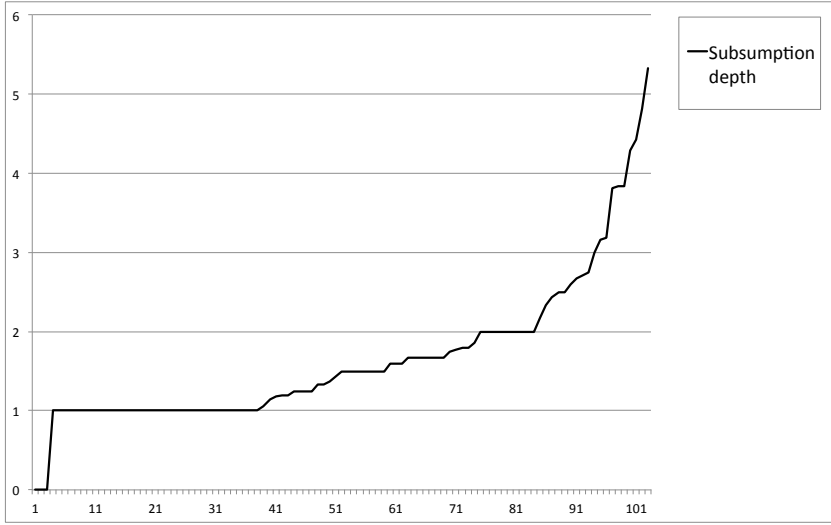


Figure 6.3: Subsumption depth indicator variance

The subsumption hierarchy depth of the patterns varies from 0 to 5.3, with a median value of 1.5 and an average value of 1.7. In other words, most of the patterns are not very deep. The distribution of values across the patterns studied are displayed in Figure 6.3. At the bottom end of the spectrum is a fairly large group (38 of 103 patterns) that have a depth of one or less. In studying this particularly shallow group, it appears to consist of two types of patterns. The first type consist of simpler domain specific vocabularies that do not employ much expressive logics, but rather act as schemas for simple datatypes that may be reused. Examples include patterns for species habitats, invoices, etc. The second type consist of very general patterns that define abstract or intangible phenomena without going into specific details. Examples include patterns modelling phenomena like participation and situation. A large part of the latter group seems to result from refactoring of top-level ontologies like DOLCE, whereas many of the patterns in the former group seem to be developed for more concrete and applied purposes.

The patterns from the <http://odp.sourceforge.net> repository are generally deeper (with an average depth of 3.29) than those from the <http://ontologydesignpatterns.org> portal. However, the latter patterns generally contain more example classes that would likely be removed before instantiation in real cases, reducing this difference.

The large variation in depth displayed indicates that this is an indicator which is suitable to include in the ODP quality model.

Existential quantification count

About half the patterns, 51 of 103, contain no explicit existential quantification axioms. If cardinality restrictions are rewritten into semantically equivalent existential restrictions as suggested in [127], the number of patterns containing no existential quantification axioms drops to 43. Of the 60 patterns that contain such axioms half, 31, contain one or two existential quantification axioms each. Studying a number of such patterns it was observed that the axioms are used sparingly and only when required.

However, in studying the patterns that contained a higher number of existential quantification axioms (i.e., three or more, as seen in 29 of the patterns), it was observed that these axioms were sometimes used in seemingly unneeded ways. For instance, subclasses restating such axioms as were already asserted on their superclasses, and existential quantification used to assert the coexistence of two individuals where it seems one individual might well exist on its own. These observed suboptimal uses of computationally expensive existential quantification axioms clearly motivates the inclusion of the indicator in an ODP quality model - it seems ODP developers need to be reminded not to take the use of this type of axioms lightly.

General concept inclusion count

General concept inclusions were not displayed by any of the studied ODPs. The author believes that this is for two reasons: firstly, because such constructs are generally not supported by ontology engineering tools such as Protégé (other than via workarounds), and secondly, because they add little new in terms of logical expressivity. Equivalent logic can be written in simpler ways by using other methods. Whatever the reason, the complete lack of any patterns displaying this indicator implies that it would be unnecessary to include it in a quality model for such Ontology Design Patterns, and it is consequently not included in the resulting model.

Tree Impurity / Tangledness

Due to the mentioned inconsistencies in how subclass relations to the `owl:Thing` concept are modelled, the tree impurity indicator is difficult to measure in a reliable manner without first modifying the studied ontologies. However, this indicator actually measures the same thing as the tangledness indicator presented in Section 5.4, i.e., how far an ontology inheritance hierarchy deviates from being a tree with one parent class per subclass. Therefore, observations regarding tangledness variation in the dataset carry over to the tree impurity indicator also.

Of the 103 studied patterns, only three display any degree of asserted tangledness at all. In all three of these cases, the number of multi-parent classes in the pattern was one. It appears that the use of asserted multiple inheritance in ODPs is rare. However, it should be noted that the number

of *inferred* multi-parent classes may be significantly greater than this number. While inferred tangledness has for technical reasons been infeasible to measure in this study, its effect on the performance of reasoning may be considerable, for which reason it is kept in the ODP quality model.

6.3.3 Results

Table 6.9 summarises the observations of performance-altering indicators discussed in the previous sections. Included in the table is a column indicating which indicators from the initial quality model that are supported or adapted, and a column with references to the source of the observation. In the following chapter, Chapter 7, the updated quality model including the indicators tested and developed in this chapter is presented, including measurement methods and scales associated with these new indicators.

Table 6.9: Performance related indicator summary.

Name	Existing indicator	Sources
Anonymous class count	Anonymous class count (I-3)	Section 6.3.2
Average class in-degree		[129, 131]
Average class out-degree		[129, 131]
Cyclomatic complexity		[129]
Depth of inheritance	Subsumption hierarchy depth (I-24)	[129, 130, 131]
Existential quantification count		[129]
OWL 2 EL adherence	DL Complexity (I-9)	[127, 128]
OWL 2 QL adherence	DL Complexity (I-9)	[127, 128]
OWL 2 RL adherence	DL Complexity (I-9)	[127, 128]
OWL Horst adherence	DL Complexity (I-9)	[125, 126]
Property domain restrictions ratio	Property domain restrictions ratio (I-18)	Section 6.3.2
Property range restrictions ratio	Property range restrictions ratio (I-19)	Section 6.3.2
Tree impurity	Tangledness (I-25)	[129, 131]

Chapter 7

Refined ODP Quality Model

The following chapter presents an updated version of the ODP quality model, based on the initial version presented in Chapter 5, and revised according to the evaluation findings presented in Chapter 6.

7.1 Metamodel

The refined ODP quality model adheres to the quality metamodel developed in Section 5.1, as shown in Figure 7.1. The concepts included in this metamodel remain unchanged since the initial quality model, and are described in some detail in Section 5.1, for which reason they are not repeated here.

7.2 Quality Characteristics

The initial quality model presented in Section 5.4 included five top level quality characteristics, and twenty sub-characteristics. In the development and evaluation of indicators for the quality model described in this thesis, most of the initial quality characteristics could be associated with one or more indicators. Of those few that could not, the majority remain strong candidates for such linkage given more development of suitable ODP quality indicators. For instance, the present lack of indicators for *Testability* does not imply that the testability of ODPs is unimportant, but only that more work is required in order to find suitable methods of measuring testability.

From the initial quality model, two quality characteristics were found to be unsuitable and replaced. The two sub-characteristics *Time behaviour efficiency* and *Resource utilisation efficiency* were found to be redundant, in that they simply express two different aspects of their parent quality characteristic *Resulting performance efficiency*. The quality characteristics of

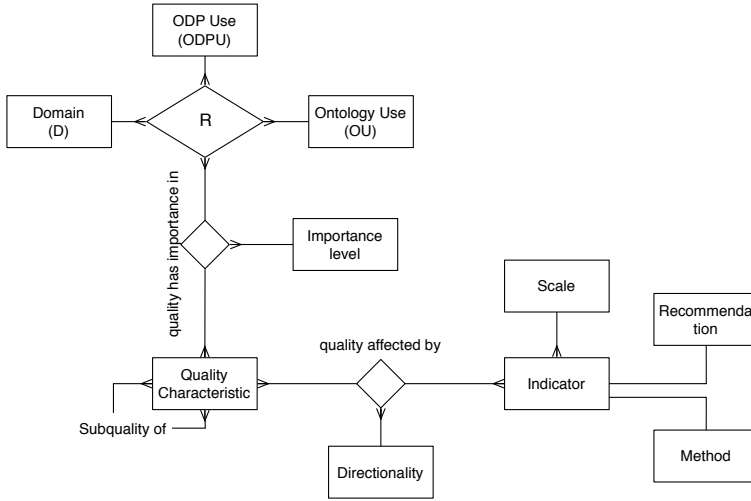


Figure 7.1: Quality Metamodel

the revised quality model are listed below. The quality characteristics that are associated with some indicators which have been evaluated in Chapter 6 are marked with an exclamation point, the ones associated with only unevaluated indicators are marked with an asterisk, and the ones for which no indicators have yet been developed are unmarked.

Functional Suitability (!)

Degree to which an ODP meets stated or implied needs.

- *Functional completeness* * – Degree to which the ODP meets expressed knowledge modelling requirements (i.e., competency questions and other design requirements).
- *Functional appropriateness* – Degree to which the ODP facilitates simple storage and retrieval of knowledge formalised according to its definitions (e.g., does the ODP require simple or complex SPARQL queries to retrieve knowledge).
- *Consistency* – Degree to which the ODP is internally logically consistent.
- *Accuracy* (*) – Degree to which the ODP accurately represents the real world domain being modelled (e.g., whether it adheres to established industry standards and protocols, or legislation).

Resulting performance efficiency (!)

Reasoner or system performance efficiency over ontologies created using the pattern.

Usability (!)

Degree to which an ODP can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction.

- *Appropriateness recognisability (!)* – Degree to which users can recognise whether an ODP is appropriate for their needs.
- *Learnability (!)* – Degree to which an ODPs structure, and intended usage can be learned by users new to it, such that they can thereafter apply the ODP successfully and efficiently.
- *Operability (*)* – Degree to which an ODP has attributes that make it easy to apply and use.
- *User error protection (!)* – Degree to which an ODP prevents users from making modelling errors.
- *User interface aesthetics* – Degree to which the ODP’s documentation (text, graphics, etc.) is pleasing for the user.
- *Accessibility* – Degree to which the ODP’s documentation can be used by people with the widest range of characteristics and capabilities.

Maintainability

Degree of effectiveness and efficiency with which an ODP (and consequently, ontologies built using that ODP) can be adapted and modified by maintainers after deployment in some usage scenario.

- *Modularity* – Degree to which the ODP is composed of discrete components such that a change to one component has minimal impact on other components.
- *Analysability (*)* – Degree of effectiveness and efficiency with which it is possible to assess the impact on an ODP of an intended change to one or more of its parts, or to diagnose an ODP for deficiencies or causes of failures, or to identify parts to be modified.
- *Modifiability (*)* – Degree to which an ODP can be effectively and efficiently modified without introducing defects or degrading existing ODP quality.
- *Testability* – Degree of effectiveness and efficiency with which test criteria can be established for an ODP and tests can be performed to determine whether those criteria have been met.

- *Stability* – Perceived change expectation on the ODP - high stability denotes a low degree of change is expected, and vice versa.

Compatibility (*)

Degree to which an ODP can successfully be reused and integrated with other ODPs or IT artefacts in the construction of ontologies or systems.

- *Reusability (!)* – Degree to which an ODP can be used in more than one system, or in building other assets.
- *Co-existence* – Degree to which an ODP can coexist with other ODPs as modules in an ontology, i.e., without detrimental impact on other ODP modules.
- *Interoperability* – Degree to which two or more ODPs share definitions of co-occurring concepts.

7.3 Indicators and Effects

Tables 7.1 and 7.2 list the updated set of indicators and corresponding effects on quality characteristics in the refined ODP quality model. The former table holds those indicators whose effects have been evaluated, either in this thesis, or in some other known empirical evaluation. In this table, the third column references said evaluation. The latter table holds those indicators whose effects have been hypothesised in the initial quality model presented in Section 5.4, but which remain to be evaluated empirically.

Two indicators have been removed from the initial quality model due to being redundant or supplanted by updated variants. The first of these, Direct import count (I-8) can be understood as a specialisation of Transitive import count (I-27). Given that the effects on ODP quality characteristics associated with import count all concern the size and complexity of the transitive import closure, measuring only the direct import count is unnecessary. The second removed indicator is DL Complexity (I-9). In the refined ODP model this indicator has been replaced by OWL 2 profile and OWL Horst adherence indicators (I-35 through I-38) which measures similar description language aspects and effects on quality, but builds on established work in the field.

Indicators that are new to or updated in the refined quality model compared to the initial quality model from Section 5.4 are listed in more detail below, including associated measurement methods, scales, and in some cases recommendations. Those indicators that are included in Tables 7.1 and 7.2 but not listed below are unchanged since the initial quality model. The directionality of indicator effects (*positive* or *negative*) is interpreted in the same manner as in the initial quality model, detailed in Section 5.4.2.

Table 7.1: Evaluated indicator effects of the refined quality model.

Nr	Indicator	Quality characteristic affected	Evaluation
I-29	Abstraction level	Usability	Section 6.2.2
I-1	Accompanying text description	Appropriateness recognisability	Section 6.1.3
I-3	Anonymous class count	Usability	Section 6.2.2
I-3	Anonymous class count	Resulting performance efficiency	Section 6.3.2
I-30	Average class in-degree	Resulting performance efficiency	[129, 131]
I-31	Average class out-degree	Resulting performance efficiency	[129, 131]
I-6	Class to property ratio	Usability	Section 6.2.2
I-32	Common pitfalls description	User error protection	Section 6.1.3
I-7	Competency question count	Appropriateness recognisability	Section 6.1.3
I-33	Cycomatic complexity	Resulting performance efficiency	[129, 131]
I-10	Documentation minimalism	Learnability	Section 6.2.2
I-12	Example illustration count	Learnability	Section 6.2.2
I-34	Existential quantification count	Resulting performance efficiency	[129]
I-35	OWL 2 EL Adherence	Resulting performance efficiency	[127, 128]
I-36	OWL 2 QL Adherence	Resulting performance efficiency	[127, 128]
I-37	OWL 2 RL Adherence	Resulting performance efficiency	[127, 128]
I-38	OWL Horst Adherence	Resulting performance efficiency	[125, 126]
I-18	Property domain restrictions	Learnability	Section 6.2.2
I-18	Property domain restrictions	Resulting performance efficiency	Section 6.3.2
I-19	Property range restrictions	Learnability	Section 6.2.2
I-19	Property range restrictions	Resulting performance efficiency	Section 6.3.2
I-21	Size	Learnability	Section 6.1.3
I-22	Structure illustration	Appropriateness recognisability	Section 6.1.3
I-24	Subsumption hierarchy depth	Resulting performance efficiency	[129, 130, 131]
I-25	Tangledness	Resulting performance efficiency	[129, 131]
I-26	Terminological cycle count	Resulting performance efficiency	[100]
I-27	Transitive import count	Reusability	Section 6.1.3
I-27	Transitive import count	Usability	Section 6.1.3

7.3.1 Updated Indicators

I-1 Accompanying text description

Method: Check that the ODP OWL file is associated with a textual description document or webpage.

Scale: Nominal (boolean)

Affects characteristics (evaluated): Appropriateness recognisability (positively)

I-3 Anonymous class count

Method: Count the cardinality of the set of anonymous classes in the associated OWL file.

Scale: Ratio

Recommendation: Anonymous classes that result from restrictions on classes are helpful in terms of learnability and usability and their use is recommended if these are prioritised qualities. Patterns that display high number of anonymous classes not associated with such restrictions are not known to have these effects and are not recommended.

Affects characteristics (evaluated): Usability (positively), Resulting performance efficiency (negatively)

Table 7.2: Unevaluated indicator effects of the refined quality model.

Nr	Indicator	Quality characteristic affected
I-2	Annotation ratio	Compatibility
I-2	Annotation ratio	Maintainability
I-2	Annotation ratio	Usability
I-4	Axiom/class ratio	Analysability
I-5	Class disjointness ratio	Resulting performance efficiency
I-7	Competency question count	Learnability
I-11	Example text count	Appropriateness recognisability
I-11	Example text count	Learnability
I-13	Functionality questionnaire time	Learnability
I-14	Minimalism	Compatibility
I-14	Minimalism	Learnability
I-14	Minimalism	Operability
I-15	Modification task time	Modifiability
I-16	Nary relation count	Resulting performance efficiency
I-16	Nary relation count	Usability
I-17	OntoClean adherence	Functional suitability
I-18	Property domain restrictions	Reusability
I-19	Property range restrictions	Reusability
I-20	Redundant axiom count	Resulting performance efficiency
I-20	Redundant axiom count	Usability
I-21	Size	Analysability
I-22	Structure illustration	Learnability
I-23	Subsumption hierarchy breadth	Usability
I-24	Subsumption hierarchy depth	Usability
I-25	Tangledness	Compatibility
I-25	Tangledness	Usability
I-27	Transitive import count	Resulting performance efficiency
I-28	User evaluation ranking	Usability

I-7 Competency question count

Method: Divide the number of competency questions expressed in the pattern documentation by the size (I-21) of the ODP.

Scale: Ratio

Recommendation: Competency questions should be viewed as requirements on the ODP, such that any functionality of the ODP that is not explicitly touched upon by at least one competency question should be considered non-essential.

Affects characteristics (evaluated): Appropriateness recognisability (positively)

Affects characteristics (unevaluated): Learnability (positively)

I-18 Property domain restrictions ratio

Method: Divide the cardinality of the set of properties that have defined domain restrictions with the cardinality of the set of all properties in the ODP.

Scale: Ratio

Recommendation: If the ODP is to be used in a scenario where pattern learnability is important, use domain restrictions extensively to document property and class behaviour. If the ODP is to be used in a scenario where performance is important, avoid the use of such restrictions.

Affects characteristics (evaluated): Learnability (positively), Resulting performance efficiency (negatively)

Affects characteristics (unevaluated): Reusability (negatively)

I-19 Property range restrictions ratio

Method: Divide the cardinality of the set of properties that have defined range restrictions with the cardinality of the set of all properties in the ODP.

Scale: Ratio

Recommendation: If the ODP is to be used in a scenario where pattern learnability is important, use range restrictions extensively to document property and class behaviour. If the ODP is to be used in a scenario where performance is important, avoid the use of such restrictions.

Affects characteristics (evaluated): Learnability (positively), Resulting performance efficiency (negatively)

Affects characteristics (unevaluated): Reusability (negatively)

I-22 Structure illustration

Method: Assert that the ODP documentation includes at least one illustration of the classes and properties proposed by the pattern and how they relate.

Scale: Nominal (boolean)

Affects characteristics (evaluated): Appropriateness recognisability (positively)

Affects characteristics (unevaluated): Learnability (positively)

I-24 Subsumption hierarchy depth

Method: Define an ancestor path as a path through the asserted subsumption hierarchy linking a leaf node concept to the top-level concept `owl:Thing`. Define the depth of an ancestor path as the cardinality of that path. The average depth of the ODP is then the sum of all depths in the ODP divided by the cardinality of the set of ancestor paths.

Scale: Ratio

Affects characteristics (evaluated): Resulting performance efficiency (negative)

Affects characteristics (unevaluated): Usability (directionality unknown)

I-25 Tangledness

Method: Classify the ODP using a reasoner. Then divide the cardinality of the set of named classes which have more than one named superclass by the cardinality of the set of all classes in the ODP.

Scale: Ratio

Affects characteristics (evaluated): Resulting performance efficiency

(negatively)

Affects characteristics (unevaluated): Compatibility (negatively), Usability (negatively)

I-27 Transitive import count

Method: Calculate the cardinality of the set of OWL files found through a recursive search over the import hierarchy of the original reusable OWL building block associated with the ODP.

Scale: Ratio

Recommendation: Be aware of the semantics of the OWL imports function, i.e., that it necessitates accepting as true the whole import closure within your context. Consequently, avoid importing ontologies that have a large import closure, or that are not strictly necessary for functionality.

Affects characteristics (evaluated): Reusability (negatively), Usability (directionality unknown)

Affects characteristics (unevaluated): Resulting performance efficiency (negatively)

7.3.2 New Indicators

I-29 Abstraction level

Method: Provide a representative group of ODP users with a survey over the set of patterns for which this indicator is to be measured, querying them for their opinion on the abstraction level of each pattern, from high to low. Ensure that the users are given sufficient time to study and apply the patterns in test scenarios before answering the survey. Providing the users with multiple patterns allows them to see a variation of patterns that makes it easier for them to answer confidently (especially if they lack prior experience of ODPs). As a benchmark of survey reliability (a sort of gold standard), include in the studied set a number of patterns with a priori known abstraction levels.

Scale: Ordinal

Affects characteristics (evaluated): Usability (negatively)

I-30 Average class in-degree

Method: Calculate the average number of ingoing RDF edges that OWL classes in the ODP have.

Scale: Ratio

Affects characteristics (evaluated): Resulting performance efficiency (negatively)

I-31 Average class out-degree

Method: Calculate the average number of outgoing RDF edges OWL classes in the ODP have.

Scale: Ratio

Affects characteristics (evaluated): Resulting performance efficiency (negatively)

I-32 Common pitfalls description

Method: Assert that the ODP documentation contains a description of common usage mistakes for said ODP.

Scale: Nominal (boolean)

Affects characteristics (evaluated): User error protection (positively)

I-33 Cyclomatic complexity

Method: Calculate the cyclomatic complexity of the ODP graph, per the following formula: $CYC = \#E - \#N + 2 * cc$, where $\#E$ is the number of edges in the RDF graph, $\#N$ is the number of nodes, and cc is the number of strongly connected components.

Scale: Ratio

Affects characteristics (evaluated): Resulting performance efficiency (negatively)

I-34 Existential quantification count

Method: Count the cardinality of the set of existential quantification axioms in the ODP.

Scale: Ratio

Recommendation: While limiting the number of existential quantification axioms is helpful for increasing performance, it is not recommended to do this via translation of said axioms into semantically equivalent cardinality axioms with a limit of one, as this may put the ODP outside of an easily computable OWL 2 profile.

Affects characteristics (evaluated): Resulting performance efficiency (negatively)

I-35 OWL 2 EL Adherence

Method: Assert that the ODP uses only such axioms which are allowed under the OWL 2 EL profile.

Scale: Nominal (boolean)

Recommendation: The EL profile is developed for efficient reasoning over ontologies containing many classes or properties. If the ODP is to be applied in the construction of such an ontology, compliance with this profile is important.

Affects characteristics (evaluated): Resulting performance efficiency (positively)

I-36 OWL 2 QL Adherence

Method: Assert that the ODP uses only such axioms which are allowed under the OWL 2 QL profile.

Scale: Nominal (boolean)

Recommendation: The QL profile is developed for efficient query answering over ontologies or knowledge bases containing large amounts of instance data. If the ODP is to be applied in the construction of an ontology used for such a purpose, compliance with this profile is important.

Affects characteristics (evaluated): Resulting performance efficiency (positively)

I-37 OWL 2 RL Adherence

Method: Assert that the ODP uses only such axioms which are allowed under the OWL 2 RL profile.

Scale: Nominal (boolean)

Recommendation: The RL profile is developed for efficient reasoning using traditional rule engine based technologies. If the ODP is to be applied in the construction of an ontology for use with such technologies, compliance with this profile is important.

Affects characteristics (evaluated): Resulting performance efficiency (positively)

I-38 OWL Horst Adherence

Method: Assert that the ODP uses only such axioms which are allowed under the OWL Horst dialect of OWL.

Scale: Nominal (boolean)

Recommendation: OWL Horst can scale out over a MapReduce-based computation cluster. If the ODP is to be used in the construction of an ontology used with simpler reasoning over very large amounts of data, compliance with OWL Horst is important.

Affects characteristics (evaluated): Resulting performance efficiency (positively)

Chapter 8

Conclusions and Future Work

The following chapter revisits the initial motivation for this thesis project, presents the contributions of the work, discusses how the performed work has helped answer the research questions, and suggests areas of continued study regarding the quality of Ontology Design Patterns.

8.1 Summary of Contributions

The problems associated with traditional ontology engineering are introduced and discussed in Chapter 1 and Chapter 2. The core of the problem is that the complexity of the ontology representation languages and tools necessitate ontology engineers having extensive experiences of knowledge modelling issues and formal logics, in addition to the domain for which the ontology is developed. This slows the uptake of semantic technologies in non-academic settings, where these technologies are perceived to be difficult to learn and apply, and where consequently, relatively simple vocabularies are used instead. Ontology Design Patterns are proposed to alleviate this problem by providing non-experts with reusable building blocks, supporting them in developing ontologies and employing semantic technologies. However, as shown in Section 2.5, there is no established understanding on how such patterns are best developed (or as it may be, found, isolated, or generated), what the guiding principles for such development should be, or what qualities patterns should display in order to be helpful and usable for different purposes. Consequently the development and use of patterns by non-academics remain limited.

This thesis project was motivated by the need for developing an understanding of Ontology Design Pattern quality, such that these patterns can more easily be developed and used by practitioners. To this end, an On-

tology Design Pattern quality model was developed. The model has two purposes. As a research artefact, it has proven valuable in structuring the phenomena and effects under study, and to help devise case studies and experiments. As a project deliverable, it packages and provides the developed understanding of ODP quality, in a format accessible to practitioners.

In Chapter 1 a list of the expected contributions of this thesis project was presented. That list is repeated here, with comments:

- *A conceptual understanding of quality, as it relates to Ontology Design Patterns.*

The ODP Quality Metamodel, presented in Section 5.1, provides a perspective on how to break apart the quality concept in an Ontology Design Pattern context. Its design was influenced by the results of the MAPPER project [67], which helped provide a conceptual understanding of quality in modelling. An important attribute of this understanding of ODP quality is the difference it makes between quality characteristics and indicators, where the former represent perspectives on quality which are not directly measurable, and the latter are quantifiable and measurable properties that contribute to said quality characteristics. Another important attribute of the quality metamodel is how it defines the importance of quality characteristics as being context-bounded; that is, quality characteristics are not in and of themselves said to be of a certain importance in the general case, but depending on the priorities of the ontology engineering situation they may be more or less important. This ODP Quality Metamodel can be used as a starting point for researchers discussing ODP quality, and further populated with developed or discovered quality characteristics and indicators in the future.

- *A set of Ontology Design Pattern quality characteristics, capturing the different relevant perspectives on ODP quality.*

Section 7.2 presents the ODP quality model set of quality characteristics, with definitions. These quality characteristics were developed by reusing existing work in neighbouring fields, as described in Sections 5.2.1 and 5.2.2. Key influences in this process were the ISO 25010 [70] software quality standard, and the PhD thesis “*On the Quality of Feature Models*” by Christer Thörn [69]. Both practitioners and researchers can make use of these quality characteristics; the former using them to prioritise requirements on constituent ODPs in ontology engineering projects involving patterns, and the latter using them as a framework to connect studied or developed indicators.

- *Indicators and methods for quantifying and measuring ODP quality characteristics.*

Section 7.3 presents the ODP quality model set of indicators, used to measure the above mentioned quality characteristics. These indicators

were first developed by reusing existing work in neighbouring fields, as described in Sections 5.2.3, 5.2.4, and 5.3. Several of them were then evaluated and developed further as described in Chapter 6. These indicators may be used by practitioners in evaluating and selecting between pattern candidates suitable for use in ontology engineering projects, based on the effect that the indicators have on the wanted quality characteristics.

- *Recommendations on suitable values for said indicators, or aspects to consider when measuring them.*

The indicators presented in Section 7.3 are in many cases associated with concrete recommendations for ontology engineers to consider. These recommendations are based on observations and experiences from the evaluation of ODP quality in Chapter 6. They provide further guidance to practitioners on reasonable indicator values given the intended ODP or ontology usage.

The resulting ODP quality model is presented in full in Chapter 7. It is the main contribution of this thesis, and provides answers to the initially posed research questions, as shown in Section 8.2.

8.2 Research Questions Revisited

Revisiting the research questions established in Chapter 1, those were:

1. Which quality characteristics of Content Ontology Design Patterns can be differentiated, and through what indicators can they be measured and observed?
2. How do the quality characteristics of Content Ontology Design Patterns interact and affect one another?

The first question is answered through the developed and partially evaluated quality model presented in Chapter 7. The quality model includes a set of quality characteristics believed, and in some cases also observed, to be relevant and useful in ontology engineering scenarios where Ontology Design Patterns are used. This listing of quality characteristics is an answer to the first half of the first question. The model also includes a set of measurable indicators in Section 7.3, complete with methods and scales, contributing to these quality characteristics. This listing of indicators provide an answer to the second half of the first research question.

The answer to the second question can be inferred from Table 7.1, where it is observed that some indicators are found to have effects on multiple quality characteristics. For instance, the number of anonymous classes affect the usability of a pattern positively, and the performance efficiency of the same pattern negatively. Similar conflicting effects can also be seen in

the indicators on the ratio of properties that have domain and range restrictions set. This implies that in an ontology engineering situation a choice needs to be made regarding the tradeoff between the usability and learnability quality characteristics on the one hand, and the performance efficiency quality characteristic on the other. This result is in line with the observation made in [127] that some established learnability-related good practices in pattern design (such as always declaring inverse properties) are incompatible with the OWL 2 EL profile and consequently significantly reduce the performance efficiency of ontologies. Pattern users and designers need to be aware of this tradeoff in order to select the patterns that are most suitable to their use cases. Other tradeoffs for developers to keep in mind concern the reusability and usability of ODPs that employ a high degree of import statements, which have shown to be both potentially helpful in validating design choices, and restrictive in terms of lowering reuse potential of ODPs, as discussed in Section 6.1.3.

It is important to note that neither of the two research questions have been exhaustively answered. There are several quality characteristics and indicators that have support in literature but which have not yet been evaluated and tested. In particular, quality characteristics and indicators relating to *Maintainability* and *Compatibility* are in need of more study, as such quality characteristics are likely to have impact on ontology maintenance costs, which could be a hindrance for the adoption of Semantic Web ontologies in enterprises. There are also most likely more indicator effects waiting to be found through further empirical evaluation of the model, effects which could interact in ways requiring more tradeoff choices to be made.

8.3 Future Work

The work presented in this thesis has answered the original research questions, but it has also opened up possibilities for further exploration and research. As has been touched upon in Section 8.2, the ODP quality model presented in this work can be strengthened in several ways, allowing the original research questions to be more exhaustively answered. The effects described in the quality model are generally based on observations from one or two cases, or experiments, each, and in order to increase the validity of the model, more extensive evaluation of these effects would be helpful. Also, several quality characteristics and indicators have not yet been evaluated, and are supported only by theory. Evaluating the effects of these indicators and the need for these quality characteristics would also strengthen the ODP quality model.

The work performed in this thesis project has illustrated the need for another research question to be studied, namely *Which quality characteristics of Content Ontology Design Patterns affect the suitability of the resulting ontologies for different uses?* As shown in the previous section it is clear that the use case in which an ODP is applied matters greatly in terms of

which qualities and consequently indicators should be prioritised. Therefore, validating the way in which the ODP quality metamodel defines an ontology use case, as well as investigating which typical use cases exist and which quality characteristics that support them, would be highly beneficial to pattern users. Closely related to this is the question of how user competence and background affect modelling performance and results in different cases.

While the Ontology Design Pattern quality model is intended to be helpful for practitioners on its own, it could benefit users more if matched by guidelines on how to best apply the model in practice. Such guidelines would cover both selection of ODPs based on prioritised quality characteristics, and the development of ODPs displaying such qualities as required for particular use cases. Developing such a set of guidelines is a natural next step in development of the ODP quality model.

Finally, there are some more practical developments in terms of tooling and methods that could be beneficial to the research community if developed after the end of this thesis project. Firstly, the OntoStats tool developed for measuring ODP indicators in the evaluation of performance-related indicators provides a rather simple way for measuring various metrics of ontologies, Ontology Design Patterns, and serialised knowledge bases either individually or in bulk. The development of more plugins for this tool could potentially help other researchers avoid reinventing the wheel. Secondly, the knowledge fusion case study resulted in several suggestions for improvement to the ODP portal, which certainly merit attention. These include an improved search engine, a better and more task-oriented pattern classification system, and clearer visualisation of pattern interdependencies.

Bibliography

- [1] K. Sandkuhl. Information Logistics in Networked Organizations: Selected Concepts and Applications. In *Enterprise Information Systems: 9th International Conference, ICEIS 2007, Funchal, Madeira, June 12-16, 2007, Revised Selected Papers*, volume 12, page 43. Springer Verlag, 2008.
- [2] R. Studer, V. R. Benjamins, and D. Fensel. Knowledge Engineering: Principles and methods. *Data & knowledge Engineering*, 25(1):161–197, 1998.
- [3] V. Tarasov and M. Lundqvist. Modeling Collaborative Design Competence with Ontologies. *International Journal of e-Collaboration (IJeC)*, 3(4):46–62, 2007.
- [4] K. Sandkuhl and A. Billig. Ontology-based artefact management in automotive electronics. *International Journal of Computer Integrated Manufacturing*, 20(7):627–638, 2007.
- [5] A. Smirnov, M. Pashkin, N. Chilov, and T. Levashova. Knowledge logistics in information grid environment. *Future Generation Computer Systems*, 20(1):61–79, 2004.
- [6] E. Blomqvist and K. Sandkuhl. Patterns in Ontology Engineering: Classification of Ontology Patterns. In *ICEIS 2005: proceedings of the Seventh International Conference on Enterprise Information Systems, Miami, USA, May 25-28, 2005*, 2005.
- [7] A. Gangemi. Ontology Design Patterns for Semantic Web Content. In *The Semantic Web-ISWC 2005*, pages 262–276. Springer, 2005.
- [8] V. Presutti, A. Gangemi, S. David, G. Aguado de Cea, M. C. Suárez-Figueroa, E. Montiel-Ponsoda, and M. Poveda. D2.5.1: A Library of Ontology Design Patterns: Reusable Solutions for Collaborative Design of Networked Ontologies. Technical report, NeOn Project, 2007.

- [9] E. Blomqvist, A. Gangemi, and V. Presutti. Experiments on Pattern-based Ontology Design. In *Proceedings of the Fifth International Conference on Knowledge Capture*, pages 41–48. ACM, 2009.
- [10] K. Hammar and K. 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.
- [11] R.L. Ackoff. From Data to Wisdom. *Journal of Applied Systems Analysis*, 16:3–9, 1989.
- [12] G. Bellinger, D. Castro, and A. Mills. Data, Information, Knowledge, and Wisdom. <http://www.systems-thinking.org/dikw/dikw.htm>, checked on: 2012-09-28, 2004.
- [13] H. Tsoukas and E. Vladimirou. What is organizational knowledge? *Journal of Management Studies*, 38(7):973–993, 2001.
- [14] R. D. Stacey. *Complex responsive processes in organizations: Learning and knowledge creation*. Psychology Press, 2001.
- [15] A. Newell. The knowledge level: Presidential address. *AI Magazine*, 2(2):1, 1981.
- [16] F. Baader. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge: Cambridge University Press, 2003.
- [17] T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, pages 199–220, 1993.
- [18] J. Z. Pan. Resource Description Framework. In *Handbook on Ontologies*, pages 71–90. Springer, 2009.
- [19] G. Antoniou and F. Van Harmelen. Web Ontology Language: OWL. In *Handbook on Ontologies*, pages 91–110. Springer, 2009.
- [20] S. Staab and R. Studer, editors. *Handbook on Ontologies*. Springer, 2009.
- [21] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):28–37, 2001.
- [22] T. Berners-Lee. Semantic Web on XML. Talk held at XML 2000 conference in Washington DC, slides published at <http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html>, 2000.
- [23] G. Klyne and J. J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. Technical report, W3C, 2004.

-
- [24] D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. Technical report, W3C, 2004.
 - [25] L.W. Lacy. *OWL: Representing Information Using the Web Ontology Language*. Trafford Publishing, 2005.
 - [26] D.L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. Technical report, W3C, 2004.
 - [27] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview (Second Edition). Technical report, W3C, 2012.
 - [28] N. Guarino, editor. *Formal Ontology in Information Systems: Proceedings of the First International Conference (FOIS'98)*. Ios Press Inc, 1998.
 - [29] T. Berners-Lee. Linked Data. <http://www.w3.org/DesignIssues/LinkedData.html>, checked on: 2012-10-10, June 2009.
 - [30] L. Ding, D. DiFranzo, S. Magidson, DL McGuinness, and J. Hendler. The Data-gov Wiki: A Semantic Web Portal for Linked Government Data. In *Proceedings of the 6th International Conference on Knowledge Capture*, 2009.
 - [31] L. Ding, D. DiFranzo, A. Graves, J.R. Michaelis, X. Li, D.L. McGuinness, and J. Hendler. Data-gov Wiki: Towards Linking Government Data. In *2010 AAAI Spring Symposium Series*, 2010.
 - [32] R. Guha, R. McCool, and E. Miller. Semantic Search. In *Proceedings of the 12th International Conference on World Wide Web*, pages 700–709. ACM, 2003.
 - [33] V. Uren, Y. Lei, V. Lopez, H. Liu, E. Motta, and M. Giordanino. The usability of semantic search tools: a review. *The Knowledge Engineering Review*, 22(04):361–377, 2007.
 - [34] A. Kiryakov, B. Popov, I. Terziev, D. Manov, and D. Ognyanoff. Semantic annotation, indexing, and retrieval. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2(1):49–79, 2004.
 - [35] Y. Lei, V. Uren, and E. Motta. SemSearch: A Search Engine for the Semantic Web. In *Managing Knowledge in a World of Networks*, pages 238–245. Springer, 2006.
 - [36] R. Bhagdev, S. Chapman, F. Ciravegna, V. Lanfranchi, and D. Petrelli. Hybrid Search: Effectively Combining Keywords and Semantic Searches. In *The Semantic Web: Research and Applications*, pages 554–568. Springer, 2008.

- [37] R. Bhagdev, A. Chakravarthy, S. Chapman, F. Ciravegna, and V. Lanfranchi. Creating and Using Organisational Semantic Webs in Large Networked Organisations. In *The Semantic Web - ISWC 2008*, pages 723–736. Springer, 2008.
- [38] D.C. Luckham and B. Frasca. Complex Event Processing in Distributed Systems. Technical report, Stanford University, 1998.
- [39] W. Yao, C.H. Chu, and Z. Li. Leveraging complex event processing for smart hospitals using RFID. *Journal of Network and Computer Applications*, 2010.
- [40] G. Hermosillo, L. Seinturier, and L. Duchien. Using Complex Event Processing for Dynamic Business Process Adaptation. In *Proceedings of the 7th IEEE 2010 International Conference on Services Computing (SCC 2010)*, pages 466–473. IEEE, 2010.
- [41] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic. Stream reasoning and complex event processing in ETALIS. *Semantic Web*, 2011.
- [42] D.F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. C-SPARQL: SPARQL for Continuous Querying. In *Proceedings of the 18th International Conference on World Wide Web*, pages 1061–1062. ACM, 2009.
- [43] D. Barbieri, D. Braga, S. Ceri, E. Della Valle, Y. Huang, V. Tresp, A. Rettinger, and H. Wermser. Deductive and Inductive Stream Reasoning for Semantic Social Media Analytics. *Intelligent Systems, IEEE*, PP: 99:1–1, 2010.
- [44] D. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. Incremental Reasoning on Streams and Rich Background Knowledge. In *The Semantic Web: Research and Applications*, pages 1–15. Springer, 2010.
- [45] V. Tarasov. Ontology-based Approach to Competence Profile Management. *Journal of Universal Computer Science*, 18(20):2893–2919, 2012.
- [46] A. Billig, E. Blomqvist, and F. Lin. Semantic Matching Based on Enterprise Ontologies. In *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, pages 1161–1168. Springer, 2007.
- [47] A. Ranganathan and R. H. Campbell. A Middleware for Context-Aware Agents in Ubiquitous Computing Environments. In *Middleware 2003*, pages 143–161. Springer-Verlag New York, Inc., 2003.

-
- [48] H. Chen, T. Finin, and A. Joshi. An ontology for context-aware pervasive computing environments. *The Knowledge Engineering Review*, 18(03):197–207, 2003.
 - [49] M. Gruninger and M. S. Fox. The Role of Competency Questions in Enterprise Engineering. In *Proceedings of the IFIP WG5*, volume 7, pages 212–221, 1994.
 - [50] M. Fernández-López, A. Gómez-Pérez, and N. Juristo. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. Technical report, American Association for Artificial Intelligence, 1997.
 - [51] Y. Sure, S. Staab, and R. Studer. On-To-Knowledge Methodology (OTKM). In *Handbook on Ontologies*. Springer, 2003.
 - [52] S. Pinto, S. Staab, Y. Sure, and C. Tempich. OntoEdit Empowering SWAP: a Case Study in Supporting DIstributed, Loosely-Controlled and evolvInG Engineering of oNTologies (DILIGENT). In *The Semantic Web: Research and Applications*, pages 16–30. Springer, 2004.
 - [53] H. S. Pinto, S. Staab, C. Tempich, and Y. Sure. Distributed Engineering of Ontologies (DILIGENT). In *Semantic Web and Peer-to-Peer*, pages 303–322. Springer, 2006.
 - [54] N. Noy and D. L. McGuinness. Ontology Development 101. Technical report, Knowledge Systems Laboratory, Stanford University, 2001.
 - [55] E. Blomqvist. *Semi-automatic Ontology Construction based on Patterns*. PhD thesis, Linköping University, 2009.
 - [56] A. Gangemi and V. Presutti. Ontology Design Patterns. In *Handbook on Ontologies*, pages 221–243. Springer, 2009.
 - [57] E. Daga, E. Blomqvist, A. Gangemi, E. Montiel, N. Nikitina, V. Presutti, and B. Villazon-Terrazas. D2.5.2: Pattern based ontology design: methodology and software support. Technical report, NeOn Project, 2007.
 - [58] E. Blomqvist, K. Sandkuhl, F. Scharffe, and V. Svatek, editors. *WOP 2009: Proceedings of the Workshop on Ontology Patterns*. CEUR Workshop Proceedings, 2009.
 - [59] E. Blomqvist, V.K. Chaudhri, O. Corcho, V. Presutti, and K. Sandkuhl, editors. *WOP 2010: Proceedings of the 2nd International Workshop on Ontology Patterns*. CEUR Workshop Proceedings, 2010.
 - [60] E. Blomqvist, A. Gangemi, K. Hammar, and M. C. Suárez-Figueroa, editors. *WOP 2012: Proceedings of the 3rd Workshop on Ontology Patterns*. CEUR Workshop Proceedings, 2012.

- [61] M. Dzbor, M. C. Suárez-Figueroa, E. Blomqvist, H. Lewen, M. Espinoza, A. Gómez-Pérez, and R. Palma. D5.6.2 Experimentation and Evaluation of the NeOn Methodology. Technical report, NeOn Project, 2007.
- [62] F. Van Harmelen, A. Ten Teije, and H. Wache. Knowledge Engineering Rediscovered: Towards Reasoning Patterns for the Semantic Web. In *The Fifth International Conference on Knowledge Capture*, pages 81–88, 2009.
- [63] M. A. Hearst. Automatic Acquisition of Hyponyms from Large Text Corpora. In *Proceedings of the 14th Conference on Computational Linguistics - Volume 2*, pages 539–545. Association for Computational Linguistics, 1992.
- [64] V. Presutti, E. Daga, A. Gangemi, and E. Blomqvist. eXtreme Design with Content Ontology Design Patterns. In *Proceedings of the Workshop on Ontology Patterns (WOP 2009), collocated with International Semantic Web Conference (ISWC 2009)*, page 83, 2009.
- [65] V. Presutti, E. Blomqvist, E. Daga, and A. Gangemi. Pattern-Based Ontology Design. In *Ontology Engineering in a Networked World*, pages 35–64. Springer, 2012.
- [66] M. Ivarsson and T. Gorschek. Technology transfer decision support in requirements engineering research: a systematic review of REj. *Requirements Engineering*, 14(3):155–175, 2009.
- [67] K. Sandkuhl, H. Tellioglu, and S. Johnsen. Orchestrating Economic, Socio-Technical and Technical Validation using Visual Modelling. In *16th European Conference on Information Systems: ECIS 2008*, 2008.
- [68] G. M. Campagnolo, G. Jacucci, S. G. Johnsen, O-W. Rahlff, K. Sandkuhl, H. Tellioglu, and I. Wagner. MAPPER Deliverable D3 - Framework for Validation of Economic, Socio-Technical and Technical Viewpoints. Technical report, MAPPER Consortium, 2006.
- [69] C. Thörn. *On the Quality of Feature Models*. PhD thesis, Department of Computer and Information Science, Linköpings Universitet, 2010.
- [70] ISO. ISO/IEC 25010: Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. Technical report, International Organization for Standardization, 2011.
- [71] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing Cardinality-based Feature Models and their Specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.

-
- [72] J. Sun, H. Zhang, Y. Fang, and L.H. Wang. Formal Semantics and Verification for Feature Modeling. In *Proceedings of the 10th International Conference on Engineering of Complex Computer Systems (ICECCS 2005)*, pages 303–312. IEEE, 2005.
 - [73] M. Eriksson, J. Börstler, and K. Borg. Software product line modeling made practical. *Communications of the ACM*, 49(12):49–54, 2006.
 - [74] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch. COVAMOF: A Framework for Modeling Variability in Software Product Families. In *Software Product Lines*, pages 25–27. Springer, 2004.
 - [75] D. Batory, C. Johnson, B. MacDonald, and D. Von Heeder. Achieving Extensibility Through Product-Lines and Domain-Specific Languages: A Case Study. In *Software Reuse: Advances in Software Reusability*, pages 83–153. Springer, 2000.
 - [76] M. Genero, G. Poels, and M. Piattini. Defining and Validating Measures for Conceptual Data Model Quality. In *Advanced Information Systems Engineering*, pages 724–727. Springer, 2006.
 - [77] D. Moody and G. Shanks. What Makes a Good Data Model? Evaluating the Quality of Entity Relationship Models. In *Entity-Relationship Approach – ER '94 Business Modelling and Re-Engineering*, pages 94–111. Springer, 1994.
 - [78] O.I. Lindland, G. Sindre, and A. Solvberg. Understanding Quality in Conceptual Modeling. *Software, IEEE*, 11(2):42–49, 1994.
 - [79] R. Watts. *Measuring software quality*. NCC Publications, 1987.
 - [80] J. McCall, P. Richards, and G. Walters. *Factors in software quality*. National Technical Information Service, 1977.
 - [81] K. Beck, R. Crocker, G. Meszaros, J. Vlissides, J. O. Coplien, L. Dominick, and F. Paulisch. Industrial Experience with Design Patterns. In *Proceedings of the 18th International Conference on Software Engineering*, pages 103–114. IEEE Computer Society, 1996.
 - [82] G. Masuda, N. Sakamoto, and K. Ushijima. Applying Design Patterns to Decision Tree Learning System. In *ACM SIGSOFT Software Engineering Notes*, volume 23, pages 111–120. ACM, 1998.
 - [83] S. Chen and B. W. Morris. Using Design Patterns, Analysis Pattern, and Case-Based Reasoning to Improve Information Modeling and Method Engineering in Systems Development. *International Journal of Applied Management and Technology*, 7(1):7, 2009.

- [84] L. Prechelt, B. Unger-Lamprecht, M. Philippsen, and W. F. Tichy. Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance. *Software Engineering, IEEE Transactions on*, 28(6):595–606, 2002.
- [85] N-L. Hsueh, P-H. Chu, and W. Chu. A quantitative approach for evaluating the quality of design patterns. *Journal of Systems and Software*, 81(8):1430–1439, 2008.
- [86] J. Bansiya and C. G. Davis. A hierarchical model for object-oriented design quality assessment. *Software Engineering, IEEE Transactions on*, 28(1):4–17, 2002.
- [87] A. Gómez-Pérez, M Fernandez-Lopez, and O Corcho. *Ontological Engineering*. Springer-Verlag, London, Berlin, 2004.
- [88] A. Gangemi, C. Catenacci, M. Ciaramita, and J. Lehmann. Modelling Ontology Evaluation and Validation. In *The Semantic Web: Research and Applications*, pages 140–154. Springer, 2006.
- [89] A. Gangemi, C. Catenacci, M. Ciaramita, J. Lehmann, R. Gil, F. Bolici, and O. Strignano. Ontology evaluation and validation. Technical report, Laboratory for Applied Ontology, ISTC-CNR, 2005.
- [90] A. Lozano-Tello and A. Gómez-Pérez. ONTOMETRIC: A Method to Choose the Appropriate Ontology. *Journal of Database Management*, 2(15):1–18, 2004.
- [91] T. L. Saaty. A scaling method for priorities in hierarchical structures. *Journal of Mathematical Psychology*, 15(3):234–281, 1977.
- [92] C. Welty and N. Guarino. Supporting ontological analysis of taxonomic relationships. *Data & Knowledge Engineering*, 39(1):51–74, 2001.
- [93] N. Guarino and C. Welty. Evaluating Ontological Decisions with OntoClean. *Communications of the ACM*, 45(2):61–65, 2002.
- [94] N. Guarino and C. A. Welty. An Overview of OntoClean. In *Handbook on Ontologies*, pages 201–220. Springer, 2009.
- [95] P. Spyns, R. Meersman, and M. Jarrar. Data modelling versus Ontology engineering. *ACM SIGMOD Record*, 31(4):12–17, 2002.
- [96] A. Gangemi, N. Guarino, C. Masolo, and A. Oltramari. Sweetening WordNet with DOLCE. *AI magazine*, 24(3):13, 2003.
- [97] M. Doerr, J. Hunter, and C. Lagoze. Towards a Core Ontology for Information Integration. *Journal of Digital Information*, 4(1), 2006.

-
- [98] M. Hepp. GoodRelations: An Ontology for Describing Products and Services Offers on the Web. In *Knowledge Engineering: Practice and Patterns*, pages 329–346. Springer, 2008.
 - [99] M. Fernández-López and A. Gómez-Pérez. The integration of OntoClean in WebODE. In *Proceedings of the EON2002 Workshop at 13th EKAW*, 2002.
 - [100] L. Lefort, K. Taylor, and D. Ratcliffe. Towards Scalable Ontology Engineering Patterns: Lessons Learned from an Experiment based on W3C’s Part-whole Guidelines. In *Proceedings of the Second Australasian Workshop on Advances in Ontologies*, pages 31–40. Australian Computer Society, Inc., 2006.
 - [101] S. Lodhi and Z. Ahmed. Content Ontology Design Pattern Presentation. Master’s thesis, Jönköping University, 2011.
 - [102] R. L. Glass, V. Ramesh, and I. Vessey. An Analysis of Research in Computing Disciplines. *Communications of the ACM*, 47(6):89–94, 2004.
 - [103] V. R. Basili. The Role of Experimentation in Software Engineering: Past, Current, and Future. In *Proceedings of the 18th International Conference on Software Engineering*, pages 442–449. IEEE Computer Society, 1996.
 - [104] D. I-K. Sjøberg, T. Dybå, B. C-D. Anda, and J. E. Hannay. Building Theories in Software Engineering. In *Guide to Advanced Empirical Software Engineering*, pages 312–336. Springer, 2008.
 - [105] R. B. Johnson and A. J. Onwuegbuzie. Mixed Methods Research: A Research Paradigm Whose Time Has Come. *Educational Researcher*, 33(7):14–26, 2004.
 - [106] G. Morgan and L. Smircich. The Case for Qualitative Research. *Academy of Management Review*, pages 491–500, 1980.
 - [107] R. L. Daft. Learning the Craft of Organizational Research. *Academy of Management Review*, pages 539–546, 1983.
 - [108] L. P. Ruddin. You Can Generalize Stupid! Social Scientists, Bent Flyvbjerg, and Case Study Methodology. *Qualitative Inquiry*, 12(4):797–812, 2006.
 - [109] P. Palvia, D. Leary, E. Mao, V. Midha, P. Pinjani, and A.F. Salam. Research Methodologies in MIS: An Update. *Communications of the Association for Information Systems*, 14:526–542, 2004.

- [110] F. Shull and R. L. Feldmann. Building Theories from Multiple Evidence Sources. In *Guide to Advanced Empirical Software Engineering*, pages 337–364. Springer, 2008.
- [111] B. Kitchenham. Procedures for Performing Systematic Reviews. Technical report, Keele University, 2004.
- [112] S. Schneider, R. Torkar, and T. Gorschek. Solutions in global software engineering: A systematic literature review. *International Journal of Information Management*, 2012.
- [113] S. Easterbrook, J. Singer, M-A. Storey, and D. Damian. Selecting Empirical Methods for Software Engineering Research. In *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer, 2008.
- [114] R. K. Yin. *Case Study Research: Design and Methods*. SAGE Publications, Inc., 4 edition, 2009.
- [115] C. B. Seaman. Qualitative Methods. In *Guide to Advanced Empirical Software Engineering*, pages 35–62. Springer, 2008.
- [116] J. Singer, S. E. Sim, and T. C. Lethbridge. Software Engineering Data Collection for Field Studies. In *Guide to Advanced Empirical Software Engineering*, pages 9–34. Springer, 2008.
- [117] P. Burnard. A method of analysing interview transcripts in qualitative research. *Nurse Education Today*, 11(6):461–466, 1991.
- [118] K. Hammar, F. Lin, and V. Tarasov. Information Reuse and Interoperability with Ontology Patterns and Linked Data. In *Business Information Systems Workshops*, pages 168–179. Springer, 2010.
- [119] M. Piattini, M. Genero, C. Calero, and G. Alarcos. Data Model Metrics. In *Handbook of Software Engineering and Knowledge Engineering*, volume 2, pages 215–229. World Scientific Pub Co Inc, 2002.
- [120] M. Fowler. *Analysis Patterns: Reusable Object Models*. Addison-Wesley, 1997.
- [121] D.R. Garrison, M. Cleveland-Innes, M. Koole, and J. Kappelman. Revisiting methodological issues in transcript analysis: Negotiated coding and reliability. *The Internet and Higher Education*, 9(1):1–8, 2006.
- [122] S. S. Stevens. On the Theory of Scales of Measurement. *Science*, 103(2684):677–680, 1946.
- [123] B. B. Brown. Delphi Process: A Methodology Used for the Elicitation of Opinions of Experts. Technical report, The RAND Corporation, 1968.

-
- [124] K. Hammar. Ontology Design Patterns in Use – Lessons Learnt from an Ontology Engineering Case. In *WOP 2012: Proceedings of the 3rd Workshop on Ontology Patterns, in conjunction with the 11th International Semantic Web Conference (ISWC) 2012*, 2012.
 - [125] J. Urbani, S. Kotoulas, E. Oren, and F. Van Harmelen. Scalable Distributed Reasoning using MapReduce. In *The Semantic Web - ISWC 2009*, pages 634–649. Springer, 2009.
 - [126] J. Urbani, S. Kotoulas, J. Maassen, F. Van Harmelen, and H. Bal. OWL reasoning with WebPIE: calculating the closure of 100 billion triples. In *The Semantic Web: Research and Applications*, pages 213–227. Springer, 2010.
 - [127] M. Horridge, M. E. Aranguren, J. Mortensen, M. Musen, and N. F. Noy. Ontology Design Pattern Language Expressivity Requirements. In *Proceedings of the 3rd Workshop on Ontology Patterns*, 2012.
 - [128] W3C. OWL 2 Web Ontology Language Profiles (Second Edition). <http://www.w3.org/TR/owl2-profiles/>, checked on: 2013-02-27.
 - [129] Y-B. Kang, Y-F. Li, and S. Krishnaswamy. Predicting Reasoning Performance Using Ontology Metrics. In *The Semantic Web – ISWC 2012*, pages 198–214, 2012.
 - [130] P. LePendou, N. Noy, C. Jonquet, P. Alexander, N. Shah, and M. Musen. Optimize First, Buy Later: Analyzing Metrics to Ramp-up Very Large Knowledge Bases. In *The Semantic Web – ISWC 2010*, pages 486–501. Springer, 2010.
 - [131] H. Zhang, Y-F. Li, and H. B. K. Tan. Measuring design complexity of semantic web ontologies. *Journal of Systems and Software*, 83(5):803–814, 2010.
 - [132] R. S. Goncalves, B. Parsia, and U. Sattler. Performance Heterogeneity and Approximate Reasoning in Description Logic Ontologies. In *The Semantic Web – ISWC 2012*, pages 82–98, 2012.

List of Figures

2.1	Ackoff's Knowledge Hierarchy	8
2.2	Course ontology example	11
2.3	Course data expressed using the ontology in Figure 2.2	11
2.4	The Semantic Web layer cake (adapted from [22])	14
2.5	Guarino's ontology classification hierarchy [28]	17
2.6	Context Dependant Information ODP	31
2.7	NeOn ODP Typology [8]	33
2.8	Blomqvist's ODP Typology [55]	35
2.9	XD Pattern Selection Approach [64]	36
2.10	XD Workflow (adapted from [64])	37
3.1	MAPPER validation framework metamodel [67, 68]	47
4.1	Licentiate project method overview	67
4.2	ODP literature study overview.	68
4.3	Performance indicator evaluation method	82
5.1	Quality Metamodel	86
6.1	Class in-degree and Class to property ratio distributions . . .	123
6.2	Class out-degree and Anonymous class count distributions . .	125
6.3	Subsumption depth indicator variance	126
7.1	Quality Metamodel	130

List of Tables

2.1	ODP papers by year	39
2.2	Institutes by paper count	39
2.3	Classification of the reviewed papers' connection to ODPs. . .	40
2.4	Validation levels of reviewed papers.	40
2.5	Quality of empirical validations.	40
2.6	Institution counts	41
2.7	ODP importance classification of reviewed papers.	41
3.1	ISO 25010 Quality In Use Model (adapted from [70])	51
3.2	ISO 25010 Product Quality Model (adapted from [70])	52
4.1	Content categories and definitions.	70
5.1	ODP Quality Model after ISO 25010 adaptation	90
5.2	Mapping of Thörn quality characteristics to ISO 25010	90
5.3	Thörn initial quality model mapping to ODP quality charac- teristics	92
5.4	ODP Quality Model after reusing Thörn [69]	92
5.5	Initial model indicator summary.	101
6.1	ODPs provided for case study participants	109
6.2	Distribution of fragments to codes.	110
6.3	ODPs used in learnability and usability evaluations.	115
6.4	Survey 1: competency question recognition correctness ratio.	115
6.5	Survey 1: class recognition correctness ratio.	116
6.6	Survey 1: time required to answer questions.	116
6.7	Task times (in minutes) and learnability metrics.	116
6.8	Perceived usability and learnability effects of indicators from survey 2.	117
6.9	Performance related indicator summary.	128
7.1	Evaluated indicator effects of the refined quality model.	133
7.2	Unevaluated indicator effects of the refined quality model.	134

- No 17 **Vojin Plavsic**: Interleaved Processing of Non-Numerical Data Stored on a Cyclic Memory. (Available at: FOA, Box 1165, S-581 11 Linköping, Sweden. FOA Report B30062E)
- No 28 **Arne Jönsson, Mikael Patel**: An Interactive Flowcharting Technique for Communicating and Realizing Algorithms, 1984.
- No 29 **Johnny Eckerland**: Retargeting of an Incremental Code Generator, 1984.
- No 48 **Henrik Nordin**: On the Use of Typical Cases for Knowledge-Based Consultation and Teaching, 1985.
- No 52 **Zebo Peng**: Steps Towards the Formalization of Designing VLSI Systems, 1985.
- No 60 **Johan Fagerström**: Simulation and Evaluation of Architecture based on Asynchronous Processes, 1985.
- No 71 **Jalal Maleki**: ICONStraint, A Dependency Directed Constraint Maintenance System, 1987.
- No 72 **Tony Larsson**: On the Specification and Verification of VLSI Systems, 1986.
- No 73 **Ola Strömfors**: A Structure Editor for Documents and Programs, 1986.
- No 74 **Christos Levcopoulos**: New Results about the Approximation Behavior of the Greedy Triangulation, 1986.
- No 104 **Shamsul I. Chowdhury**: Statistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology, 1987.
- No 108 **Rober Bilos**: Incremental Scanning and Token-Based Editing, 1987.
- No 111 **Hans Block**: SPORT-SORT Sorting Algorithms and Sport Tournaments, 1987.
- No 113 **Ralph Rönquist**: Network and Lattice Based Approaches to the Representation of Knowledge, 1987.
- No 118 **Mariam Kamkar, Nahid Shahmehri**: Affect-Chaining in Program Flow Analysis Applied to Queries of Programs, 1987.
- No 126 **Dan Strömberg**: Transfer and Distribution of Application Programs, 1987.
- No 127 **Kristian Sandahl**: Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems, 1987.
- No 139 **Christer Bäckström**: Reasoning about Interdependent Actions, 1988.
- No 140 **Mats Wirén**: On Control Strategies and Incrementality in Unification-Based Chart Parsing, 1988.
- No 146 **Johan Hultman**: A Software System for Defining and Controlling Actions in a Mechanical System, 1988.
- No 150 **Tim Hansen**: Diagnosing Faults using Knowledge about Malfunctioning Behavior, 1988.
- No 165 **Jonas Löwgren**: Supporting Design and Management of Expert System User Interfaces, 1989.
- No 166 **Ola Petersson**: On Adaptive Sorting in Sequential and Parallel Models, 1989.
- No 174 **Yngve Larsson**: Dynamic Configuration in a Distributed Environment, 1989.
- No 177 **Peter Åberg**: Design of a Multiple View Presentation and Interaction Manager, 1989.
- No 181 **Henrik Eriksson**: A Study in Domain-Oriented Tool Support for Knowledge Acquisition, 1989.
- No 184 **Ivan Rankin**: The Deep Generation of Text in Expert Critiquing Systems, 1989.
- No 187 **Simin Nadjim-Tehrani**: Contributions to the Declarative Approach to Debugging Prolog Programs, 1989.
- No 189 **Magnus Merkel**: Temporal Information in Natural Language, 1989.
- No 196 **Ulf Nilsson**: A Systematic Approach to Abstract Interpretation of Logic Programs, 1989.
- No 197 **Staffan Bonnier**: Horn Clause Logic with External Procedures: Towards a Theoretical Framework, 1989.
- No 203 **Christer Hansson**: A Prototype System for Logical Reasoning about Time and Action, 1990.
- No 212 **Björn Fjellborg**: An Approach to Extraction of Pipeline Structures for VLSI High-Level Synthesis, 1990.
- No 230 **Patrick Doherty**: A Three-Valued Approach to Non-Monotonic Reasoning, 1990.
- No 237 **Tomas Sokolnicki**: Coaching Partial Plans: An Approach to Knowledge-Based Tutoring, 1990.
- No 250 **Lars Strömberg**: Postmortem Debugging of Distributed Systems, 1990.
- No 253 **Torbjörn Näslund**: SL DFA-Resolution - Computing Answers for Negative Queries, 1990.
- No 260 **Peter D. Holmes**: Using Connectivity Graphs to Support Map-Related Reasoning, 1991.
- No 283 **Olof Johansson**: Improving Implementation of Graphical User Interfaces for Object-Oriented Knowledge- Bases, 1991.
- No 298 **Rolf G Larsson**: Aktivitetsbaserad kalkylering i ett nytt ekonomisystem, 1991.
- No 318 **Lena Srömbäck**: Studies in Extended Unification-Based Formalism for Linguistic Description: An Algorithm for Feature Structures with Disjunction and a Proposal for Flexible Systems, 1992.
- No 319 **Mikael Pettersson**: DML-A Language and System for the Generation of Efficient Compilers from Denotational Specification, 1992.
- No 326 **Andreas Kågedal**: Logic Programming with External Procedures: an Implementation, 1992.
- No 328 **Patrick Lambrix**: Aspects of Version Management of Composite Objects, 1992.
- No 333 **Xinli Gu**: Testability Analysis and Improvement in High-Level Synthesis Systems, 1992.
- No 335 **Torbjörn Näslund**: On the Role of Evaluations in Iterative Development of Managerial Support Systems, 1992.
- No 348 **Ulf Cederling**: Industrial Software Development - a Case Study, 1992.
- No 352 **Magnus Morin**: Predictable Cyclic Computations in Autonomous Systems: A Computational Model and Implementation, 1992.
- No 371 **Mehran Noghabai**: Evaluation of Strategic Investments in Information Technology, 1993.

No 378	Mats Larsson: A Transformational Approach to Formal Digital System Design, 1993.
No 380	Johan Ringström: Compiler Generation for Parallel Languages from Denotational Specifications, 1993.
No 381	Michael Jansson: Propagation of Change in an Intelligent Information System, 1993.
No 383	Jonni Harrius: An Architecture and a Knowledge Representation Model for Expert Critiquing Systems, 1993.
No 386	Per Österling: Symbolic Modelling of the Dynamic Environments of Autonomous Agents, 1993.
No 398	Johan Boye: Dependency-based Groudnness Analysis of Functional Logic Programs, 1993.
No 402	Lars Degerstedt: Tabulated Resolution for Well Founded Semantics, 1993.
No 406	Anna Moberg: Satellitkontor - en studie av kommunikationsmönster vid arbete på distans, 1993.
No 414	Peter Carlsson: Separation av företagsledning och finansiering - fallstudier av företagsledarutköp ur ett agent-teoretiskt perspektiv, 1994.
No 417	Camilla Sjöström: Revision och lagreglering - ett historiskt perspektiv, 1994.
No 436	Cecilia Sjöberg: Voices in Design: Argumentation in Participatory Development, 1994.
No 437	Lars Viklund: Contributions to a High-level Programming Environment for a Scientific Computing, 1994.
No 440	Peter Loborg: Error Recovery Support in Manufacturing Control Systems, 1994.
FHS 3/94	Owen Eriksson: Informationssystem med verksamhetskvalitet - utvärdering baserat på ett verksamhetsinriktat och samskapande perspektiv, 1994.
FHS 4/94	Karin Pettersson: Informationssystemstrukturer, ansvarsfördelning och användarinflytande - En komparativ studie med utgångspunkt i två informationssystemstrategier, 1994.
No 441	Lars Poignant: Informationsteknologi och företagsetablering - Effekter på produktivitet och region, 1994.
No 446	Gustav Fahl: Object Views of Relational Data in Multidatabase Systems, 1994.
No 450	Henrik Nilsson: A Declarative Approach to Debugging for Lazy Functional Languages, 1994.
No 451	Jonas Lind: Creditor - Firm Relations: an Interdisciplinary Analysis, 1994.
No 452	Martin Sköld: Active Rules based on Object Relational Queries - Efficient Change Monitoring Techniques, 1994.
No 455	Pär Carlshamre: A Collaborative Approach to Usability Engineering: Technical Communicators and System Developers in Usability-Oriented Systems Development, 1994.
FHS 5/94	Stefan Cronholm: Varför CASE-verktyg i systemutveckling? - En motiv- och konsekvensstudie avseende arbetssätt och arbetsformer, 1994.
No 462	Mikael Lindvall: A Study of Traceability in Object-Oriented Systems Development, 1994.
No 463	Fredrik Nilsson: Strategi och ekonomisk styrning - En studie av Sandviks förvärv av Bahco Verktyg, 1994.
No 464	Hans Olsén: Collage Induction: Proving Properties of Logic Programs by Program Synthesis, 1994.
No 469	Lars Karlsson: Specification and Synthesis of Plans Using the Features and Fluents Framework, 1995.
No 473	Ulf Söderman: On Conceptual Modelling of Mode Switching Systems, 1995.
No 475	Choong-ho Yi: Reasoning about Concurrent Actions in the Trajectory Semantics, 1995.
No 476	Bo Lagerström: Successiv resultatavräkning av pågående arbeten. - Fallstudier i tre byggföretag, 1995.
No 478	Peter Jonsson: Complexity of State-Variable Planning under Structural Restrictions, 1995.
FHS 7/95	Anders Avdic: Arbetsintegrerad systemutveckling med kalkylprogram, 1995.
No 482	Eva L. Ragnemalm: Towards Student Modelling through Collaborative Dialogue with a Learning Companion, 1995.
No 488	Eva Toller: Contributions to Parallel Multiparadigm Languages: Combining Object-Oriented and Rule-Based Programming, 1995.
No 489	Erik Stoy: A Petri Net Based Unified Representation for Hardware/Software Co-Design, 1995.
No 497	Johan Herber: Environment Support for Building Structured Mathematical Models, 1995.
No 498	Stefan Svenberg: Structure-Driven Derivation of Inter-Lingual Functor-Argument Trees for Multi-Lingual Generation, 1995.
No 503	Hee-Cheol Kim: Prediction and Postdiction under Uncertainty, 1995.
FHS 8/95	Dan Fristedt: Metoder i användning - mot förbättring av systemutveckling genom situationell metodkunskap och metodanalys, 1995.
FHS 9/95	Malin Bergvall: Systemförvaltning i praktiken - en kvalitativ studie avseende centrala begrepp, aktiviteter och ansvarsroller, 1995.
No 513	Joachim Karlsson: Towards a Strategy for Software Requirements Selection, 1995.
No 517	Jakob Axelsson: Schedulability-Driven Partitioning of Heterogeneous Real-Time Systems, 1995.
No 518	Göran Forslund: Toward Cooperative Advice-Giving Systems: The Expert Systems Experience, 1995.
No 522	Jörgen Andersson: Bilder av småföretagares ekonomistyrning, 1995.
No 538	Staffan Flodin: Efficient Management of Object-Oriented Queries with Late Binding, 1996.
No 545	Vadim Engelson: An Approach to Automatic Construction of Graphical User Interfaces for Applications in Scientific Computing, 1996.
No 546	Magnus Werner : Multidatabase Integration using Polymorphic Queries and Views, 1996.
FiF-a 1/96	Mikael Lind: Affärsprocessinriktat förändringsanalys - utveckling och tillämpning av synsätt och metod, 1996.
No 549	Jonas Hallberg: High-Level Synthesis under Local Timing Constraints, 1996.
No 550	Kristina Larsen: Förutsättningar och begränsningar för arbete på distans - erfarenheter från fyra svenska företag. 1996.
No 557	Mikael Johansson: Quality Functions for Requirements Engineering Methods, 1996.
No 558	Patrik Nordling: The Simulation of Rolling Bearing Dynamics on Parallel Computers, 1996.
No 561	Anders Ekman: Exploration of Polygonal Environments, 1996.

- No 563 **Niclas Andersson:** Compilation of Mathematical Models to Parallel Code, 1996.
- No 567 **Johan Jenvald:** Simulation and Data Collection in Battle Training, 1996.
- No 575 **Niclas Ohlsson:** Software Quality Engineering by Early Identification of Fault-Prone Modules, 1996.
- No 576 **Mikael Ericsson:** Commenting Systems as Design Support—A Wizard-of-Oz Study, 1996.
- No 587 **Jörgen Lindström:** Chefers användning av kommunikationsteknik, 1996.
- No 589 **Esa Falkenroth:** Data Management in Control Applications - A Proposal Based on Active Database Systems, 1996.
- No 591 **Niclas Wahllöf:** A Default Extension to Description Logics and its Applications, 1996.
- No 595 **Annika Larsson:** Ekonomisk Styrning och Organisatorisk Passion - ett interaktivt perspektiv, 1997.
- No 597 **Ling Lin:** A Value-based Indexing Technique for Time Sequences, 1997.
- No 598 **Rego Granlund:** C³Fire - A Microworld Supporting Emergency Management Training, 1997.
- No 599 **Peter Ingels:** A Robust Text Processing Technique Applied to Lexical Error Recovery, 1997.
- No 607 **Per-Arne Persson:** Toward a Grounded Theory for Support of Command and Control in Military Coalitions, 1997.
- No 609 **Jonas S Karlsson:** A Scalable Data Structure for a Parallel Data Server, 1997.
- FiF-a 4 **Carita Åbom:** Videomötesteknik i olika affärssituationer - möjligheter och hinder, 1997.
- FiF-a 6 **Tommy Wedlund:** Att skapa en företagsanpassad systemutvecklingsmodell - genom rekonstruktion, värdering och vidareutveckling i T50-bolag inom ABB, 1997.
- No 615 **Silvia Coradeschi:** A Decision-Mechanism for Reactive and Coordinated Agents, 1997.
- No 623 **Jan Ollinen:** Det flexibla kontorets utveckling på Digital - Ett stöd för multiflex? 1997.
- No 626 **David Byers:** Towards Estimating Software Testability Using Static Analysis, 1997.
- No 627 **Fredrik Eklund:** Declarative Error Diagnosis of GAPLog Programs, 1997.
- No 629 **Gunilla Ivefors:** Krigsspel och Informationsteknik inför en oförutsägbart framtid, 1997.
- No 631 **Jens-Olof Lindh:** Analysing Traffic Safety from a Case-Based Reasoning Perspective, 1997
- No 639 **Jukka Mäki-Turja:** Smalltalk - a suitable Real-Time Language, 1997.
- No 640 **Juha Takkinen:** CAFE: Towards a Conceptual Model for Information Management in Electronic Mail, 1997.
- No 643 **Man Lin:** Formal Analysis of Reactive Rule-based Programs, 1997.
- No 653 **Mats Gustafsson:** Bringing Role-Based Access Control to Distributed Systems, 1997.
- FiF-a 13 **Boris Karlsson:** Metodanalys för förståelse och utveckling av systemutvecklingsverksamhet. Analys och värdering av systemutvecklingsmodeller och dess användning, 1997.
- No 674 **Marcus Bjärelund:** Two Aspects of Automating Logics of Action and Change - Regression and Tractability, 1998.
- No 676 **Jan Håkegård:** Hierarchical Test Architecture and Board-Level Test Controller Synthesis, 1998.
- No 668 **Per-Ove Zetterlund:** Normering av svensk redovisning - En studie av tillkomsten av Redovisningsrådets rekommendation om koncernredovisning (RR01:91), 1998.
- No 675 **Jimmy Tjäder:** Projektledaren & planen - en studie av projektledning i tre installations- och systemutvecklingsprojekt, 1998.
- FiF-a 14 **Ulf Melin:** Informationssystem vid ökad affärs- och processororientering - egenskaper, strategier och utveckling, 1998.
- No 695 **Tim Heyer:** COMPASS: Introduction of Formal Methods in Code Development and Inspection, 1998.
- No 700 **Patrik Hägglund:** Programming Languages for Computer Algebra, 1998.
- FiF-a 16 **Marie-Therese Christiansson:** Inter-organisatorisk verksamhetsutveckling - metoder som stöd vid utveckling av partnerskap och informationssystem, 1998.
- No 712 **Christina Wennestam:** Information om immateriella resurser. Investeringar i forskning och utveckling samt i personal inom skogsindustrin, 1998.
- No 719 **Joakim Gustafsson:** Extending Temporal Action Logic for Ramification and Concurrency, 1998.
- No 723 **Henrik André-Jönsson:** Indexing time-series data using text indexing methods, 1999.
- No 725 **Erik Larsson:** High-Level Testability Analysis and Enhancement Techniques, 1998.
- No 730 **Carl-Johan Westin:** Informationsförsörjning: en fråga om ansvar - aktiviteter och uppdrag i fem stora svenska organisationers operativa informationsförsörjning, 1998.
- No 731 **Åse Jansson:** Miljöhänsyn - en del i företags styrning, 1998.
- No 733 **Thomas Padron-McCarthy:** Performance-Polymorphic Declarative Queries, 1998.
- No 734 **Anders Bäckström:** Värdeskapande kreditgivning - Kreditriskhantering ur ett agentteoretiskt perspektiv, 1998.
- FiF-a 21 **Ulf Seigerroth:** Integration av förändringsmetoder - en modell för välgrundad metodintegration, 1999.
- FiF-a 22 **Fredrik Öberg:** Object-Oriented Frameworks - A New Strategy for Case Tool Development, 1998.
- No 737 **Jonas Mellin:** Predictable Event Monitoring, 1998.
- No 738 **Joakim Eriksson:** Specifying and Managing Rules in an Active Real-Time Database System, 1998.
- FiF-a 25 **Bengt E W Andersson:** Samverkande informationssystem mellan aktörer i offentliga åtaganden - En teori om aktörsarenor i samverkan om utbyte av information, 1998.
- No 742 **Pawel Pietrzak:** Static Incorrectness Diagnosis of CLP (FD), 1999.
- No 748 **Tobias Ritzau:** Real-Time Reference Counting in RT-Java, 1999.
- No 751 **Anders Ferntoft:** Elektronisk affärskommunikation - kontaktkostnader och kontaktprocesser mellan kunder och leverantörer på producentmarknader, 1999.
- No 752 **Jo Skåmedal:** Arbete på distans och arbetsformens påverkan på resor och resmönster, 1999.
- No 753 **Johan Alvehus:** Mötets metaforer. En studie av berättelser om möten, 1999.

- No 754 **Magnus Lindahl:** Bankens villkor i låneavtal vid kreditgivning till högt belånade företagsförvärv: En studie ur ett agentteoretiskt perspektiv, 2000.
- No 766 **Martin V. Howard:** Designing dynamic visualizations of temporal data, 1999.
- No 769 **Jesper Andersson:** Towards Reactive Software Architectures, 1999.
- No 775 **Anders Henriksson:** Unique kernel diagnosis, 1999.
- FiF-a 30 **Pär J. Ågerfalk:** Pragmatization of Information Systems - A Theoretical and Methodological Outline, 1999.
- No 787 **Charlotte Björkegren:** Learning for the next project - Bearers and barriers in knowledge transfer within an organisation, 1999.
- No 788 **Håkan Nilsson:** Informationsteknik som drivkraft i granskningsprocessen - En studie av fyra revisionsbyråer, 2000.
- No 790 **Erik Berglund:** Use-Oriented Documentation in Software Development, 1999.
- No 791 **Klas Gäre:** Verksamhetsförändringar i samband med IS-införande, 1999.
- No 800 **Anders Subotic:** Software Quality Inspection, 1999.
- No 807 **Svein Bergum:** Managerial communication in telework, 2000.
- No 809 **Flavius Gruian:** Energy-Aware Design of Digital Systems, 2000.
- FiF-a 32 **Karin Hedström:** Kunskapsanvändning och kunskapsutveckling hos verksamhetskonsulter - Erfarenheter från ett FOU-samarbete, 2000.
- No 808 **Linda Askenäs:** Affärssystemet - En studie om teknikens aktiva och passiva roll i en organisation, 2000.
- No 820 **Jean Paul Meynard:** Control of industrial robots through high-level task programming, 2000.
- No 823 **Lars Hult:** Publika Gränsytor - ett designexempel, 2000.
- No 832 **Paul Pop:** Scheduling and Communication Synthesis for Distributed Real-Time Systems, 2000.
- FiF-a 34 **Göran Hultgren:** Nätverksinriktad Förändringsanalys - perspektiv och metoder som stöd för förståelse och utveckling av affärsrelationer och informationssystem, 2000.
- No 842 **Magnus Kald:** The role of management control systems in strategic business units, 2000.
- No 844 **Mikael Cäker:** Vad kostar kunden? Modeller för intern redovisning, 2000.
- FiF-a 37 **Ewa Braf:** Organisationers kunskapsverksamheter - en kritisk studie av "knowledge management", 2000.
- FiF-a 40 **Henrik Lindberg:** Webbaserade affärsprocesser - Möjligheter och begränsningar, 2000.
- FiF-a 41 **Benneth Christiansson:** Att komponentbasera informationssystem - Vad säger teori och praktik?, 2000.
- No. 854 **Ola Pettersson:** Deliberation in a Mobile Robot, 2000.
- No 863 **Dan Lawesson:** Towards Behavioral Model Fault Isolation for Object Oriented Control Systems, 2000.
- No 881 **Johan Moe:** Execution Tracing of Large Distributed Systems, 2001.
- No 882 **Yuxiao Zhao:** XML-based Frameworks for Internet Commerce and an Implementation of B2B e-procurement, 2001.
- No 890 **Annika Flycht-Eriksson:** Domain Knowledge Management in Information-providing Dialogue systems, 2001.
- FiF-a 47 **Per-Arne Segerkvist:** Webbaserade imaginära organisationers samverkansformer: Informationssystemarkitektur och aktörssamverkan som förutsättningar för affärsprocesser, 2001.
- No 894 **Stefan Svarén:** Styrning av investeringar i divisionaliserade företag - Ett koncernperspektiv, 2001.
- No 906 **Lin Han:** Secure and Scalable E-Service Software Delivery, 2001.
- No 917 **Emma Hansson:** Optionsprogram för anställda - en studie av svenska börsföretag, 2001.
- No 916 **Susanne Odar:** IT som stöd för strategiska beslut, en studie av datorimplementerade modeller av verksamhet som stöd för beslut om anskaffning av JAS 1982, 2002.
- FiF-a-49 **Stefan Holgersson:** IT-system och filtrering av verksamhetskunskap - kvalitetsproblem vid analyser och beslutsfattande som bygger på uppgifter hämtade från polisens IT-system, 2001.
- FiF-a-51 **Per Oscarsson:** Informationssäkerhet i verksamheter - begrepp och modeller som stöd för förståelse av informationssäkerhet och dess hantering, 2001.
- No 919 **Luis Alejandro Cortes:** A Petri Net Based Modeling and Verification Technique for Real-Time Embedded Systems, 2001.
- No 915 **Niklas Sandell:** Redovisning i skuggan av en bankkris - Värdering av fastigheter. 2001.
- No 931 **Fredrik Elg:** Ett dynamiskt perspektiv på individuella skillnader av heuristisk kompetens, intelligens, mentala modeller, mål och konfidens i kontroll av mikrovärlden Moro, 2002.
- No 933 **Peter Aronsson:** Automatic Parallelization of Simulation Code from Equation Based Simulation Languages, 2002.
- No 938 **Bourhane Kadmiry:** Fuzzy Control of Unmanned Helicopter, 2002.
- No 942 **Patrik Haslum:** Prediction as a Knowledge Representation Problem: A Case Study in Model Design, 2002.
- No 956 **Robert Sevenius:** On the instruments of governance - A law & economics study of capital instruments in limited liability companies, 2002.
- FiF-a 58 **Johan Petersson:** Lokala elektroniska marknadsplatser - informationssystem för platsbundna affärer, 2002.
- No 964 **Peter Bunus:** Debugging and Structural Analysis of Declarative Equation-Based Languages, 2002.
- No 973 **Gert Jervan:** High-Level Test Generation and Built-In Self-Test Techniques for Digital Systems, 2002.
- No 958 **Fredrika Berglund:** Management Control and Strategy - a Case Study of Pharmaceutical Drug Development, 2002.
- FiF-a 61 **Fredrik Karlsson:** Meta-Method for Method Configuration - A Rational Unified Process Case, 2002.
- No 985 **Sorin Manolache:** Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times, 2002.
- No 982 **Diana Szentiványi:** Performance and Availability Trade-offs in Fault-Tolerant Middleware, 2002.
- No 989 **Iakov Nakhimovski:** Modeling and Simulation of Contacting Flexible Bodies in Multibody Systems, 2002.
- No 990 **Levon Saldamli:** PDEModelica - Towards a High-Level Language for Modeling with Partial Differential Equations, 2002.

- No 991 **Almut Herzog:** Secure Execution Environment for Java Electronic Services, 2002.
- No 999 **Jon Edvardsson:** Contributions to Program- and Specification-based Test Data Generation, 2002.
- No 1000 **Anders Arpteg:** Adaptive Semi-structured Information Extraction, 2002.
- No 1001 **Andrzej Bednarski:** A Dynamic Programming Approach to Optimal Retargetable Code Generation for Irregular Architectures, 2002.
- No 988 **Mattias Arvola:** Good to use! : Use quality of multi-user applications in the home, 2003.
- FiF-a 62 **Lennart Ljung:** Utveckling av en projektivitetsmodell - om organisationers förmåga att tillämpa projektarbetsformen, 2003.
- No 1003 **Pernilla Qvarfordt:** User experience of spoken feedback in multimodal interaction, 2003.
- No 1005 **Alexander Siemers:** Visualization of Dynamic Multibody Simulation With Special Reference to Contacts, 2003.
- No 1008 **Jens Gustavsson:** Towards Unanticipated Runtime Software Evolution, 2003.
- No 1010 **Calin Curescu:** Adaptive QoS-aware Resource Allocation for Wireless Networks, 2003.
- No 1015 **Anna Andersson:** Management Information Systems in Process-oriented Healthcare Organisations, 2003.
- No 1018 **Björn Johansson:** Feedforward Control in Dynamic Situations, 2003.
- No 1022 **Traian Pop:** Scheduling and Optimisation of Heterogeneous Time/Event-Triggered Distributed Embedded Systems, 2003.
- FiF-a 65 **Britt-Marie Johansson:** Kundkommunikation på distans - en studie om kommunikationsmediets betydelse i affärstransaktioner, 2003.
- No 1024 **Aleksandra Tešanovic:** Towards Aspectual Component-Based Real-Time System Development, 2003.
- No 1034 **Arja Vainio-Larsson:** Designing for Use in a Future Context - Five Case Studies in Retrospect, 2003.
- No 1033 **Peter Nilsson:** Svenska bankers redovisningsval vid reservering för befarade kreditförluster - En studie vid införandet av nya redovisningsregler, 2003.
- FiF-a 69 **Fredrik Ericsson:** Information Technology for Learning and Acquiring of Work Knowledge, 2003.
- No 1049 **Marcus Comstedt:** Towards Fine-Grained Binary Composition through Link Time Weaving, 2003.
- No 1052 **Åsa Hedenskog:** Increasing the Automation of Radio Network Control, 2003.
- No 1054 **Claudiu Duma:** Security and Efficiency Tradeoffs in Multicast Group Key Management, 2003.
- FiF-a 71 **Emma Eliason:** Effektanalys av IT-systems handlingsutrymme, 2003.
- No 1055 **Kari Cederberg:** Experiments in Indirect Fault Injection with Open Source and Industrial Software, 2003.
- No 1058 **Daniel Karlsson:** Towards Formal Verification in a Component-based Reuse Methodology, 2003.
- FiF-a 73 **Anders Hjalmarsson:** Att etablera och vidmakthålla förbättringsverksamhet - behovet av koordination och interaktion vid förändring av systemutvecklingsverksamheter, 2004.
- No 1079 **Pontus Johansson:** Design and Development of Recommender Dialogue Systems, 2004.
- No 1084 **Charlotte Stoltz:** Calling for Call Centres - A Study of Call Centre Locations in a Swedish Rural Region, 2004.
- FiF-a 74 **Björn Johansson:** Deciding on Using Application Service Provision in SMEs, 2004.
- No 1094 **Genevieve Gorrell:** Language Modelling and Error Handling in Spoken Dialogue Systems, 2004.
- No 1095 **Ulf Johansson:** Rule Extraction - the Key to Accurate and Comprehensible Data Mining Models, 2004.
- No 1099 **Sonia Sangari:** Computational Models of Some Communicative Head Movements, 2004.
- No 1110 **Hans Nässla:** Intra-Family Information Flow and Prospects for Communication Systems, 2004.
- No 1116 **Henrik Sällberg:** On the value of customer loyalty programs - A study of point programs and switching costs, 2004.
- FiF-a 77 **Ulf Larsson:** Designarbete i dialog - karaktärisering av interaktionen mellan användare och utvecklare i en systemutvecklingsprocess, 2004.
- No 1126 **Andreas Borg:** Contribution to Management and Validation of Non-Functional Requirements, 2004.
- No 1127 **Per-Ola Kristensson:** Large Vocabulary Shorthand Writing on Stylus Keyboard, 2004.
- No 1132 **Pär-Anders Albinsson:** Interacting with Command and Control Systems: Tools for Operators and Designers, 2004.
- No 1130 **Ioan Chisalitã:** Safety-Oriented Communication in Mobile Networks for Vehicles, 2004.
- No 1138 **Thomas Gustafsson:** Maintaining Data Consistency in Embedded Databases for Vehicular Systems, 2004.
- No 1149 **Vaida Jakoniënė:** A Study in Integrating Multiple Biological Data Sources, 2005.
- No 1156 **Abdil Rashid Mohamed:** High-Level Techniques for Built-In Self-Test Resources Optimization, 2005.
- No 1162 **Adrian Pop:** Contributions to Meta-Modeling Tools and Methods, 2005.
- No 1165 **Fidel Vascós Palacios:** On the information exchange between physicians and social insurance officers in the sick leave process: an Activity Theoretical perspective, 2005.
- FiF-a 84 **Jenny Lagsten:** Verksamhetsutvecklande utvärdering i informationssystemprojekt, 2005.
- No 1166 **Emma Larsdotter Nilsson:** Modeling, Simulation, and Visualization of Metabolic Pathways Using Modelica, 2005.
- No 1167 **Christina Keller:** Virtual Learning Environments in higher education. A study of students' acceptance of educational technology, 2005.
- No 1168 **Cécile Åberg:** Integration of organizational workflows and the Semantic Web, 2005.
- FiF-a 85 **Anders Forsman:** Standardisering som grund för informationssamverkan och IT-tjänster - En fallstudie baserad på trafikinformationstjänsten RDS-TMC, 2005.
- No 1171 **Yu-Hsing Huang:** A systemic traffic accident model, 2005.
- FiF-a 86 **Jan Olausson:** Att modellera uppdrag - grunder för förståelse av processinriktade informationssystem i transaktionsintensiva verksamheter, 2005.
- No 1172 **Petter Ahlström:** Affärsstrategier för seniorbostadsmarknaden, 2005.
- No 1183 **Mathias Cöster:** Beyond IT and Productivity - How Digitization Transformed the Graphic Industry, 2005.
- No 1184 **Åsa Horzella:** Beyond IT and Productivity - Effects of Digitized Information Flows in Grocery Distribution, 2005.
- No 1185 **Maria Kollberg:** Beyond IT and Productivity - Effects of Digitized Information Flows in the Logging Industry, 2005.

No 1190 **David Dinka**: Role and Identity - Experience of technology in professional settings, 2005.

No 1191 **Andreas Hansson**: Increasing the Storage Capacity of Recursive Auto-associative Memory by Segmenting Data, 2005.

No 1192 **Nicklas Bergfeldt**: Towards Detached Communication for Robot Cooperation, 2005.

No 1194 **Dennis Maciuszek**: Towards Dependable Virtual Companions for Later Life, 2005.

No 1204 **Beatrice Alenljung**: Decision-making in the Requirements Engineering Process: A Human-centered Approach, 2005.

No 1206 **Anders Larsson**: System-on-Chip Test Scheduling and Test Infrastructure Design, 2005.

No 1207 **John Wilander**: Policy and Implementation Assurance for Software Security, 2005.

No 1209 **Andreas Käll**: Översättningar av en managementmodell - En studie av införandet av Balanced Scorecard i ett landsting, 2005.

No 1225 **He Tan**: Aligning and Merging Biomedical Ontologies, 2006.

No 1228 **Artur Wilk**: Descriptive Types for XML Query Language Xcerpt, 2006.

No 1229 **Per Olof Pettersson**: Sampling-based Path Planning for an Autonomous Helicopter, 2006.

No 1231 **Kalle Burbeck**: Adaptive Real-time Anomaly Detection for Safeguarding Critical Networks, 2006.

No 1233 **Daniela Mihailescu**: Implementation Methodology in Action: A Study of an Enterprise Systems Implementation Methodology, 2006.

No 1244 **Jörgen Skågeby**: Public and Non-public gifting on the Internet, 2006.

No 1248 **Karolina Eliasson**: The Use of Case-Based Reasoning in a Human-Robot Dialog System, 2006.

No 1263 **Misook Park-Westman**: Managing Competence Development Programs in a Cross-Cultural Organisation - What are the Barriers and Enablers, 2006.

FiF-a 90 **Amra Halilovic**: Ett praktikperspektiv på hantering av mjukvarukomponenter, 2006.

No 1272 **Raquel Flodström**: A Framework for the Strategic Management of Information Technology, 2006.

No 1277 **Viacheslav Izosimov**: Scheduling and Optimization of Fault-Tolerant Embedded Systems, 2006.

No 1283 **Håkan Hasewinkel**: A Blueprint for Using Commercial Games off the Shelf in Defence Training, Education and Research Simulations, 2006.

FiF-a 91 **Hanna Broberg**: Verksamhetsanpassade IT-stöd - Designteori och metod, 2006.

No 1286 **Robert Kaminski**: Towards an XML Document Restructuring Framework, 2006.

No 1293 **Jiri Trnka**: Prerequisites for data sharing in emergency management, 2007.

No 1302 **Björn Hägglund**: A Framework for Designing Constraint Stores, 2007.

No 1303 **Daniel Andreasson**: Slack-Time Aware Dynamic Routing Schemes for On-Chip Networks, 2007.

No 1305 **Magnus Ingmarsson**: Modelling User Tasks and Intentions for Service Discovery in Ubiquitous Computing, 2007.

No 1306 **Gustaf Svedjemo**: Ontology as Conceptual Schema when Modelling Historical Maps for Database Storage, 2007.

No 1307 **Gianpaolo Conte**: Navigation Functionalities for an Autonomous UAV Helicopter, 2007.

No 1309 **Ola Leifler**: User-Centric Critiquing in Command and Control: The DKExpert and ComPlan Approaches, 2007.

No 1312 **Henrik Svensson**: Embodied simulation as off-line representation, 2007.

No 1313 **Zhiyuan He**: System-on-Chip Test Scheduling with Defect-Probability and Temperature Considerations, 2007.

No 1317 **Jonas Elmqvist**: Components, Safety Interfaces and Compositional Analysis, 2007.

No 1320 **Håkan Sundblad**: Question Classification in Question Answering Systems, 2007.

No 1323 **Magnus Lundqvist**: Information Demand and Use: Improving Information Flow within Small-scale Business Contexts, 2007.

No 1329 **Martin Magnusson**: Deductive Planning and Composite Actions in Temporal Action Logic, 2007.

No 1331 **Mikael Asplund**: Restoring Consistency after Network Partitions, 2007.

No 1332 **Martin Fransson**: Towards Individualized Drug Dosage - General Methods and Case Studies, 2007.

No 1333 **Karin Camara**: A Visual Query Language Served by a Multi-sensor Environment, 2007.

No 1337 **David Broman**: Safety, Security, and Semantic Aspects of Equation-Based Object-Oriented Languages and Environments, 2007.

No 1339 **Mikhail Chalabine**: Invasive Interactive Parallelization, 2007.

No 1351 **Susanna Nilsson**: A Holistic Approach to Usability Evaluations of Mixed Reality Systems, 2008.

No 1353 **Shanai Ardi**: A Model and Implementation of a Security Plug-in for the Software Life Cycle, 2008.

No 1356 **Erik Kuiper**: Mobility and Routing in a Delay-tolerant Network of Unmanned Aerial Vehicles, 2008.

No 1359 **Jana Rambusch**: Situated Play, 2008.

No 1361 **Martin Karresand**: Completing the Picture - Fragments and Back Again, 2008.

No 1363 **Per Nyblom**: Dynamic Abstraction for Interleaved Task Planning and Execution, 2008.

No 1371 **Fredrik Lantz**: Terrain Object Recognition and Context Fusion for Decision Support, 2008.

No 1373 **Martin Östlund**: Assistance Plus: 3D-mediated Advice-giving on Pharmaceutical Products, 2008.

No 1381 **Håkan Lundvall**: Automatic Parallelization using Pipelining for Equation-Based Simulation Languages, 2008.

No 1386 **Mirko Thorstensson**: Using Observers for Model Based Data Collection in Distributed Tactical Operations, 2008.

No 1387 **Bahlol Rahimi**: Implementation of Health Information Systems, 2008.

No 1392 **Maria Holmqvist**: Word Alignment by Re-using Parallel Phrases, 2008.

No 1393 **Mattias Eriksson**: Integrated Software Pipelining, 2009.

No 1401 **Annika Öhgren**: Towards an Ontology Development Methodology for Small and Medium-sized Enterprises, 2009.

No 1410 **Rickard Holmsmark**: Deadlock Free Routing in Mesh Networks on Chip with Regions, 2009.

No 1421 **Sara Stymne**: Compound Processing for Phrase-Based Statistical Machine Translation, 2009.

No 1427 **Tommy Ellqvist**: Supporting Scientific Collaboration through Workflows and Provenance, 2009.

No 1450 **Fabian Segelström**: Visualisations in Service Design, 2010.

No 1459 **Min Bao**: System Level Techniques for Temperature-Aware Energy Optimization, 2010.

No 1466 **Mohammad Saifullah:** Exploring Biologically Inspired Interactive Networks for Object Recognition, 2011

No 1468 **Qiang Liu:** Dealing with Missing Mappings and Structure in a Network of Ontologies, 2011.

No 1469 **Ruxandra Pop:** Mapping Concurrent Applications to Multiprocessor Systems with Multithreaded Processors and Network on Chip-Based Interconnections, 2011.

No 1476 **Per-Magnus Olsson:** Positioning Algorithms for Surveillance Using Unmanned Aerial Vehicles, 2011.

No 1481 **Anna Vapen:** Contributions to Web Authentication for Untrusted Computers, 2011.

No 1485 **Loove Broms:** Sustainable Interactions: Studies in the Design of Energy Awareness Artefacts, 2011.

FiF-a 101 **Johan Blomkvist:** Conceptualising Prototypes in Service Design, 2011.

No 1490 **Håkan Warnquist:** Computer-Assisted Troubleshooting for Efficient Off-board Diagnosis, 2011.

No 1503 **Jakob Rosén:** Predictable Real-Time Applications on Multiprocessor Systems-on-Chip, 2011.

No 1504 **Usman Dastgeer:** Skeleton Programming for Heterogeneous GPU-based Systems, 2011.

No 1506 **David Landén:** Complex Task Allocation for Delegation: From Theory to Practice, 2011.

No 1507 **Kristian Stavåker:** Contributions to Parallel Simulation of Equation-Based Models on Graphics Processing Units, 2011.

No 1509 **Mariusz Wzorek:** Selected Aspects of Navigation and Path Planning in Unmanned Aircraft Systems, 2011.

No 1510 **Piotr Rudol:** Increasing Autonomy of Unmanned Aircraft Systems Through the Use of Imaging Sensors, 2011.

No 1513 **Anders Carstensen:** The Evolution of the Connector View Concept: Enterprise Models for Interoperability Solutions in the Extended Enterprise, 2011.

No 1523 **Jody Foo:** Computational Terminology: Exploring Bilingual and Monolingual Term Extraction, 2012.

No 1550 **Anders Fröberg:** Models and Tools for Distributed User Interface Development, 2012.

No 1558 **Dimitar Nikolov:** Optimizing Fault Tolerance for Real-Time Systems, 2012.

No 1582 **Dennis Andersson:** Mission Experience: How to Model and Capture it to Enable Vicarious Learning, 2013.

No 1586 **Massimiliano Raciti:** Anomaly Detection and its Adaptation: Studies on Cyber-physical Systems, 2013.

No 1588 **Banafsheh Khademhosseini:** Towards an Approach for Efficiency Evaluation of Enterprise Modeling Methods, 2013.

No 1589 **Amy Rankin:** Resilience in High Risk Work: Analysing Adaptive Performance, 2013.

No 1592 **Martin Sjölund:** Tools for Understanding, Debugging, and Simulation Performance Improvement of Equation-Based Models, 2013.

No 1606 **Karl Hammar:** Towards an Ontology Design Pattern Quality Model, 2013.