

Linköping Studies in Science and Technology

Dissertations. No. 1879

Content Ontology Design Patterns: Qualities, Methods, and Tools

by

Karl Hammar



JÖNKÖPING UNIVERSITY



LINKÖPING
UNIVERSITY

Linköping University
Department of Computer and Information Science
Division of Human-Centered Systems
SE-581 83 Linköping, Sweden

Linköping 2017

© 2017 Karl Hammar
Cover photograph by Jenny Hammar

ISBN: 978-91-7685-454-9
ISSN: 0345-7524

URL: <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-139584/>
Printed by LiU Tryck, Linköping 2017

Abstract

Ontologies are formal knowledge models that describe concepts and relationships and enable data integration, information search, and reasoning. Ontology Design Patterns (ODPs) are reusable solutions intended to simplify ontology development and support the use of semantic technologies by ontology engineers. ODPs document and package good modelling practices for reuse, ideally enabling inexperienced ontologists to construct high-quality ontologies. Although ODPs are already used for development, there are still remaining challenges that have not been addressed in the literature. These research gaps include a lack of knowledge about (1) which ODP features are important for ontology engineering, (2) less experienced developers' preferences and barriers for employing ODP tooling, and (3) the suitability of the eXtreme Design (XD) ODP usage methodology in non-academic contexts.

This dissertation aims to close these gaps by combining quantitative and qualitative methods, primarily based on five ontology engineering projects involving inexperienced ontologists. A series of ontology engineering workshops and surveys provided data about developer preferences regarding ODP features, ODP usage methodology, and ODP tooling needs. Other data sources are ontologies and ODPs published on the web, which have been studied in detail. To evaluate tooling improvements, experimental approaches provide data from comparison of new tools and techniques against established alternatives.

The analysis of the gathered data resulted in a set of measurable quality indicators that cover aspects of ODP documentation, formal representation or axiomatisation, and usage by ontologists. These indicators highlight quality trade-offs: for instance, between ODP *Learnability* and *Reusability*, or between *Functional Suitability* and *Performance Efficiency*. Furthermore, the results demonstrate a need for ODP tools that support three novel property specialisation strategies, and highlight the preference of inexperienced developers for template-based ODP instantiation—neither of which are supported in prior tooling. The studies also resulted in improvements to ODP search engines based on ODP-specific attributes. Finally, the analysis shows that XD should include guidance for the developer roles and responsibilities in ontology engineering projects, suggestions on how to reuse existing ontology resources, and approaches for adapting XD to project-specific contexts.

Populärvetenskaplig sammanfattning

De senaste två decennierna har användningen av Internet och dess killer app World Wide Web (i dagligt tal *webben*) ökat explosionsartat, såväl vad gäller antal användare som antal tillgängliga tjänster. Vi surfar inte längre bara på webben för att söka efter information – snarare lever vi i allt högre utsträckning våra liv uppkopplade via den. Vi handlar mat och gör bankärenden, vi bokat semestrar och läser böcker, vi delar bilder och videos och minnen med varandra. I de flesta avseenden har webben och dess möjligheter utvecklats långt bortom vad de flesta trodde var möjligt.

I andra avseenden har vi dock bara skrapat på ytan. Webben är fortfarande i huvudsak ett medium för kommunikation människor emellan. Det innehåll som publiceras på webben, oavsett om det är i text-, bild-, eller videoformat, är till större del oförståeligt för programvara – det är avsett för konsumtion av människor, som tolkar, förstår, och eventuellt agerar på det. Om webbinnehåll i stället kunde tolkas och förstås maskinellt av mjukvaror, så skulle det möjliggöra mängder av innovativa nya integrerade tjänster och produkter: intelligenta mobila agenter skulle kunna svara på användarens frågor genom att läsa och förstå information publicerat på webben, snarare än att bara svara vad de blivit programmerade till; information från olika företags eller myndigheters webbsidor skulle enkelt kunna samköras, så att till exempel kunder enkelt kan jämföra liknande produkter eller tjänster hos olika webbhandlare; rapportering från olika nyhetssajter skulle kunna jämföras maskinellt för att detektera olika tolkningar eller vinklingar på det rapporterade materialet, etc. Visionen av den här framtida webben där människor och maskiner delar information på ett sömlöst och integrerat sätt har ett namn: den *Semantiska Webben*.

För att möjliggöra den Semantiska Webben så krävs att deltagande människor och system kommer överens om och standardiserar kommunikation på två olika nivåer: dels på format- och syntaxnivå, och dels på konceptuell definitionsnivå. Den första nivån rör det rent tekniska informationsutbytet mellan system för olika sorters data (textsträngar, siffror, bilder, etc.). Parallellt med och på grund av Internets och webbens framväxt så har ett flertal standarder etablerats kring dessa format för datarepresentation

och -utbyte, vilka fungerar väl även för en Semantisk Webb. På den senare nivån, standardiseringen av formella definitioner för de ting som system och människor skall kunna kommunicera om (de *semantiska ontologierna*), finns det betydligt mer kvar att göra.

En stor utmaning är att konstruktionen av dessa ontologier är förhållandevis komplex och kräver kunskaper som få programmerare eller analytiker besitter – de behöver ha djupgående förståelse för alla de koncept som de vill fånga och formalisera definitioner av (produkter, händelser, organisationer, processer, etc), de behöver ha stor erfarenhet av konceptuell modellering, och de behöver känna till de relativt komplicerade format och verktyg som används för att konstruera ontologier.

Designmönster för ontologier (*Ontology Design Patterns*, eller *ODP:er*) är avsedda att förenkla utvecklingen av ontologier. ODP:er beskriver, i text och i bild, vanligt förekommande modelleringsproblem och etablerade lösningar på dessa problem. En ODP kan till exempel beskriva hur man bäst modellerar händelser, oaktat vilken typ av händelse (en konsert, ett kurstillfälle, en flygresa, etc.) det rör sig om. ODP:er brukar, utöver sagda beskrivning, även bestå av en liten och återanvändbar bit ontologi-kod, som en utvecklare enkelt kan anpassa och återanvända. Genom användning av ODP:er och ODP-baserade verktyg så kan utvecklarens behov av djupare kunskaper inom konceptuell modellering och ontologiutveckling minskas väsentligt.

Den här avhandlingen studerar ODP:er och ODP-användning ur tre perspektiv. Till att börja med undersöker författaren vilka egenskaper eller kvaliteter hos ODP:er som är viktiga för deras användning, och hur dessa kvaliteter kan mätas. Avhandlingens resultat avseende dessa frågor formaliseras i en kvalitetsmodell som inkluderar ett antal olika kvaliteter (*Functional Suitability, Usability, Maintainability, Compatibility, Resulting performance efficiency*), ett antal underkvaliteter för var och en av dessa, och ett antal (38 st.) kvalitetsindikatorer som bidrar till respektive kvalitet eller underkvalitet.

Vidare studeras i avhandlingen hur verktyg för användning av ODP:er kan förbättras så att utvecklare enklare kan hitta lämpliga ODP:er, och lättare kan använda dessa i sina ontologier på korrekt sätt. Resultat inom detta område inkluderar nya metoder (och tillhörande verktyg) för instansiering av ODP:er in i en ontologi, metoder som är särskilt lämpade i scenarion där användaren är mindre kunnig om ontologi-utveckling, eller i scenarion där användaren enklare vill kunna integrera sina ontologier och sina data med sedan tidigare publicerade ontologier och data. Andra resultat på verktygsområdet inkluderar en sökmotor för ODP:er som presterar bättre än tidigare använd teknik för att hitta ODP:er.

Slutligen undersöks i avhandlingen hur en etablerad projektmetod för användning av ODP:er, eXtreme Design-metoden (XD), fungerar i praktiken och hur den kan förbättras för att matcha utvecklarens behov av metodstöd. Författaren finner att XD kan förbättras genom tydligare dokumentation

av olika projektroller och ansvarsområden, genom rekommendationer för återanvändning av etablerade ontologier (inte bara ODP:er), och genom anpassningar som tar större hänsyn till projektspecifika sammanhang som kundrelationer, utvecklingsteamets kompetenser, och teamets organisation.

Sammanfattningsvis bidrar den här avhandlingens resultat till att öka kunskapen om hur ODP:er bör vara konstruerade och beskrivna, hur ODP:er används på bäst sätt, och hur verktyg för att stödja sådan användning bör fungera. Dessa bidrag möjliggör förenklad utveckling av ontologier och ontologi-baserad teknik, vilket i sin tur bidrar till utvecklingen av den Semantiska Webben och de många funktioner och tjänster som följer utav den.

Acknowledgements

This research was financed by and carried out at the Department of Computer Science and Informatics at Jönköping University and the Department of Computer and Information Science at Linköping University. Data gathering took place in several research projects, funded by both national and international organisations. I would like to express my gratitude to the partners involved in and the funders of these projects:

- **IMSK**: Integrated Mobile Security Kit (funded by the European Union Seventh Framework Programme, FP7/2007-2013, grant agreement 218038)
- **VALCRI**: Visual Analytics for Sense-Making in Criminal Intelligence Analysis (funded by the European Union Seventh Framework Programme, FP7/2007-2013, grant agreement 608142)
- **SSyncAHD**: Standardising Syndromic Classification in Animal Health Data (funded by Vinnova)
- **OSTAG**: Ontology-based Software Test Case Generation (funded by the Swedish Knowledge Foundation grant number 20140170)
- **E-care@home** (a “SIDUS”—Strong Distributed Research Environment—funded by the Swedish Knowledge Foundation)

My supervision team has consisted of Professor **Henrik Eriksson** (Linköping University), Associate Professor **Vladimir Tarasov** (Jönköping University), and Assistant Professor **Eva Blomqvist** (Linköping University). In the earlier half of this PhD project (which awarded the Swedish licentiate degree), Professor **Kurt Sandkuhl** was also engaged. I owe a tremendous debt of gratitude to all of my supervisors, who have helped me with ideas, feedback, and encouragement, over the past years. However, I would like to single out **Eva** in particular—a PhD project is a rather lengthy and sometimes tiring undertaking, and Eva has helped see this through, not only as a supervisor, but as a mentor, and as a friend. Thank you.

In addition to the supervision team, several other colleagues have also contributed to this work. At Jönköping University, **Christer Thörn**, **Ulf Seigerroth**, **Fredrik Abrahamsson**, and **Ulf Johansson** have all read

and commented on parts of the dissertation. **Anders Arvidsson** has kept management and students off my back so that I had time to get the research done. At Linköping University, **Anne Moe** has kept the formal processes running smoothly, and **Brittany Shahmehri** has helped with language-proofing the text. Any remaining flaws in content or style are of course entirely my own.

Last but not least, I am most grateful for the support I've had from friends and family. **Fredrik R Krohnman** has helped me stay clinically and cynically sane. My beautiful daughter **Astrid** has helped me keep the big picture in focus, by always putting a smile on my face. **Per-Olof** and **Gudrun Nyberg** have, in turn, kept a smile on Astrid's face, and helped take care of things when I've been travelling to conferences and project meetings. Finally, and most importantly, my wonderful and talented wife **Jenny** has supported me through thick and thin. Jenny: I love you endlessly.

Karl Hammar
Jönköping, July 2017

Contents

1	Introduction	1
1.1	Problem	3
1.2	Research Questions	4
1.2.1	Delimitations	5
1.3	Contributions	5
1.4	Summary of Publications	6
1.5	Dissertation Outline	10
2	Background and Related Work	13
2.1	Knowledge Modelling and Ontologies	13
2.1.1	Data, Information, and Knowledge	13
2.1.2	Terminological and Assertional Knowledge	15
2.1.3	Ontology Components	16
2.1.4	RDF, RDFS, and OWL	19
2.2	Ontology Applications	23
2.2.1	Ontology Types	23
2.2.2	Linked Data	23
2.2.3	Semantic Search	24
2.2.4	Reasoning Tasks	26
2.3	Ontology Development	27
2.3.1	METHONTOLOGY	27
2.3.2	On-To-Knowledge	28
2.3.3	DILIGENT	29
2.3.4	SAMOD	31
2.3.5	Ontology Development 101	32
2.4	Ontology Design Patterns	33
2.4.1	ODP Typologies	36
2.4.2	ODP-based Ontology Construction	39
2.4.3	Other Perspectives on ODPs	42
2.5	Quality Frameworks	43
2.5.1	MAPPER	43
2.5.2	Conceptual Model Quality	44
2.5.3	Entity Relationship Model Quality	46
2.5.4	Information System Quality	47

2.5.5	Pattern Quality	49
2.6	Ontology Quality Evaluation	51
2.6.1	O ² and oQual	51
2.6.2	ONTOMETRIC	52
2.6.3	OntoClean	53
2.6.4	Terminological Cycle Effects	55
2.6.5	ODP Documentation Template Effects	55
3	Research Method	57
3.1	Applicable Methods in the Computing Disciplines	57
3.1.1	Design Science—A Pragmatic Approach	60
3.1.2	Systematic Literature Review	63
3.1.3	Interviews	65
3.1.4	Surveys	66
3.1.5	Researcher Logs or Participant Diaries	69
3.1.6	Experimentation	70
3.2	Research Process	71
3.2.1	Answering Research Question 1	75
3.2.2	Answering Research Question 2	78
3.2.3	Answering Research Question 3	80
3.2.4	Projects	82
3.2.5	Research Logs	85
3.2.6	Qualitative Data Analysis	85
3.2.7	Surveys Employed	88
3.3	Attributes of the Research Process	90
3.3.1	Workshop Observations	90
3.3.2	Surveys	91
3.3.3	ODP Feature Studies	91
3.3.4	Experiments	92
4	ODP Quality Model	93
4.1	Initial Model	93
4.1.1	Quality Metamodel Development	94
4.1.2	Initial Quality Characteristics	95
4.1.3	Initial Quality Indicators	97
4.2	Second Generation Model	101
4.2.1	IMSK Workshop	101
4.2.2	ILOG Course Study	105
4.2.3	Performance Indicator Evaluation	111
4.3	Third Generation Model	119
4.3.1	Ontology Engineering Survey	120
4.3.2	ODP Design Preferences Survey	122
4.3.3	Ontology Engineering Workshop Observations	125
4.4	Summary: Resulting Quality Model	128
4.4.1	Quality Metamodel	128
4.4.2	Quality Characteristics	130

4.4.3	Quality Indicators and Effects	132
4.4.4	Quality Trade-offs	133
4.4.5	Notes on Unstudied Qualities	136
5	ODP Tool Support Improvement	139
5.1	ODP Search	139
5.1.1	Motivation	140
5.1.2	Proposed Solution	140
5.1.3	Evaluation	141
5.2	ODP Specialisation Strategies	143
5.2.1	Understanding ODP Specialisation Practices	143
5.2.2	Strategy Usages and Effects	148
5.3	Template-Based Instantiation	152
5.3.1	Motivation	153
5.3.2	Proposed Solution	155
5.3.3	Evaluation	157
5.4	Summary: eXtreme Design for Protégé	159
5.4.1	Motivation	159
5.4.2	Developed Solution	161
5.4.3	Evaluation	163
6	ODP Methodology Development	169
6.1	Project Roles	169
6.1.1	Observation: Role and Task Challenges	170
6.1.2	Suggestion: XD Roles and Responsibilities	175
6.2	Ontology Reuse	177
6.2.1	Observation: Ontology Reuse Challenges	178
6.2.2	Suggestion: An Ontology Reuse Checklist	181
6.3	Context-Based Methodology Adaptation	184
6.3.1	Observation: Real World XD Project Contexts	184
6.3.2	Suggestion: Project Adaptation Questionnaire and Recommendations	186
6.4	Summary: eXtreme Design 1.1	188
7	Discussion	193
7.1	Research Questions Revisited	193
7.2	Research Consequences and Future Challenges	195
7.2.1	ODP Quality Model	195
7.2.2	Tooling Support	198
7.2.3	Methodology Development	200
7.3	Summary of Future Work	201
8	Conclusions	203
	Bibliography	207

A ODP Quality Model Indicators	225
A.1 Documentation Indicators	225
A.2 Model Indicators	227
A.3 In-Use Indicators	232
List of Figures	235
List of Tables	237

Chapter 1

Introduction

This dissertation concerns the development of methods, tools, and measures for Ontology Design Patterns, and specifically Content Ontology Design Patterns. One of the most commonly used definitions of the term *ontology* within the information sciences is attributed to Studer et al., who (building on previous work by Gruber [65]) define an ontology as a “*formal, explicit specification of a shared conceptualisation*” [153, p. 25]. In layman’s terms, it is a commonly agreed upon (*shared*) model of a domain of discourse (*conceptualisation*) that is specific and clear enough that it can be interpreted by a computer (*formal, explicit*). An ontology is thus a type of formal *knowledge model*. Ontologies allow organisations to formally define how they view their information, in turn enabling harmonisation of information systems across the organisation. Software developers 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. Ontologies and ontology-based technology has seen significant adoption, with examples of ontology use ranging from schemas for publishing linked data [16] to biomedical research integration [148] to question answering for TV quiz shows [51], and everything in between.

While in a broad sense such ontologies can be developed in any modelling or object-oriented programming language, unless it is prefixed by some additional identifier, the term is most often reserved for Semantic Web ontologies—that is, ontologies that are defined using a set of standards developed for the future Web by the World Wide Web Consortium (W3C). These standards, which have had a large impact on the ontology research community, include the RDF data model, the RDF Schema extension, and the OWL knowledge representation language. The W3C ontology languages have several advantages over other types of data or knowledge representation languages; as they are community standards they are not tied to any particular vendor, implementation platform or programming language; as they are initially developed from an RDF graph formalism, they can easily

accommodate heterogeneous data; and as they use IRI identifiers, ontologies built using these languages integrate well with other web resources. In this dissertation, unless explicitly mentioned otherwise, the term *ontology* indicates an ontology built using the W3C standards¹.

Ontology engineering is the discipline or trade of developing ontologies. High-quality ontology engineering is costly, as performing it requires the union of rather specific skills—the ontology engineers need to have 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. Alternatively, domain experts and modelling experts might be paired together, each contributing according to their competence, but also at greatly increased cost of development. Given how ontologies are often reused and depended on by many other components in a large system, the risk of design mistakes in ontology modelling needs to be minimised, as such mistakes can be particularly expensive to rectify at a later stage. It is nonetheless not uncommon to see such failures in practice (see e.g., [35]). Driven by these conflicting challenges of reducing ontology engineering costs while maintaining or increasing quality, the Knowledge Modelling research community has over the years put much effort into developing methods and tools for simplifying ontology engineering, with the goal of making the work easy and intuitive enough that a domain expert might perform it efficiently and correctly.

One such method is the reuse of established best practices in the form of *Ontology Design Patterns* (ODPs). The use of *Design Patterns* to describe reusable solutions (an idea first proposed by Christopher Alexander in the field of architecture [2]) has some history in computer science, most notably the *Object-oriented Design Patterns* proposed by the “Gang of Four” [54], and the *Analysis Patterns* developed and discussed by Martin Fowler [53]. The idea of employing the design pattern idea for ontology engineering, in the form of ODPs, was introduced by Gangemi [55] and Blomqvist & Sandkuhl [25] in 2005 (extending ideas by the W3C Semantic Web Best Practices and Deployment Working Group²). ODPs package known good solutions to commonly occurring ontology modelling problems, which can be reused in many different ontology development use cases. The intent is that the use of such ODPs, and appropriate support tooling, will guide ontology engineers (whether experienced or novices) and support them in developing high quality ontologies with greater ease and confidence.

Since their introduction in 2005, ODPs have received quite a bit of research attention, and a community has formed³ based on the developments of these ideas as explored primarily within the NeOn project [133]. Pattern workshops have been held at the largest academic Semantic Web and Knowl-

¹While the findings presented in this dissertation may be applicable to other ontology languages also, the author has not evaluated any such applicability.

²<http://www.w3.org/2001/sw/BestPractices/>

³<http://www.ontologydesignpatterns.org>

edge Modelling conferences, and a number of Ontology Design Patterns have been published. One particularly important result of this work is the development of the eXtreme Design (XD) ontology engineering methodology (see Section 2.4.2 or [131]).

There are several proposed types of Ontology Design Patterns being studied, concerning everything from naming standards to reasoning procedures (see Section 2.4.1 or [133]). Of these pattern types, Content ODPs in particular have received significant attention. Such patterns package commonly recurring ontology features as small and generically reusable building blocks, to be reused by ontology engineers in development. They are, in a sense, analogous to the aforementioned *Analysis Patterns*, in that they emphasise the reusability of the developed domain model, rather than the technical specifics covered by other types of ODPs. Content patterns are intended 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 with fewer modelling errors and inconsistencies. To the author's best knowledge, the validity of the former assumption has not been established, but the latter is supported by some empirical evidence [24, 21].

1.1 Problem

Despite the considerable amount of work that has been published on the topic of ODPs development and use over the course of the last twelve years, there are still gaps in research that are not fully addressed. This dissertation addresses some of these gaps:

- *Lack of knowledge of ODP quality:* While many patterns have been presented, and while patterns are being used in various system development projects, there are few publications documenting and evaluating the effects of using these patterns for different purposes. Even less work has been done on the structure and design of patterns themselves, and consequently, little is known about which qualities or properties of patterns are beneficial in ontology engineering tasks, and inversely, which properties are not helpful or are possibly even harmful in such tasks.
- *Lack of fine-grained method and tool support:* The XD methodology prescribes certain tasks that should be performed in a certain order, but the granularity of these tasks is rather coarse, for instance, “Reuse and integrate selected CPs”, which is a task that is very likely composed of many sub-tasks. There is neither detailed guidance on how

to perform these subtasks, nor sufficient tool support to guide users on what choices to make and the trade-offs that they may imply.

- *Lack of empirical grounding of the XD methodology:* While the XD methodology has been used in several projects and described in several articles, these publications have tended to focus on the resulting ontologies, not evaluation of the methodology itself. Consequently, we lack information on how well the XD methodology works in different scenarios, and which type of adaptations or modifications might need to be made to the method to make it work in different usage contexts.
- *Lack of knowledge of practitioner use cases:* The projects in which ODPs have been used have generally been performed in academic contexts. Consequently, there is little knowledge of the preferences and requirements of non-academic ontologists, nor do we know much about how well suited the XD methodology and ODP support tooling are to ontology engineering projects with such non-academic participants.

Filling the above gaps in research is important both in terms of strengthening the theoretical underpinnings and academic understanding of ODPs and their usages, but also in terms of supporting the uptake of ODPs and, in turn, ontology-based technologies in industry.

1.2 Research Questions

The knowledge gaps discussed in the previous section motivate the research work presented in this dissertation and the overarching objective which is defined as follows:

To develop an understanding of important ODP quality issues, and to develop ODP tooling and usage methodologies as required to support the use of ODP-based ontology engineering, particularly by inexperienced ontologists.

The research questions addressed in this dissertation, derived from the above objective, are as follows:

1. Which ODP features or qualities are important in supporting pattern understanding and use?
2. How can the features and functionality of ODP usage tools be improved to support inexperienced ontologists?
3. How can ODP usage methodology be improved to support inexperienced ontologists?

The first question is treated via the development of an ODP quality model, as described in Chapter 4. The second question is addressed through the development of new methods and tools supporting ODP search and instantiation, in Chapter 5. The third and final question is the subject of Chapter 6, in which experiences of applying the XD methodology are used to develop methodology improvement suggestions. For further details on how the research project was organised and the questions treated, see Section 3.2.

1.2.1 Delimitations

This work focuses exclusively on Content ODPs. In Section 2.4.1 the interested reader may learn about the NeOn typology of Ontology Design Patterns [133] and the other types of ontology patterns that have been proposed. However, in the above research questions, and in the remainder of this dissertation (unless stated otherwise) the terms Ontology Patterns and Ontology Design Patterns, and the ODP abbreviation, all refer to Content Ontology Design Patterns per the NeOn definition.

Note that the term “inexperienced ontologist” as it is used in the research questions does not refer to someone who is necessarily completely untrained in computer technology or programming; rather, it refers to someone who is not well acquainted with ontology engineering methods, tasks and tooling. Focusing the research project on this user group is a deliberate decision, stemming from the author’s interest in enabling greater industry adoption of semantic and ontology-based technologies.

The attentive reader will note that none of the research questions could realistically be answered in an exhaustive manner within the scope of a PhD project. The work presented in this dissertation does not aim for the sort of completeness required to provide exhaustive answers—instead, the work is inductive in nature, with the empirical data gathering and analysis performed within the project contributing new pieces of knowledge to different facets of ODP use, but not necessarily providing absolutely delineated, complete, and *certain* answers to each research question. For further discussion on the merits and consequences of inductive research, the reader is referred to Chapter 3.

1.3 Contributions

This work contributes to new academic knowledge within understudied areas, namely the real-world usability of ODPs, ODP support tooling, and ODP methods. Additionally, the project has addressed practical issues that are of importance to industry and will support industry uptake of semantic technologies, specifically the improvement of those same ODPs, ODP support tooling, and ODP usage methods. The contributions, and the sections of this dissertation in which they are detailed, are summarised below.

- Contributions to Ontology Engineering Research:
 - A conceptual understanding of quality as it relates to Ontology Design Patterns (Section 4.4.1).
 - A catalogue of quality characteristics and quality indicators compliant with the above, and methods for measuring the latter (Sections 4.4.2 and 4.4.3, and Appendix A)
 - Increased understanding of the requirements that inexperienced ontologists have on ODP usage methods and tools (Section 5.3).
 - Improved algorithms and heuristics for finding, specialising and instantiating ODPs (Sections 5.1, 5.2, and 5.3).
 - A partial evaluation of the XD methodology in real-world ontology engineering projects involving non-academic ontologists, and updates to the XD methodology based on said evaluation (Chapter 6, primarily Section 6.4).
- Contributions to Ontology Engineering Practice:
 - Development tools and associated services designed to support key ontology engineering tasks with ODPs (Section 5.4).
 - Recommendations on improvements to the features and data quality of the community ODP portal, supporting increased use of ODPs and ODP-based tooling (Section 7.2.1).
 - A set of recommendations on which values that ODP quality indicators should assume, aiding inexperienced ontologists in selecting ODPs that are compliant with their project requirements (Appendix A).

1.4 Summary of Publications

The following peer reviewed workshop papers, conference papers and anthology contributions were produced and published during the author's PhD project. They detail many of the project results that are also presented in this dissertation. The papers are listed in an ascending chronological order, and are each accompanied by a brief description of how they contribute to the dissertation.

- K. Hammar, F. Lin, and V. Tarasov. Information Reuse and Interoperability with Ontology Patterns and Linked Data. In W. Abramowicz, R. Tolksdorf, and K. Wecl, editors, *BIS 2010: Business Information Systems Workshops*, number 57 in Lecture Notes in Business Information Processing, pages 168–179. Springer, 2010

- **Contribution:** The paper discusses the application of semantic technologies and ODPs in a project in the Information Logistics domain. In this project we observe some issues relating to the use of `owl:imports`, observations that contribute to the ODP quality model developed and presented in Chapter 4. The author’s contribution to the work consists of both participation in practical modelling, and having authored the majority of the paper.
- K. Hammar and K. Sandkuhl. The State of Ontology Pattern Research: A Systematic Review of ISWC, ESWC and ASWC 2005–2009. In E. Blomqvist, V. K. Chaudhri, O. Corcho, V. Presutti, and K. Sandkuhl, editors, *Proceedings of the 2nd International Workshop on Ontology Patterns – WOP2010*, number 671 in CEUR Workshop Proceedings, pages 5–17, 2010
 - **Contribution:** The paper presents a systematic literature review covering ODP-related papers presented at the top three Semantic Web conferences during 2005–2009. The findings indicate that many papers in this field are lacking in empirical validation, and that while ODPs are being presented and used, they are not being sufficiently studied as IT artefacts of their own; consequently, not enough is known about what makes for an efficient, effective, and usable ODP. These findings motivate the direction this PhD project has taken and the research questions chosen. The author’s contribution to this paper includes the majority of both research and authoring.
- K. Hammar. DC Proposal: Towards an ODP Quality Model. In L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, and N. Noy, editors, *The Semantic Web – ISWC 2011*, volume 2 of *Lecture Notes in Computer Science*, pages 277–284. Springer, 2011
 - **Contribution:** This paper, presented and discussed at the ISWC 2011 Doctoral Consortium, describes an early version of this PhD project, including initial method choices. It also includes the first draft of the ODP quality metamodel (discussed further in Chapter 4).
- K. Hammar. The State of Ontology Pattern Research. In L. Niedrite, R. Strazdina, and B. Wangler, editors, *Perspectives in Business Informatics Research: Associated Workshops and Doctoral Consortium*, pages 29–37. Riga Technical University, 2011
 - **Contribution:** An updated version of the similarly named paper discussed above, extending the dataset studied to include additional conferences and journals, and including the years 2010–2011. The findings confirm those of the previous paper, and the impact on this PhD is similar.

- K. Hammar. Modular Semantic CEP for Threat Detection. In L. Villavargas, L. Sheremetov, and H.-D. Haasis, editors, *ORADM 2012: Operations Research and Data Mining Workshop Proceedings*, Cancun, Mexico, 2012. National Polytechnic Institute
 - **Contribution:** The paper discusses the use of ODPs as plug-gable configuration modules for a Complex Event Processing system within the IMSK project (see Section 3.2.4). This scenario and the development of the ontologies and the technology platform described in this paper were the context for the subsequent paper described just below.
- K. Hammar. Ontology Design Patterns in Use: Lessons Learnt from an Ontology Engineering Case. In E. Blomqvist, A. Gangemi, K. Hammar, and M. C. Suárez-Figueroa, editors, *Proceedings of the 3rd Workshop on Ontology Patterns*, number 929 in CEUR Workshop Proceedings, 2012
 - **Contribution:** This paper presents an observational case study of ODP usage in the IMSK project. Key findings include several features that users prefer or dislike in ODPs, as well as recommendations on improvements to the community ODP portal (see Chapter 4 for further details).
- K. Hammar. Reasoning Performance Indicators for Ontology Design Patterns. In A. Gangemi, M. Gruninger, K. Hammar, L. Lefort, V. Presutti, and A. Scherp, editors, *Proceedings of the 4th Workshop on Ontology and Semantic Web Patterns*, number 1188 in CEUR Workshop Proceedings, 2013
 - **Contribution:** The paper surveys existing literature on performance indicators in ontologies that are also applicable to ODPs, and studies how those indicators are expressed in published ODPs, suggesting recommendations on design of ODPs that perform efficiently (contributes to Chapter 4).
- K. Hammar. Ontology Design Patterns: Improving Findability and Composition. In V. Presutti, E. Blomqvist, R. Troncy, H. Sack, I. Papadakis, and A. Tordai, editors, *The Semantic Web: ESWC 2014 Satellite Events*, number 8798 in Lecture Notes in Computer Science, pages 3–13. Springer, 2014
 - **Contribution:** This paper discusses challenges related to finding suitable ODPs and to composing ODP-based ontology modules with a project under development, and proposes some partially evaluated solutions to overcome these challenges. The solutions were later integrated into the XDP tools discussed in Chapter 5.

- K. Hammar. Ontology Design Pattern Property Specialisation Strategies. In K. Janowicz, S. Schlobach, P. Lambrix, and E. Hyvönen, editors, *EKA W 2014: Knowledge Engineering and Knowledge Management*, number 8876 in Lecture Notes in Computer Science, pages 165–180. Springer, 2014
 - **Contribution:** The paper presents an analysis of different strategies by which ODPs are typically specialised, evaluates the effects of this strategy choice, and suggests tool improvements to support these different strategies. The work contributes to the XDP tools discussed in Chapter 5.
- K. Hammar. Ontology Design Patterns in WebProtégé. In S. Villata, J. Z. Pan, and M. Dragoni, editors, *Proceedings of the ISWC 2015 Posters & Demonstrations Track*, number 1486 in CEUR Workshop Proceedings, 2015
 - **Contribution:** This paper demonstrates the XDP tooling developed based on the findings in Chapter 5.
- E. Blomqvist, K. Hammar, and V. Presutti. Engineering Ontologies with Patterns – The extreme Design Methodology. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns - Foundations and Applications*, volume 25 of *Studies on the Semantic Web*, pages 23–50. IOS Press, 2016
 - **Contribution:** This anthology chapter describes the eXtreme Design ontology engineering methodology. The author’s contribution to the chapter covers the recent methodology developments that are in this dissertation presented in Sections 5.3 and 6.3.
- K. Hammar. Quality of Content Ontology Design Patterns. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns - Foundations and Applications*, volume 25 of *Studies on the Semantic Web*, pages 51–71. IOS Press, 2016
 - **Contribution:** This anthology chapter summarises, in condensed format, the ODP quality model which is the subject and result of Chapter 4.
- K. Hammar, E. Blomqvist, D. Carral, M. Van Erp, A. Fokkens, A. Gangemi, W. R. Van Hage, P. Hitzler, K. Janowicz, N. Karima, et al. Collected Research Questions Concerning Ontology Design Patterns. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns - Foundations*

and Applications, volume 25 of *Studies on the Semantic Web*, pages 189–198. IOS Press, 2016

- **Contribution:** This anthology chapter summarises areas of future research for the ODP community, as submitted by researchers active in the field. The author contributed several research questions, and compiled the joint chapter.
- K. Hammar. Template-Based Content ODP Instantiation. In K. Hammar, P. Hitzler, A. Krisnadhi, A. Lawrynowicz, A. G. Nuzzolese, and M. Solanki, editors, *Advances in Ontology Design and Patterns*, Studies on the Semantic Web. IOS Press, Forthcoming 2017
 - **Contribution:** The paper presents an alternate template-based approach to ODP instantiation, evaluates the effects of the proposed approach, and suggests and evaluates a concrete implementation method. The work contributes to the further development of the XDP tools discussed in Chapter 5.
- N. Karima, K. Hammar, and P. Hitzler. How to Document Ontology Design Patterns. In K. Hammar, P. Hitzler, A. Krisnadhi, A. Lawrynowicz, A. G. Nuzzolese, and M. Solanki, editors, *Advances in Ontology Design and Patterns*, Studies on the Semantic Web. IOS Press, Forthcoming 2017
 - **Contribution:** This paper investigates user preferences and established practice for ODP documentation, finding a set of key documentation fields that ODPs need to display, and further finding that ODPs published in the community ODP portal often lack these documentation fields. This work contributes to the ODP Quality Model presented in Chapter 4. The author’s contribution consists of having designed and performed two out of the three user surveys discussed, and having authored significant portions of the paper.

Additionally, an early version of the ODP Quality Model which is the main topic of Chapter 4 is presented in the author’s Licentiate thesis, *Towards an Ontology Design Pattern Quality Model* (Linköping Studies in Science and Technology, Licentiate Thesis No 1606).

1.5 Dissertation Outline

The remainder of this dissertation is structured as follows:

- Chapter 2 introduces basic concepts with which the reader may wish to familiarise themselves, including knowledge modelling theory, semantic technologies, ontology evaluation, ontology engineering methods, and Ontology Design Patterns.

- Chapter 3 discusses issues of method in computer and information systems research, gives an overview of how such methods have been applied in this dissertation to answer the research questions, and characterises the cases from which the bulk of the empirical material this dissertation depends on originates.
- Chapter 4 describes the work performed to answer the first research question, and the results obtained in so doing, namely an ODP Quality Model.
- Chapter 5 details the work performed to answer the second research question, and the results of this work, namely the ODP-based XDP extension to the WebProtégé ontology engineering environment.
- Chapter 6 presents the work performed to answer the third research question, and the results of that work, namely, a set of extensions to the eXtreme Design method.
- Chapter 7 discusses the results that were obtained, the research process that was followed, and areas open to future work.
- Chapter 8 concludes the dissertation by revisiting the research questions and summarising the developed answers to those questions.

Chapter 2

Background and Related Work

The following chapter is intended for the reader who is new to knowledge-based systems, ontology engineering, the Semantic Web, or Ontology Design Pattern research. It provides an overview of concepts, technologies and research in the field, with a special focus on topics that are relevant to the work presented in this dissertation.

2.1 Knowledge Modelling and Ontologies

Even though some of the technical standards for using ontologies on the Semantic Web were developed only recently, 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 introduced, and the modern-day standards of RDF, RDFS, and OWL are then briefly described.

2.1.1 Data, Information, and Knowledge

In Chapter 1, ontologies are introduced as formal knowledge models that can be used in information systems. While the terms *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 [1] and described by Bellinger et al. [14]. In this model, displayed in Figure 2.1, several different levels of understanding of phenomena are defined:

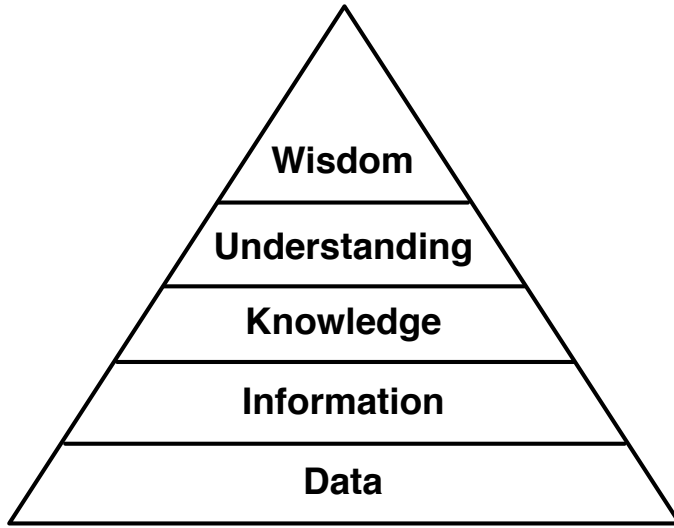


Figure 2.1: Ackoff's Knowledge Hierarchy.

- *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 in which 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 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.

At the time of writing, treating the two highest levels of this model, understanding and wisdom, is outside of the realm of the computationally feasible, even if we knew 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 Vladimirov [162] and Stacey [150], 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 [122], 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 dissertation and in his research, the author sides with the latter perspective. Data is considered to be simple raw facts without context; information is data that is linked to provide a greater understanding; knowledge is information that is reasoned upon by either a human or a machine, 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 A6 is a car*”, or “*this type D battery puts out 1.5 volts*” [6].

In any computer system dealing with information or knowledge this distinction is made between the general (a database schema, a vocabulary, a class definition) and the specific (database rows, RDF instance data, instantiated objects). 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 [65, 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 [153] ontology definition from Chapter 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 dissertation, 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 description logic axioms that define classes, properties, and individuals [64]. The following sections describe these constructs 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. For the sake of simplicity, uses of the core properties defined in the RDF, RDFS, or OWL standards (introduced in Section 2.1.4) are displayed in the figures as in-line labels.

Classes

Classes are a way of grouping things that are similar in some respects. 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 intensional 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 the type of software known as a

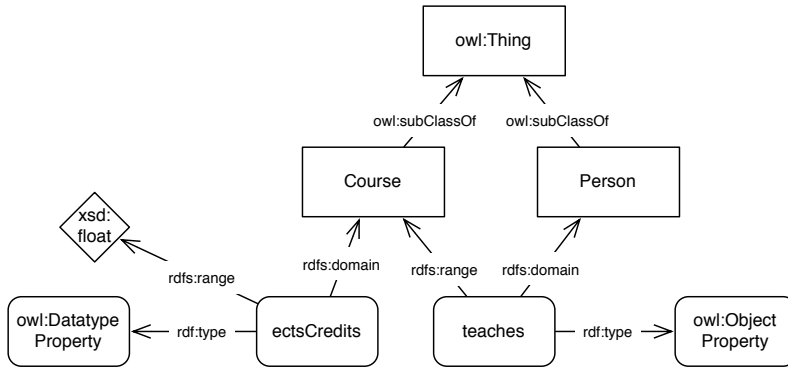


Figure 2.2: Course ontology example.

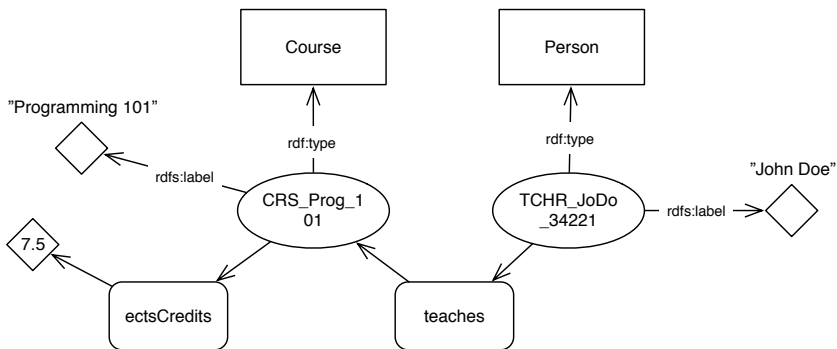


Figure 2.3: Course data expressed using the ontology in Figure 2.2.

reasoner is to sort individuals into classes based on the properties that they exhibit and the intensional definitions of the classes [125, 5].

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 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 is 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* [125, 5].

In other knowledge modelling languages classes are known varyingly as concepts, types, categories, etc. In this dissertation, 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) [5]. Other languages, such as Protégé-Frames, do not distinguish between the two types of properties, but treat both as simple slots in a class definition that can be filled 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 instances of 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* (from the XML Schema Datatypes¹ definitions, as used by W3C ontologies). The latter is an object property with the domain *Person* and range *Course*.

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 assertional knowledge rather than as terminological knowledge, there are some cases when it makes sense to refer to individuals in

¹<https://www.w3.org/TR/xmlschema-2/>

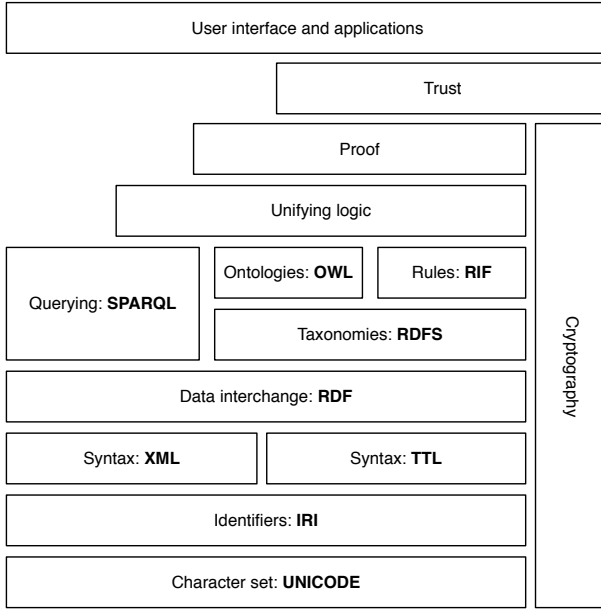


Figure 2.4: The Semantic Web layer cake (adapted from [15]).

an ontology. One such case is when defining classes extensionally, that is, by defining 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 *entities*. 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

For a long time in the 1980s and 90s there were 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 [17], via which humans and computers alike could find, consume, and reason using published knowledge. What Berners-Lee saw was that this vision of the future Web could never come to fruition unless decentralised and open knowledge rep-

resentation 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 each holding different 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 introduces 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, and again in 2014. 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 [103, 37].

The RDF data model is based on graphs, as opposed to the tuples that underlie traditional relational data models. In RDF, a data graph is constructed by the union of a number of three part assertions called *triples*. 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 an entirely non-standard simplified syntax intended for illustrative purposes) the six triples making up the graph displayed in Figure 2.3. `Programming_101` and `John_Doe` are subjects, `rdf:type`, `rdfs:label`, `ectsCredits`, and `teaches` are predicates, and `Course`, `“Programming 101”`, `7.5`, `Person`, `“John Doe”`, and `Programming_101` (when used on the right hand side of a triple) are objects.

Listing 2.1: RDF triples example

<code>Programming_101</code>	<code>rdf:type</code>	<code>Course</code>
<code>Programming_101</code>	<code>rdfs:label</code>	<code>“Programming 101”</code>
<code>Programming_101</code>	<code>ectsCredits</code>	<code>7.5</code>
<code>John_Doe</code>	<code>rdf:type</code>	<code>Person</code>
<code>John_Doe</code>	<code>rdfs:label</code>	<code>“John Doe”</code>
<code>John_Doe</code>	<code>teaches</code>	<code>Programming_101</code>

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 it links. However, the semantics of typing in pure RDF are rather vague, and one must go to more complex languages such as RDFS and OWL to model class extensions as discussed in Section 2.1.3.

In RDF all resources are referenced using IRIs (not shown in the example), enabling global lookup of distributed knowledge via HTTP, FTP, or other distribution mechanisms supported by the IRI standard. To simplify modelling, namespace prefixes are often 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), each of which are defined in their own namespaces.

The RDF syntax standards describe how these triples are serialised into files. There are several such RDF serialisations, designed for different use cases. The original standard, XML/RDF, 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. In this dissertation, to the extent that RDF data is shown, the more recent Turtle syntax will be used, due to its superior readability [134].

RDFS and OWL

The RDF Schema (RDFS) standard, released along with the second generation of RDF in 2004 (and updated in 2014), defines 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 [27, 28]:

- *Classes*: Defines the concept of classes to which resources may belong, strengthening the definition of the RDF type predicate.
- *Subclasses*: Classes may be subsumed by superclasses, in which case all instances of the subclass are also instances of the superclass.
- *Domain*: Defines the class of instances that may act as subjects to a certain predicate.
- *Range*: Defines the 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 [104], the RDFS language does however have some restrictions in expressivity that prevent 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 to provide better support for such higher-level expressiveness. OWL is built on both RDF/RDFS, and on description logic (a derivative of predicate logic) foundations—in fact, classes, properties, and individuals in OWL are defined formally by way of description logic axioms. From the latter foundation the OWL language has inherited the use of an *Open World Assumption*, that is, the assumption that absence of fact does not imply negation of fact. This makes OWL logic and ontologies particularly well-suited to modelling situations where knowledge is distributed over a network where not all nodes are reachable at all times, such as the Internet. However, the open world assumption also comes with some drawbacks, including the inability to model default values or relations. Some key features of OWL include [118]:

- *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 IRIs at different knowledge sources, but are in fact semantically equivalent.
- *disjointWith*: Defines class disjointness, that is, that two defined classes may not have any joint individuals.
- *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 IRIs but in fact refer to the same information.
- *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 they allow descendant nodes many steps down the tree to be inferred to be related to higher nodes.
- *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. Several new features (keys, property chains, datatype restrictions, etc.) were added to the standard when it was updated in 2009 [168].

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 various 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 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 the types of gains that could be had by integrating, reusing, or searching their information, which an ontology engineer would not necessarily have. In deploying different types of reasoning systems, whether it be for purposes of Complex Event Processing or ubiquitous computing, system users and administrators being able to develop the ontologies that govern system behaviour by themselves, would be superior to handing off such configuration tasks to an ontology engineer.

2.2.1 Ontology Types

When classifying or structuring ontologies, one approach is to organise them by intended usage domain. This is likely the result of different academic disciplines picking up ontology modelling for different purposes. When dealing with reuse and patterns, such a view of 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 [67] is of another kind, differentiating between ontologies based on their level of generality. 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 specific 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 academia, for instance) irrespective the task for which they are intended. 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 particular tasks within particular domains, and therefore often reuse and build upon both domain and task ontologies.

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 offers 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 [16]:

1. Use IRIs as names for things
2. Use HTTP IRIs so that people can look up those names.
3. When someone looks up an IRI, provide useful information, using the standards (RDF*, SPARQL)
4. Include links to other IRIs, 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 institutions² have recognised that by allowing users and customers 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 have produced themselves. 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 required, strictly speaking, 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 can use existing ontologies rather than individually constructing schemas for their data. As an example of this, the FOAF ontology³ is very widely used when publishing data about individuals or organisations.

2.2.3 Semantic Search

A common problem for knowledge workers is finding the correct information needed for performing some task or fulfilling some role. The two main options are all too often to either search through some file server directory structure step by step 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 allows the knowledge worker to query over the information content of the documents in question. Semantic search methods aim to solve this problem in different ways.

²Data.gov, the EU Open Data Portal, the Wikimedia movement, etc.

³Friend-Of-A-Friend, <http://www.foaf-project.org>

Semantic fact search

The semantic search solution by Guha et al. [70] allows users to search over the Semantic Web to find knowledge triples related to an entity or concept. Their basic approach to semantic fact search, of first finding a canonical IRI corresponding to a search string, and then querying known knowledge bases to aggregate more RDF triples involving this IRI, is still in use in modern solutions. Uren et al. discuss approaches and methods for semantic search extensively in [166]. 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. This is the simplest type of semantic query. Relation-based search attempts to find the path connecting two RDF resources, to learn how two known concepts or individuals are connected in a dataset. Parametrised search uses templates with parameters that can be applied to an RDF graph to “stamp out” those parts of the RDF graph that are consistent with the parametrised search query.

Text-based search with annotations

When using an annotations-based approach, the documents over which a 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 [101]. Their method allows users to query for both instance data extracted from documents and for documents 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 and then bring up the documents related to those facts. Lei et al. [106] present a system that uses a simple Google-like search interface. This interface makes use of a controlled natural language such that users can pose queries in pseudo-English, which the system translates into formal queries. The 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 systems make use of information extraction techniques to retrieve metadata from published documents. Such extraction can be simplified significantly if the source documents comply with some metadata structure to begin with. The schema.org⁴ vocabulary (backed by Google, Microsoft, Yahoo and Yandex) standardises such a metadata structure by providing a set of simple vocabularies for web content.

⁴<http://schema.org/>

2.2.4 Reasoning Tasks

The description logic foundations of Semantic Web ontologies make them suitable for a variety of uses where the logic consequences of a certain set of data or knowledge need to be computed. This type of computation is most often performed using a Semantic Web reasoning engine, which is typically capable of tasks such as consistency checking, subsumption calculation, individual realisation, and concept satisfiability checking. The following sections provide some illustrations of such advanced usages of ontologies and semantic technology.

Complex Event Processing

Luckham and Frasca [114] introduce the notion of Complex Event Processing (CEP), in which patterns based 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. CEP is utilised in many areas, from improving operational efficiency in healthcare [172] to dynamic adaption of business process models [87]. However, as indicated by Anicic et al. in [4], traditional CEP approaches have some drawbacks, particularly in terms of recognising events using background knowledge. To overcome these limitations, [4] suggests the use of Semantic CEP, in which background knowledge is encoded into knowledge bases that are accessed by a rules engine to support CEP. Another approach to enabling semantic processing of time-indexed data is proposed by Barbieri et al. in [10] and further developed in [9] and [8]. They propose an extension of the SPARQL query language, enabling continuous querying over timestamped RDF graphs using configurable sliding windows. They also develop support for reasoning over such sliding windows.

Ubiquitous computing

Berners-Lee's original vision for the Semantic Web [17] exemplifies the usage of a meaningful machine-interpretable Web through a ubiquitous computing scenario. Several systems have been developed that try to fulfil this vision, see for instance [135] and [33]. 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. 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 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 dissertation does not have enough room to cover and discuss all of them, a subset of commonly mentioned and discussed methods is presented 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 in developing Semantic Web ontologies in their own, which as we will see in the next section 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 (sometimes abbreviated CQs). Competency questions are introduced by Gruninger and Fox [66] as a set of problems that the logic axioms of an ontology must be able to represent and solve. In [66] several such competency questions are given as examples, including planning questions (*“what sequences of activities must be completed to achieve some goal?”* [66, 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?”* [66, 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 [49]. 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 more general types of issues). Fernández et al. suggest, based largely on the authors’ own experiences of ontology engineering, an ontology lifecycle consisting of six 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*.

To implement 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 whatever implementation language that is eventually used.
4. *Integration:* 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, rather it is 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) [157] is, similarly to METHONTOLOGY, a methodology 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 [157] denoted the *Knowledge Meta Process*).

The method prescribes a set of sequential phases: *Kick-off*, *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 the input into a new *Kick-off* phase.

OTKM requires collaboration between domain experts and ontology engineers in the *Kick-off* 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 as sufficiently covering all requirements. In the subsequent *Refinement* phase an ontology engineer formalises the initial semi-formal model into a real ontology on their own, without aid of a domain expert. Once the ontology engineer is satisfied that the developed ontology fulfils the 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 ([157] 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 organisational 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. [129, 128], 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 ontology users who have direct personal knowledge of the specific uses to which the ontology will be put. These implemented user adaptations may at times need to be back-ported into the core ontology for maintenance and evolution reasons, 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 analyses 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. To support these steps a collaborative ontology engineering environment is required.

In evaluating this approach in two case studies, Pinto et al. [129, 128] 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*” [128, 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.

The NeOn methodology [154] is not actually a single methodology in the sense that it encodes a single recommended workflow and set of activities. Instead, NeOn provides guidance and recommendations concerning a variety of different processes and activities that might be involved in an ontology engineering project. This approach means that different aspects of the NeOn methodology can be employed regardless of whether the ontology engineering project aims to develop a new ontology from scratch, to re-engineer an existing (ontological or non-ontological) resource, to align ontologies, or for that matter, to reuse ODPs.

For this purpose, the NeOn methodology provides:

- A glossary of terms relevant in ontology engineering processes and activities. In total the glossary covers 58 such terms.

- A set of nine scenarios exemplifying how some of the processes and activities from the glossary might be combined to achieve some ontology engineering objective (e.g., *Reusing and re-engineering non-ontological resources* or *Restructuring ontological resources*).
- Two life cycle models describing how these processes and activities might be organised into project phases (the models cover the *waterfall* and *incremental-iterative* approaches).
- Methodological guidelines and recommendations on how to perform some of the processes and activities from the glossary.

The first three of the above contributions present recommendations and discussions on a macro level, discussing which overall processes need be implemented and how they might be organised to achieve some goal. In contrast, the guidelines provided by the last contribution are more specific and instead discuss more concrete methods, tasks, and algorithms. These latter guidelines are presented in the form of individual chapters within [155]. One such chapter specifically concerns the use of ODPs—that chapter describes and prescribes the use of the eXtreme Design methodology which is also studied within this dissertation (and which is presented in Section 2.4.2).

2.3.4 SAMOD

SAMOD [126], or *Simplified Agile Methodology for Ontology Development*, is a recently developed methodology that builds on and borrows from test-driven and agile methods (in particular eXtreme Design).

SAMOD emphasises the use of tests to confirm that the developed ontology is consistent with requirements, and prescribes that the developer construct three types of such tests: *model tests* (testing the ontology module’s terminological definitions against the requirements scenario, and ensuring that the terminological definitions are consistent), *data tests* (testing that the examples provided in the requirements scenario are covered by the ontology module’s assertional definitions, and that those assertional definitions are compliant with the terminological definitions), and *query tests* (testing whether the SPARQL queries intended for use with the ontology module are well-formed and can be executed against the ontology module yielding such results as are expected based on the examples provided in the requirements scenario).

The SAMOD process is very light-weight, consisting of three main steps that are repeated in a development loop that is iterated once for every requirements scenario:

1. Collect requirements for one usage scenario, formalise those requirements, construct tests based on the requirements, and construct an ontology module covering the requirements and fulfilling the tests.

2. Merge the ontology module (covering a single scenario) and its tests with main branch ontology under development (covering several scenarios).
3. Refactor the main branch under ontology as needed.

After each of these steps, all the tests defined for the module and/or main branch ontology are executed. The developer does not proceed to the next step until all tests are passed.

SAMOD defines two participant roles: *domain experts* and *ontology engineers*. The two roles work jointly on collecting and formalising requirements, but SAMOD explicitly requires that once this work is done, the remainder of the work within the first step, and the two subsequent steps, should be performed by ontology engineers only.

SAMOD also requires that the developer adhere to certain design principles:

- Develop small ontologies or ontology modules, for the sake of keeping development and debugging reasonably simple.
- Reuse knowledge in the form of ODPs whenever possible.
- Employ a middle-out development strategy, that is, start working with ontology entities that are of neither too generic and abstract nor too specific and concrete.
- Use self-explanatory entity naming on IRI level: names should be human readable, should be capitalised or in camel-case notation, properties should use verbs, etc.

Finally, SAMOD suggests that ontology documentation and ontology alignment to other resources be performed not when developing each module within the first step of the development loop, but rather at the end of the loop, when refactoring the main branch to accommodate the newly merged ontology module.

2.3.5 Ontology Development 101

The Ontology Development 101 guide by Noy and McGuinness [123] 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 dissertation 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 (which give 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 IRI 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 most ontology engineering tools and methods are geared towards this category of developers, 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, that is, the overall phases or large granularity tasks that need to be performed in an ontology engineering task. Except for 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 [55] and

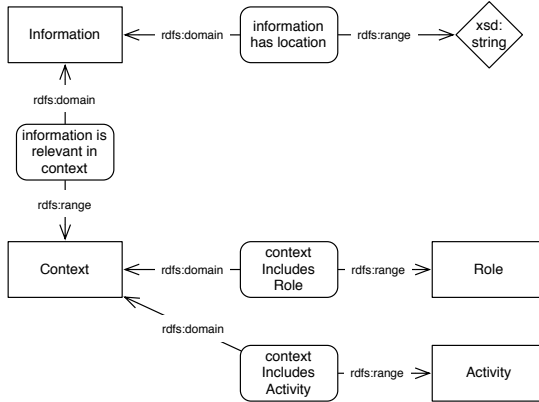
Blomqvist and Sandkuhl [25]. The former defines such patterns by way of the characteristics that they display, including examples such as “[an ODP] is a template to represent, and possibly solve, a modelling problem” [55, p. 267] and “[an ODP] can/should be used to describe a ‘best practice’ of modelling” [55, 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 might look like, see Figure 2.5.

Gangemi [55] 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 [25] on the other hand support the need for patterns by referring to the potential for reuse that they bring, particularly in automatic or semi-automatic ontology engineering scenarios. In subsequent work [133, 18, 60] these two sets of motivations for Ontology Design Patterns have come together, such that at the time of writing, patterns are considered to produce benefits with regard to both reusability and guidance. Additionally, as pointed out by Blomqvist in [18] 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 for pattern use, which can vary, as shown above. If one considers reuse to be the main and only reason 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 reasons, then the pattern documentation may be of greater importance. From a philosophical perspective it can be argued that both parts of the pattern are 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 dissertation 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 a fair amount of research, see for instance the deliverables of the EU FP6 NeOn Project⁵ [133, 40] and the work presented at instances of the

⁵<http://www.neon-project.org/>



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/ff5/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 2.5: Context Dependant Information ODP.

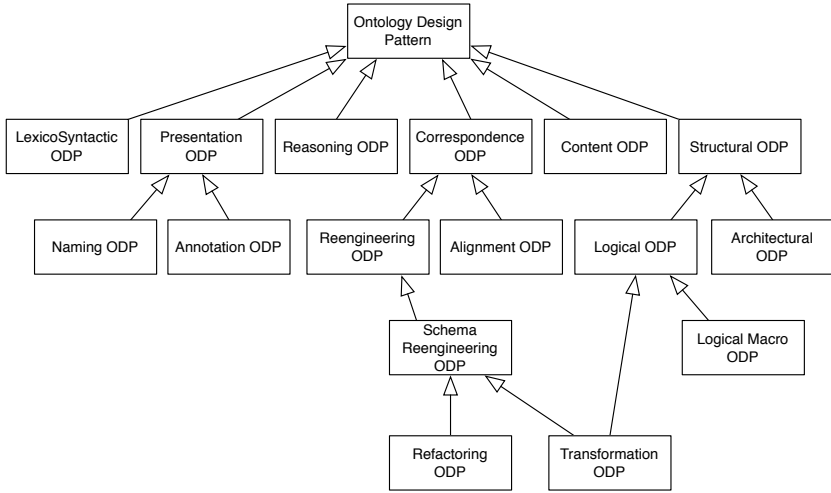


Figure 2.6: NeOn ODP typology (adapted from [133]).

Workshop on Ontology Patterns⁶ [26, 19, 20, 58, 42, 23] at the International Semantic Web Conference. To the author’s best knowledge there are 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 inexperienced users [21, 44].

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 [133] shown in Figure 2.6⁷, and the eXtreme Design collaborative ontology development methods, based on pattern use [40]. The typology of patterns has been developed further within the OntologyDesignPatterns.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 practices in reasoning, naming, transformation, content modelling, etc. In this view, certain categories of patterns are subcategories of one or several other categories. While the work in this dissertation concerns only Content ODPs, all the top-level

⁶<http://ontologydesignpatterns.org/wiki/WOP:Main>

⁷The 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.

pattern categories from the NeOn typology are presented below for the sake of completeness, 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 that 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 reengineering 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 alignment are 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 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, [167] presents 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 [85] mapped to ontology constructs.

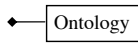
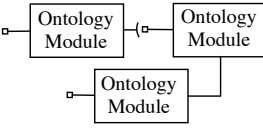
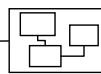

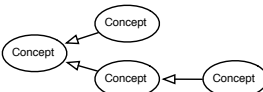


Abstraction	Granularity	Illustration of level
Application pattern	Complete ontology	 <p>Ontology requirements and interface, possibly described in a software architecture description language.</p>
Architecture pattern	Complete ontology	 <p>Overall ontology organisation and parts, possibly described in 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:ObjectProperty> ... </owl:ObjectProperty> </pre> <p>Representation of parts of an ontology in an ontology representation language.</p>
	Elements of the ontology	<pre> <owl:Class rdf:about=""> <rdf:subClassOf rdf:resource=""> </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 2.7: Blomqvist's ODP typology (source: [18]).

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 [18] presents a different typology based on the level of abstraction and granularity of the reusable solution. According to this categorisation structure, shown in Figure 2.7, 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 (this meaning of the term Ontology Design Pattern differs from its use in the NeOn typology and in the rest of this dissertation). 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, that is, 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” [132, p. 83]. The method is influenced by the eXtreme Programming (XP) agile software development method, and like that method, 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 [131].

Conceptually, XD describes approaches for selecting ODPs 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 during a project. The solution space is the set of reusable solutions to common problems, that is, ODPs. Included in each pattern is a description of the Generic Use Case (GUC) in which

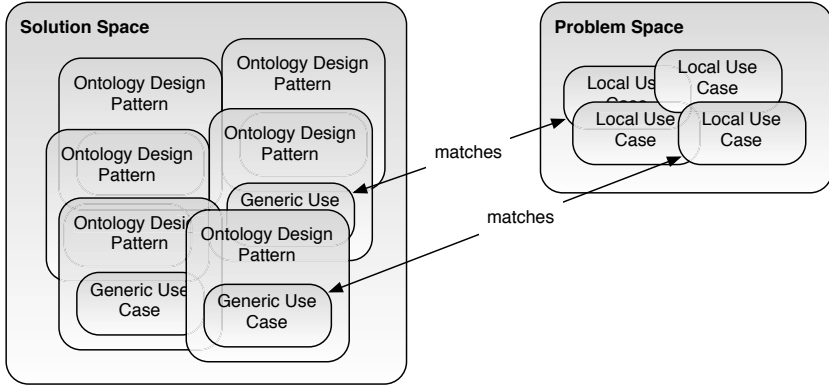


Figure 2.8: XD pattern selection approach (source: [132]).

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 2.8 [132, 131].

The proposed selection method is appropriate for finding patterns that satisfactorily solve a certain 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. However, it does 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 situation. The right choice could then depend on non-functional requirements on the ontology (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, if there is an example ontology using it, etc.). In this scenario, an ODP quality model could 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 2.9. 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 who understands 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

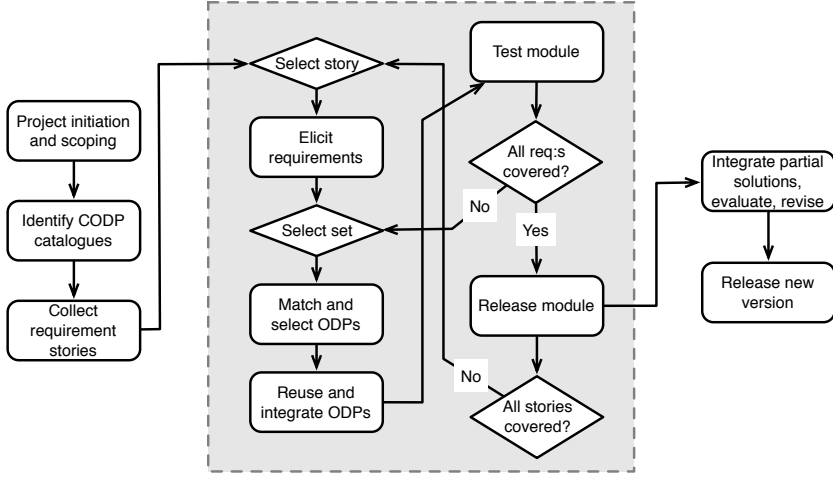


Figure 2.9: XD workflow (adapted from [132]).

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 2.5. 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 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 [132, 131].

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 is found, adapted, and integrated, and so on. Once all requirements associated with 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 [132, 131].

Deliverables from the NeOn project prescribe ways of finding or developing patterns [133], 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 in particular they leave out a discussion on the desired or beneficial attributes of patterns (while a set of pragmatic ODP features is presented in [60], these features are quite general and do not provide measurable design criteria). The choices of the type of pattern to create, which 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 (such as the one developed and presented in Chapter 4 of this dissertation) would be useful in guiding the developer in producing high-quality patterns.

2.4.3 Other Perspectives on ODPs

Falbo et al. [47] argue for a different perspective on ODPs and their use, and Ruy et al. [137] extend upon this argument. Per this view, ODPs⁸ can be divided into those that are extracted from ontologies covering foundational concepts (FOPs), and those that are extracted from domain-related ontologies (DROPs). Neither FOPs nor DROPs have language-specific implementations (e.g., OWL building blocks)—rather, they attempt to solve a modelling issue in a reusable manner, regardless of the technology stack used. Per this perspective, FOPs would typically be reused *by analogy*, while DROPs would typically be reused *by extension*. The former is analogous to Template-based ODP instantiation as discussed in Section 5.3, whereas the latter is analogous to Specialisation-based ODP instantiation.

While [47] and [137] exemplify both types of ODP use, they do not evaluate the consequences of either approach, nor do they propose any specific method including concrete steps a person (or machine) should take to perform either. Further the *FOP-analogy* and *DROP-extension* pairing might not necessarily hold in all cases—in fact, in many cases it would be quite useful for interoperability purposes to extend foundational concepts—while cloning (i.e., reuse by analogy) may be very useful when adapting a domain-specific ODP to a related domain.

⁸The cited works argue that ontology patterns that include concepts or content are disjoint from patterns that concern design issues, so they prefer the use of the term *Ontology Conceptual Pattern*, rather than *Ontology Design Pattern* or *Content Ontology Design Pattern*—but for the sake of not confusing the reader unduly, here we will stick with the abbreviation ODP.

Yet another approach to formalising and using ODPs was developed in the CO-ODE project. This approach builds on the Ontology Pre-Processing Language (OPPL), a macro language that was initially designed to simplify rapid transformation of large ontologies [45]. The OPPL macro engine can add and remove ontology entities and axioms based on variables that are selected from and conditions that are evaluated against the existing ontology - in this way changes to ontology structure can easily be performed repeatedly with precision.

The same technique has been extended to formalise ODPs as OPPL macros [93]. These ODP macros contain unbound variables that the ontology developer fills with existing or new ontology entities before executing the OPPL engine that instantiates the ODP structures into the target ontology. Additionally, the ODP version of the OPPL language and tooling adds features and syntax required to better support ODP use cases, including user-friendly textual representation of ODP macros, syntax allowing macros to call one another, and annotation properties to embed metadata about ODP macro usage in target ontologies.

The OPPL-based approach to ODP use is technically impressive and from a feature perspective it would likely complement the XD method very well. All the same, OPPL-based ODP use has seen limited uptake in practice. This is likely due to some drawbacks that the use of OPPL entails, namely the need to learn yet another language in addition to RDFS/OWL and the lack of tool support for developing and maintaining OPPL ODPs.

2.5 Quality Frameworks

The following section presents and discusses existing approaches for evaluating the quality of models, systems, and patterns, which have been used as input in the development of the ODP Quality Model discussed in Chapter 4.

2.5.1 MAPPER

The MAPPER validation framework [139, 32] was developed within the MAPPER project, the overall goal of which was to develop model-based approaches to improving product and process engineering. 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 many 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 4.1.1), the entirety of the validation framework or its metamodel has not been adopted outright, nor is it exhaustively covered here.

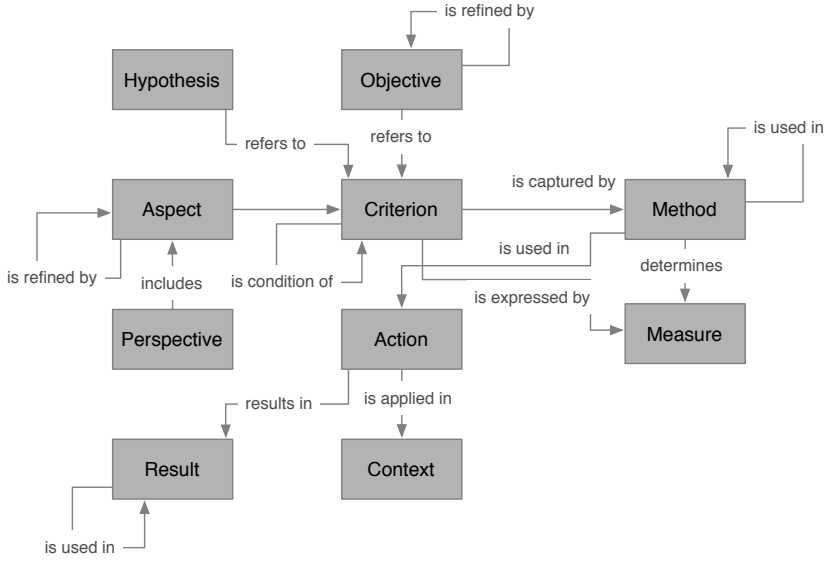


Figure 2.10: MAPPER validation framework metamodel (source: [32]).

The MAPPER validation framework [139, 32] 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 2.10). In evaluating an 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 distinction between 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.

2.5.2 Conceptual Model Quality

Ontologies are essentially models of the world, or at least, a domain of discourse. ODPs can therefore be considered as small reusable models of a

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* [161] by Christer Thörn deals extensively with how to study and ascertain the quality of conceptual 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 ODPs 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 [38, 156]. 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 interpret, 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 for many years, by users of varying technical background and skills [46, 146, 12]. Ontology Design Patterns are intended to support users of varying technical background and skills in conceptual modelling, a not entirely dissimilar task.

Thörn develops and presents a quality model containing six quality factors believed to be most relevant for feature models [161, p. 152]:

- *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.

These quality factors were selected from an initially larger set of factors and attributes based on 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 that

are 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 [161] 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 that are intended to guide feature model developers in constructing models adhering to the defined prioritisation of quality factors.

2.5.3 Entity Relationship Model Quality

Entity Relationship (often abbreviated ER) models are a type of conceptual model 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 considered 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 [61] 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 to evaluating understandability and modifiability is clearly also applicable outside of this domain.

Moody and Shanks [120] 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 views expressed by Lindland et al. in [110]: 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 [120] 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 the opinions of several different stakeholders, with different perspectives. They suggest that these perspectives should be captured via rating by three categories of users, depending on their 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 also likely to be suitable for reuse in studying Ontology Design Pattern quality.

2.5.4 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 [169, 117, 94]. 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 [94] 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 software product or computer system that

includes a certain piece of 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 25000–25099). These definitions include, among other things [94]:

- *Software Quality Characteristic*: Category of software quality attributes that have some bearing 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, optionally including 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. As the name implies, measures are measurable and quantifiable. Quality characteristics are more abstract and general.

The Quality in Use model is displayed in Table 2.1. 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*” [94, p. 8]. These quality characteristics are affected by and measure factors relating to the individual software product, the information system of which the product is a part, the usage 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 [94] is displayed in Table 2.2. 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 at the level of granularity where 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

Table 2.1: ISO 25010 quality in use model (adapted from [94])

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

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 [94] 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 [94] is a complete standard for IT software systems, that is widely used in practice. Since, as mentioned, ontologies and ODPs are used in such systems, it 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 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.

2.5.5 Pattern Quality

Ontology Design Patterns are a sort of design pattern, 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. It is reasonable to assume that quality indicators associated with such design patterns could also be applicable to design patterns in the ontology domain.

The usefulness of object oriented design patterns in software engineering has been shown many times, see for instance [13, 116, 34]. These patterns encode common best practices for how to solve different 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.

Table 2.2: ISO 25010 product quality model (adapted from [94])

Quality characteristic	Subcharacteristic
Functional suitability	Functional completeness Functional correctness Functional appropriateness
Performance efficiency	Time behaviour Resource utilisation Capacity
Compatibility	Co-existence Interoperatbility
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

However, there are fewer 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. [130] report on two experiments in which the characteristics of design pattern code implementations (specifically, 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 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 on that software code. Semantic Web ontologies and ODPs also encode comments in the pattern implementation (in the form of `rdfs:comment` annotations), possibly making this finding on the importance of documenting patterns transferrable to an ODP context.

2.6 Ontology Quality 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 discussed in Chapter 4. In addition to the ontology evaluation work introduced below, the interested reader is referred to [63], in which Gómez-Peréz et al. summarise and discuss several types of common taxonomical errors and anti-patterns in ontologies.

2.6.1 O^2 and oQual

A thorough study on the evaluation of ontologies is performed by Gangemi et al. in [56] and [57]. 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, that is, information objects with intended conceptualisations to be used in 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, the intended meaning of the ontologies, their 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 for a whole, ontology engineering project.

Based on the concepts in O^2 , three dimensions of quality or evaluation, each associated with its own group of measures, are presented and discussed—*structure*, *functionality*, and *usability*. The first group of measures treat the ontology as a directed graph (which is consistent with the RDF data model) and concern the structures present in this graph. This includes measures like subsumption hierarchy depth, breadth, fan-out, cycle ratios, density, etc. The second group 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 group of measures, finally, concern the communication aspects of the ontology, such as how it is documented, annotated, and understood by users. This includes measures related to recognition, efficiency, and interfacing. While the structural measures included in [57] 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 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.

2.6.2 ONTOMETRIC

ONTOMETRIC, introduced in [113], is an approach for formalising ontology suitability for different tasks, heavily influenced by the Analytic Hierarchy Process (AHP) [138], 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 ONTOMET-

RIC, the highest score would be associated with the most suitable ontology for reuse).

To support this method, ONTOMETRIC [113] 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 employing 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 [113] seems very suitable. However, the method does have some drawbacks: it 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. Furthermore, 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. Finally, the process is rather complicated—while this might not be a drawback in a case where one is choosing one out of a set of (potentially rather large) candidate ontologies for reuse, it adds significant overhead in the ODP-based scenario, where one must more frequently choose between a set of (smaller) ODPs to reuse.

2.6.3 OntoClean

One of the most well-known methodologies for evaluating the conceptual consistency of ontologies is OntoClean by Guarino and Welty [170, 68, 69]. OntoClean uses a logical framework including very general ground notions and definitions from philosophy that are 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 denoted in OntoClean parlance as *metaproperties*. Below these metaproperties are explained and exemplified (examples are taken from [68] and [69]):

- *Rigidity*: Rigidity is related to *essence*. A class is considered *essential* for some individual entity if the entity must logically be a member of said class at all times. Essence can be exemplified by the class *HardThing*, which is essential for a hammer (every hammer must always 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 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 non-essential for every student—all students can cease being students and still exist. Finally, *Semi-rigid* classes 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. 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 [69]:

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 several projects [149, 59, 43, 86]. Plugins supporting the use of OntoClean in different ontology engineering environments have also been developed, including WebODE [48] and Protégé 2000⁹.

2.6.4 Terminological Cycle Effects

In [105] 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 on the computability of an ontology. Using different state-of-the-art reasoning engines they find that, to a large degree, reasoner performance over large ontologies 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, that is, 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. [105] note that the computational performance characteristics of a reasoner-ontology pair is highly dependent on the description logic language used.

2.6.5 ODP Documentation Template Effects

In their master thesis Lodhi and Ahmed [111] 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 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 considering this. The patterns and documentation templates studied are all taken from the NeOn Ontology Design Patterns portal¹⁰. Lodhi and Ahmed [111] perform two online surveys, targeting novice and expert users respectively. Both surveys indicate that there are certain fields of ODP documentation that are considered by users 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

⁹<http://protege.stanford.edu/ontologies/ontoClean/ontoCleanOntology.html>

¹⁰<http://ontologydesignpatterns.org>

are not considered to be as important in these regards (though still relevant) include textual descriptions of ODP elements and ODP domain classification.

Chapter 3

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 PhD project, several methods have been employed. In the following chapter, Section 3.1 introduces well-established methods that are frequently used within the computing disciplines. Section 3.2 then details how those methods have been selected and employed to gather and develop the knowledge required to answer the project's research questions.

3.1 Applicable Methods in the Computing Disciplines

Before entering into a discussion on method, it is relevant to frame the work performed in this dissertation 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 several related academic disciplines, each with their own academic traditions including perspectives on epistemology and philosophy of science, preferences regarding 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* [62]¹.

¹While 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 to be distinct.

Of these three, the work performed in this thesis aligns most closely with Software Engineering: the work 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 [11], research in Software Engineering that has to do with products, processes, or people is best performed using an inductive as opposed to a 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 the testing, the theory is disproven. In this paradigm, the scholar applies general theory to a specific case [11, 147]. 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 m/s^2), which can be evaluated via experiment, possibly disproving the original theory.

On the other hand, in the inductive paradigm, individual empirical observations about some phenomenon are used as grounding for the postulation of general laws and generalisations regarding said phenomenon [147]. 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 various means, tries to capture and understand [11].

In performing research within either of these two paradigms, the scholar may employ quantitative or qualitative 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 piece of 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 as an impartial observer [96]. 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 [96].

Such quantitative methods can be contrasted with 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 [121, 39]. 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 shallower quantitative studies [136]. Also, this problem can be partially overcome if the researcher takes care to support any assertions made with 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’s assertion concerns. This is known as triangulation, and it is common in case study research, which: “[...] *relies on multiple sources of evidence, with data needing to converge in a triangulating fashion [...]*” [173, p. 18].

When evaluating research, we typically expect research activities to display different qualities or attributes depending on how they are classified per these dichotomies. Exactly which qualities those are or should be is subject to ongoing debate within the academic community, but certain terms or ideas are more or less established - these include the attributes *generalisability*, *reliability*, and *validity*.

- *Generalisability*, as the name implies, concerns the degree to which the findings of the research can be generalised to a wider context or applied to related contexts or fields. This quality is sometimes referred to as *external validity* or *transferability*. Questions one might ask to evaluate the degree of generalisability of some research work include *Is the method based on existing theories?* and *Are there limitations to the evidence collected that might suggest limited generalisability of the findings?*
- *Reliability* concerns how stable the findings or results are, for instance over time, over functionally equivalent datasets, or over different researchers or research groups. Somewhat synonymous terms include *dependability* and *consistency*. Control questions for reliability include *Are methods, experiments, and procedures described in sufficient detail?* and *Is it clear which data is used for testing and experiments?*
- *Validity* (sometimes *internal validity*) is perhaps the most important quality of research. It concerns how truthful the presented findings are. In quantitative approaches this is typically indicated by the strength of causal relationships between independent and dependent variables; in qualitative approaches validity relates to the credibility of findings,

which is strengthened by, for instance, multiple subjects or participants expressing similar sentiments, or by deeper embedding of the researcher in the context or case under study. Control questions for validity might include *Does the method actually solve the problem? Completely or only partly?* or *Does the evidence support the claims for the research results sufficiently?*

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 methods encode large and overarching theories, making rather general recommendations that sometimes include perspectives on philosophical ontology (not to be confused with Semantic Web ontologies) and epistemology. Examples of these include Hermeneutics, Design Science Research, Mixed Methods Research, etc. Other methods have a narrower focus, recommending how to approach a certain problem in terms of selecting and employing data gathering methods 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, ethnographies, etc. Finally, at the lowest and most detailed level, we have concrete 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 dimensions discussed above. 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 [62] and [124]).

3.1.1 Design Science—A Pragmatic Approach

The aforementioned research paradigms and dichotomies (deductive versus inductive, quantitative versus qualitative) are largely grounded in and derived from the deeply philosophical and rather thorny dichotomy of positivism versus interpretivism. The positivist philosophical perspective postulates that there are an objective reality and objective truths that a researcher may study and learn about by way of observations and logical reasoning. Positivists argue that the observer should distance themselves from the observed in order not to affect the reality under observation, that universal laws govern the world, and that ideally findings should generalise to all contexts [127, 96]. Interpretivists suggest that the positivist paradigm is unsuitable in studies involving humans as social actors. Instead, they suggest that given that each person constructs their own understanding of reality, it is practically impossible to establish an objective truth. Consequently, context-free generalisations are neither useful nor possible to reach. Further, they sug-

gest that the researcher cannot disentangle themselves from the subject of study [127, 96].

Obviously these two schools of thought are largely incompatible. The finer points of both perspectives and their impacts on method have been debated endlessly by senior scholars; indeed, in some fields it would not be possible to attain a professorship without first penning a substantial article on the subject. At present, there's little reason to believe that the debate will be settled shortly. As the computing disciplines typically involve both natural sciences and maths problems on the one hand, and the usage of computer systems by human actors on the other hand, selecting which philosophical foundation to adhere to is difficult for researchers in these fields. There is however a third option open to such researchers, namely *pragmatism*.

The pragmatist perspective ignores much of the fundamental ontological and epistemological debate, and instead emphasises the impact of developed research theory and findings. As put by Burke Johnson & Onwuegbuzie [96]:

The pragmatic rule or maxim or method states that the current meaning or instrumental or provisional truth value [...] of an expression [...] is to be determined by the experiences or practical consequences of belief in or use of the expression in the world.

This clear focus on what *works*, rather than just *how* or just *why* something works, has many advantages. To begin with, a pragmatic researcher is not married to either an inductive or deductive paradigm, or to quantitative or qualitative method choices. Instead, a pragmatic researcher can select among established paradigms and methods and choose those that are relevant to achieving a specific research outcome, that is, to studying some situation or concept, developing some theory or deliverable, etc. Many times, pragmatists combine different research paradigms and methods to arrive at stronger and more well-founded conclusions, as advocated by *Mixed Method Research* proponents [96, 97]. Secondly, the pragmatic perspective is arguably analogous to and derived from how most people naturally operate and prefer to operate: that is, we try out different solutions to see which ones work. While this problem-solving mind-set may be too simplistic to be optimal in very large research endeavours such as the development of a Grand Unified Theory of physics, the large majority of research projects, particularly in the computing disciplines, are much smaller and more applied. Thirdly and relatedly, theories and results developed in pragmatic research naturally tend to be applied and practical, that is, they serve to inform and develop practice outside of academia.

The *Design Science* methodology, formalised and described by Hevner et al. [89], aligns well to such a pragmatist perspective². This methodology, which is common in Information Systems research, traces its roots to

²Though [89] does not propose such a linkage, later work by the same author presented in [88] positions Design Science as drawing from pragmatist philosophy.

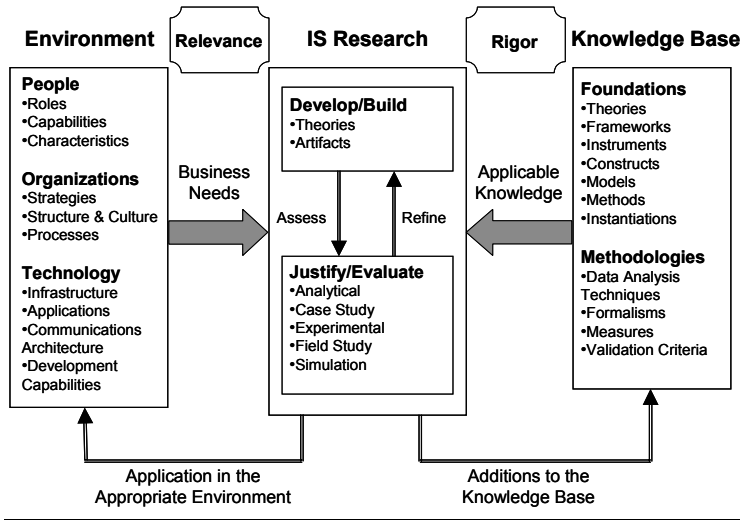


Figure 3.1: Information Systems research framework by Hevner et al. (source: [89]).

engineering science and the sciences of the artificial [144]. At its core, it concerns how to develop artefacts for human usage in a manner consistent with scientific methods and requirements, with an emphasis on the practical relevance of the developed solution and rigour in its development. Hevner et al. frame Design Science (and indeed, Information Systems research as a whole) as drawing relevance from the practical/business environment in which research is performed (e.g., the technology, organisations, and people that the research involves), and drawing rigour from the established academic knowledge base (theories, frameworks, models, methods, techniques, etc), while contributing back to both, in the form of applied results to practice, and theory additions to knowledge, as shown in Figure 3.1.

In order to concretise and communicate Design Science and how they propose Design Science should be performed, Hevner et al. present seven guidelines of good Design Science research, copied here in full [89]:

- *Guideline 1—Design as an Artifact*: Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
- *Guideline 2—Problem Relevance*: The objective of design-science research is to develop technology-based solutions to important and relevant business problems.

- *Guideline 3—Design Evaluation:* The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
- *Guideline 4—Research Contributions:* Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
- *Guideline 5—Research Rigor:* Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
- *Guideline 6—Design as a Search Process:* The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
- *Guideline 7—Communication of Research:* Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

An extensive presentation and exemplification of each guideline is available in [89]. We will only briefly emphasise two things. Firstly, Guideline 3 requires developed artefacts to be evaluated. Design Science does not mandate the use of any particular evaluation method, but Hevner et al. summarise some commonly used and appropriate types of artefact evaluation methods: observational (case studies, field studies), analytical (static analysis, architecture analysis, optimisation, dynamic analysis), experimental (controlled experiments, simulations), testing (black box testing, white box testing), and descriptive evaluation (argumentative or scenario-based) [89]. Secondly, Guideline 6 discusses design as search process. This is pedagogically illustrated in the Develop/Build-Justify/Evaluate cycle at the centre of Figure 3.1—the Design Science researcher will develop an artefact, evaluate that artefact against alternatives, requirements, or constraints, and either improve upon it, or attempt to construct another better solution, in an iterative manner. This approach is compatible with and essentially a reformulation of the Generate-Test cycle described by Simon [144], from whose work the Design Science methodology traces part of its lineage.

3.1.2 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, either to 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 currently 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 [143].

An important difference between a systematic literature review and other types of unstructured studies 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 means in practice is that in each step of the literature review, search parameters, analysis methods, and extracted metadata 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 [102] suggests a set 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, to be palatable to scholars on both sides of the qualitative/quantitative fence, it needs to be extensively documented and structured to be considered both transparent and trustworthy.

An issue to note is that as 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 under 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 [95], and Schneider et al.'s review of solutions to globalisation challenges in Software Engineering [141].

3.1.3 Interviews

Interview methods are often employed to gather data from stakeholders or case participants familiar with the phenomenon being studied, without necessarily observing the phenomenon itself. In a software engineering context, such interviews can be performed, for instance, to gain an understanding of how an artefact is appreciated by users, or to elicit requirements for a system or service. Sometimes interviews are used to follow up on an observational study, in which the interview subject is asked to explain the observed behaviour, to establish a better understanding of the observations made [142].

Interviews can be *structured*, *semi-structured*, or *unstructured*. In the structured interview case, the interviewer has designed questions beforehand, and asks 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 very clear and specifically defined information requirements. On the other hand, in unstructured interviews, 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. Finally, Semi-structured interviews 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 [142, 145].

When working with data gathered via interviews, the interview recordings are first transcribed into text, and those text transcripts are then 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 fragments (i.e., a unit of text of suitable size for subsequent analysis, typically a sentence, a statement of one or more sentences, or an answer to a particular question) 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 theme or aspect of the phenomenon of study. To reduce the impact of any individual researcher bias, these processes are often performed in parallel by multiple researchers [31]. Consequently, this transcription and analysis process can be very time-consuming, and for this 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 [142].

The reliability of qualitatively analysed work such as interviews depends largely on the transparency by which the research process has been performed, and the availability of the source materials. Ideally, another researcher should be able to study the source material, analysing it to develop similar coding and arrive at similar conclusions. In the real world, this is unlikely to occur. For one thing, the researcher's background and bias are impossible to root out altogether. For another, the source materials may be subject to restrictions in terms of distribution, making such validation impossible in practice. Consequently when evaluating reliability of qualitative research emphasis tends to be placed on evaluation of how the process was designed and documented.

3.1.4 Surveys

Surveys are a common method for gathering data from a larger set of respondents, where interviews do not scale well, e.g., for reasons of constraints in time, the geographic distance to respondents, or other reasons. In a survey, each such respondent is queried using some appropriate data collection techniques or tooling. Typically, questionnaires are used, but this is not a requirement of the survey method as such: surveys can also be performed using structured interviews, database record analyses, etc. However, it is important that each respondent be subject to identical data collection methods, so unstructured or semi-structured interviews are not appropriate to employ when gathering data for survey purposes. In this dissertation, unless stated otherwise, the term “survey” is understood to indicate questionnaire-based surveys. [127]

Surveys typically aim to extract the relations between two specific variables under study—for instance, in the context of this PhD project, survey methods could be used to link features of ODPs to perceived usability. Surveys are also sometimes used (possibly less commonly in the computing disciplines) to study features or characteristics of the respondent population themselves—for instance, one might use a survey method to query users in certain age brackets regarding their opinions about Internet banking.

For survey results to be generalisable to a larger population than the respondent set, that respondent set needs to be a representative sample of the larger population. Consequently, either a rather large respondent set is needed or some other method needs be employed to ensure that the survey population is not biased. If neither of these two conditions can be met, the survey results cannot be guaranteed to be generalisable [127]. That does not imply that the results of such surveys are useless; they can still be valuable input in development processes or as signals for further research. Also, the generalisability of quantitative style survey results is not a binary proposition, but rather can be expressed in terms of confidence levels and confidence intervals, analysed down to the individual question level. Consequently, in the case that a significant enough majority of respondents agree

on the response to some survey question, such a finding may be indicative of a trend that holds in the general population also, despite the respondent set being too small to speak of generalisability of the survey results as a whole.

Most readers are likely already familiar with questionnaires and have filled out at least a few at some point in their life. Some have probably also constructed questionnaires. It might come as a surprise to the former group (and less so to the latter) that questionnaire surveys are notoriously tricky to get right. There are two main obstacles to constructing such a survey: firstly, typically the researcher is not personally available to the respondent and able to answer questions on questionnaire structure or the meaning of terms or questions. Consequently, the survey needs to be constructed just right from the very outset. Secondly, regardless of distribution method (paper questionnaires sent by snail mail, email distribution, web surveys, etc.), the response rate for academic surveys is typically low. While response rates can be driven up by way of incentive systems, it is important that these incentives do not skew the results (i.e., offering monetary rewards on a public-facing Internet survey might not necessarily result in the most trustworthy of results).

Given the above challenges, it is particularly important that the questions asked in the survey are indeed the right ones needed to study the research question at hand. Some additional important characteristics of a good questionnaire include that it [127]:

- Is not too long to answer in a reasonable timeframe.
- Is not biased in question selection, nor written with leading or biased questions.
- Has questions that are clear and unambiguous.
- Has a logical progression of questions or themes.
- In the case of branching paths (i.e., if earlier responses affect the selection of later questions), branching should be intuitive. Branching should not be over-used.

The questions in a questionnaire survey can be either closed or open—in the former case, the researcher provides a series of answer alternatives for the respondent to select from, whereas in the latter case, the respondent is asked to fill in a free text response. Open-ended questions of the latter variety may be quite useful in that they give the respondents a chance to comment on something that the researcher did not think of themselves. This adds a qualitative flavour to the questionnaire, allowing the researcher to discover and explore entirely new ideas that were not part of the initial research question or hypothesis. However, as no follow-up questions can be posed, and as deeper context or narrative is lacking, open-ended questionnaire responses are a very shallow form of qualitative material in

comparison to interview transcripts or observational studies. Also, questionnaire respondents typically do not like to answer such open-ended questions; they generally require more thought and more time to respond to. For this reason, there are typically only a few open-ended questions in most surveys [127]. Some findings indicate that prefacing the question with a more extensive (though possibly redundant or repetitive) introduction can lead to respondents providing more information [52].

Closed questions, on the other hand, are more common. There are several categories of such questions. In single-choice questions the respondent selects only one option out of a provided set of alternative answers. For such questions, it is important that the answer set either be exhaustive (i.e., the alternatives are *dichotomous*, such as “Yes” and “No”), or that a catch-all response such as “Other” be included, to ensure that the respondents find at least one suitable alternative to select [127]. It should be noted that single-choice questions cannot be strictly enforced in a paper-style questionnaire, so in such surveys they may be a cause of error. Multiple-choice questions allow for multiple choices and thus avoid this risk. However, they too may make use of an “Other” response category (possibly combined with a free text entry field), to cater to respondents who do not find any of the provided alternatives suitable. Note however that the use of free text responses can complicate analysis significantly. Ranking questions gauge respondent preference or prioritisation among provided alternatives, by assigning an ordering over all of the alternatives. Scale questions, finally, ask respondents to select a value from a given response scale for each question. There are a variety of methods for constructing scale questions and response scales; perhaps the most commonly used one is the Likert scale [109], per which respondents are asked to how large a degree they agree with a given statement, such as “I find the system easy to use”, typically employing a scale of five or seven steps, ranging from “Strongly agree” to “Strongly disagree”.

The validity of survey results depends primarily on whether the participants have responded to the survey questions in accordance with their actual opinions, experiences, or beliefs. Proving this conclusively is of course impossible, but provided that respondents have no incentives to lie or exaggerate, it is reasonable to assume that most respondents will be honest in their responses. However, it is important to note that such incentives are not limited by anonymising responses; it may well be the case that research subjects who know the researcher (such as university students taking part in their professor’s survey study) would want to help the researcher obtain a successful result. Generalisability of survey results, much like other quantitative results, require that the respondents be representative of the greater population to which the results are intended to be generalised, which can be deduced either by way of sampling strategy, or by way of having a large enough number of respondents.

3.1.5 Researcher Logs or Participant Diaries

The use of participant diaries or researcher’s logs (for the sake of simplicity, “diaries” will be used in the following to indicate both approaches) for data collection is relatively unusual in the computing disciplines—possibly because the format eschews the traditional divide between the observer and the subject of observation. By necessity, the researcher or the participant who writes such a diary applies a certain structuring and filtering or prioritisation of their observations (possibly subconsciously) when writing a diary entry. This encodes subjectivity into the data on a rather fundamental level, which one must be aware of in subsequent analysis. Data gathered in this manner might, for instance, not be suitable as the foundation for a new theory. Nevertheless, provided one takes care and is mindful of their limitations, diaries can be very useful in developing insight into thought processes, ideas, methods, etc. that might otherwise remain unexplored or forgotten. They allow the gathering of data in immediate proximity to the event or process that is being studied, even if the researcher is not present themselves. They can also be useful for gathering data pertaining to the evaluation of new artefacts, or for triangulation of other data sources. [127]

Working with participant diaries poses some additional challenges that researcher logs do not have, namely that the participants must be comfortable with the data collection process and structure, and that any ethical issues with regard to how the data is to be treated must be thoroughly settled beforehand. While it is always important to treat participant data in an ethical manner, it becomes particularly important when that data is gathered via a method as potentially intimate and personal as a diary. Essentially, the participants need to trust the researcher to a much larger extent than if they had simply submitted a questionnaire or responded in an interview.

In addition to the ethical and trust issues, participants also need to be comfortable with the process of writing diary entries. Preferences regarding this process vary: some people prefer to be provided with structured diary sheets that they fill out, containing a series of questions that they answer (not entirely unlike answering a series of open-question surveys). Others prefer writing their own diary entries in an entirely free-form way. The latter might be more complex to subsequently analyse, but can on the other hand provide richer and more unexpected data. Either way, it is important that participants be given instructions and examples of how to use the diary. Such instructions should cover the frequency with which entries should be penned (upon the occurrence of some event, once daily, once weekly, etc.), which are the most important aspects to note down or reflect upon in each entry (checklists can be useful for this), what terms and definitions are used in the project or research community (to aid participants in finding a shared vocabulary), etc. [127]

For researcher’s logs, the issues of instructions, ethics, and the observing party being comfortable with the format of the observation protocol, are of

less importance, as the researcher is the observer. However, to the extent that entries in the log discuss or identify other people involved in the research process, such issues of course remain.

Schatzman and Strauss identify three types of researcher log entries [140]:

- *Observational notes*: These log entries contain statements about events the researcher observed through watching and listening to some event or process. Observational notes are (consciously) interpreted as little as possible by the researcher. In penning an observational note, the researcher might be guided by the classic questions of journalism: Who, What, When, Where and How.
- *Theoretical notes*: Attempts to derive meaning from observational notes, that is, to analyse, deconstruct, and create theory from observation. This may in many cases require the researcher being self-conscious about their own perspectives and biases.
- *Methodological notes*: Such notes cover the planning or execution of some method or task within a research project. Examples might be a note to oneself, a reminder, an instruction, or something of that nature, that might not make it into some final publication, but is useful to keep for posterity all the same. Schatzman and Strauss remark that, on a meta-level, a methodological note could be considered an observational note of the researcher's own research process [140].

The format of a researcher's log can vary from the very comprehensive (i.e., one giant document containing a great number of notes taken throughout the course of some large research project) to a more ad-hoc style (i.e., a scrapbook of different notes taken at different times and in different contexts). There is no one prescribed way that works for all researchers [127]. While the former makes for significantly simpler structuring and analysis, the latter of course requires less overhead to initially set up and get started with.

3.1.6 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*) of 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 that are not 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 [11] 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 a 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 approaches experiments are used not 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 a 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 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, [11] does not define many constraints on what may be considered an experiment or not—for instance, by this understanding there is no requirement that data resulting from Software Engineering experiments need necessarily 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 dissertation.

3.2 Research Process

As discussed already in Chapter 1, the knowledge generated within the context of this PhD project has been developed primarily in an inductive manner³. That is to say, based on several different data gathering and research activities, the author has developed empirical material, which has subsequently been treated and analysed, in order to try to develop an understanding of the phenomena under study, with the goal of answering the proposed research questions. Per this paradigm, it is difficult to exhaustively answer the research questions with precision—instead, the researcher aims to develop new theories that cover the study objects and which hopefully generalise to other cases also.

³Though some individual research activities have in fact been organised as quantitative/deductive experiments, the overall thrust of the work has been inductive.

Furthermore, the work has been planned and performed in a pragmatic manner, adhering to the *Design Science* guidelines introduced in Section 3.1.1 [89]:

- *Guideline 1—Design as an Artefact:* All three research questions have been explored via the development of and improvements to one or more artefact(s), namely an ODP Quality Model (Research Question 1), the XDP ODP usage tooling (Research Question 2), and ODP usage methodology (Research Question 3).
- *Guideline 2—Problem Relevance:* As shown in Chapter 1, ODPs show promise in simplifying ontology engineering for non-academic uses. This PhD project aims to fill known knowledge gaps regarding appropriate ODP qualities, tools, and techniques, with an emphasis on usage by inexperienced ontologists (i.e., people who are not ontology researchers). The end goal of this work is thus to enable and support the uptake of ontology-based technology (which has been shown to be a viable solution to a great many applied information management problems) by practitioners outside of the Semantic Web academic context.
- *Guideline 3—Design Evaluation:* Two of the three main artefacts developed within this PhD project have been evaluated; the contents of the ODP Quality Model have been evaluated iteratively through both experimental and practically applied approaches, and the XDP ODP usage tooling has been evaluated through separate evaluation of both its components and a usability evaluation of the tooling as a whole. However, due to lack of time and lack of sufficient study cases, the contributions to the XD methodology have not been evaluated.
- *Guideline 4—Research Contributions:* The research contributions of this work include a conceptual understanding of quality as it relates to ODPs, a catalogue of quality characteristics, quality indicators, and recommendations compliant with this conceptual understanding, partial evaluations of the XD methodology in real world Ontology Engineering projects, and improved algorithms and heuristics for finding, instantiating and composing ODPs.
- *Guideline 5—Research Rigour:* The rigour of the work is guaranteed through adherence to established research practices and methods, as discussed and evaluated in Section 3.3.
- *Guideline 6—Design as a Search Process:* The ODP Quality Model was developed in an iterative manner in three loops of generation and evaluation. Likewise, several of the components of the XDP ODP usage tooling were developed in a similar process of trial-and-error, most notably the template-based ODP instantiation heuristics (Section 5.3) and the CompositeSearch component (Section 5.1).

- *Guideline 7—Communication of Research:* This is the guideline that the presented work follows least: while the author has taken care to communicate the work in as applied and practitioner-friendly a manner as possible (e.g., the checklists, decision trees, etc., in Section 6.4, or the poster publication presenting the XDP tooling [79]), the main audience of this PhD dissertation remains an academic/technical one, rather than a management-oriented one. For that academic/technical audience, the work has been communicated in several publications, see Section 1.4.

In keeping with the inductive/pragmatic theme of the work, neither the project’s research questions nor method choices, were set in stone at project initiation; rather, both the research questions and the methods employed to explore those questions were selected and developed in the context of the author’s involvement with five ontology engineering projects (detailed in Section 3.2.4). Figure 3.2 illustrates this development of the research questions⁴. The first research question, which initiated the whole PhD project, concerned the quality of ODPs (*Which ODP features or qualities are important in supporting pattern understanding and use?*). This question was developed from the findings of two systematic literature surveys that the author performed, covering the key Semantic Web conferences and journals [84, 72]. To develop an understanding of quality in an ODP context, the author constructed and evaluated a quality model, through three Generate/Test-iterations. During the first evaluations, the author observed that the quality of ODPs is not the only barrier hindering their adoption—lack of tooling, particularly tooling targeting novice developers, is also a considerable problem. This generated the second research question (*How can the features and functionality of ODP usage tools be improved to support inexperienced ontologists?*). To develop and evaluate such tooling, the author was invited to participate in four ontology engineering projects. While working on those projects, the author observed yet another issue hindering ODP use, namely, the need for improved methodology support. This in turn lead to the third research question (*How can ODP usage methodology be improved to support inexperienced ontologists?*), which was subsequently explored within three of the same projects.

These three questions were studied via a variety of research and data-gathering activities. Some of these activities were initiated and developed entirely by the author; others were in fact part of the ongoing work within the projects in question, into which the author was given the opportunity to immerse himself to observe people and processes.

The remainder of this section first gives an introductory overview of the data-gathering activities and how they contribute to answering the research questions. This introduction is split into subsections corresponding to each

⁴This figure glosses over some details, which are covered in the more detailed figures 3.3, 3.4, and 3.5.

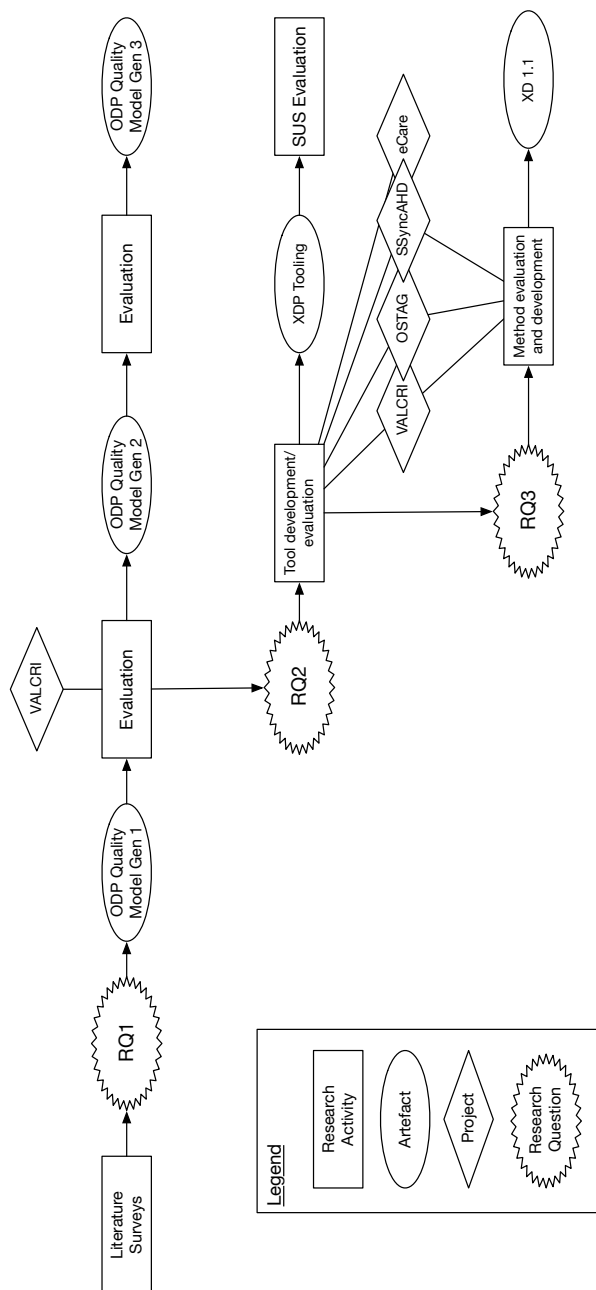


Figure 3.2: Research question evolution and interrelations.

research question under study, each including a further graphical illustration indicating the relations between data gathering activities and results. Most of the descriptions on how the work was performed are included together with the discussion of analysis and results in the subsequent chapters 4 through 6—those chapters are intended to be relatively free-standing. However, since certain data gathering activities have contributed to answering more than one research question, this section also introduces the research projects in which those cross-cutting activities took place and the data gathering and analysis methods employed.

3.2.1 Answering Research Question 1

Figure 3.3 presents an overview of the research process and the included research activities performed to answer Research Question 1: *Which ODP features or qualities are important in supporting pattern understanding and use?* Answering this question necessitated the development of an ODP Quality Model. This model was developed in three iterations, with the latter two development iterations evolving and refining the model developed in the previous ones.

First Iteration

The first generation of the quality model consisted of a quality metamodel and constructs to populate said metamodel. The metamodel is best explained as a conceptual model of how to think of quality-related phenomena in an ODP context, including concepts such as quality characteristics, quality indicators, measurement methods, scales, etc. As previously mentioned, the first iteration quality model also includes an initial set of constructs, that is, concrete quality characteristics, quality indicators, etc. Both the metamodel and the initial constructs were developed from existing theory within research on artefact quality, including results from the MAPPER project [32], the ISO 25010 software quality standard [94], the work on feature model quality presented by Christer Thörn in his PhD dissertation [161], and several other pieces of literature relating to ontology quality, ER quality, etc.

Second Iteration

The first generation of the quality model was then evaluated within three different studies, as indicated in Figure 3.3, the results of which lead to certain refactoring of the model, and the release of a second-generation model⁵.

Firstly, a two-day workshop within the IMSK project (Section 4.2.1) enabled the author to evaluate structural quality indicators affecting ODP

⁵It should be noted that the entire model was not evaluated—rather, only a subset of the model (those quality characteristics and indicators that could be tested within the confines of the listed studies), were evaluated.

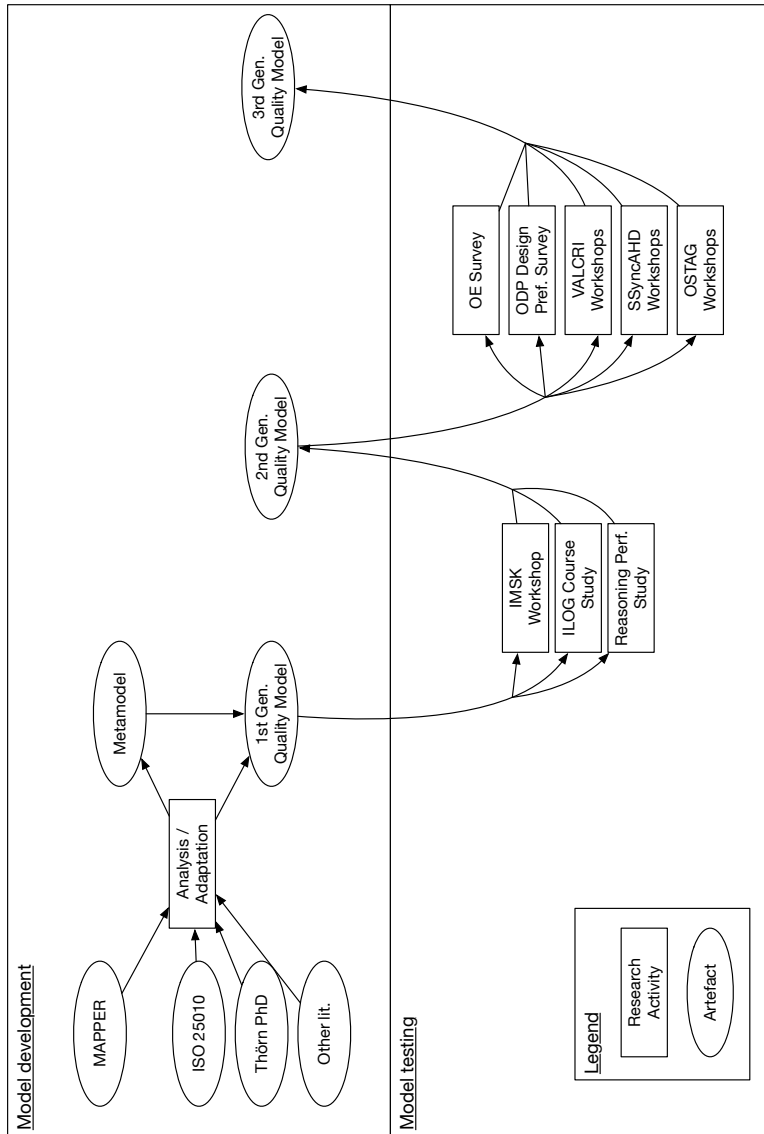


Figure 3.3: Overview of research process for answering RQ 1.

usability, by way of observations of workshop participant behaviour and practice, and through interviews with participants about that behaviour. These data gathering methods were chosen due to their suitability in the given project context, where the participants were interested in receiving guidance on and testing out the use of ontologies and ontology engineering in practical work, but were not as keen on taking the role of research subjects in a more structured experimental setup.

Secondly, a combination of experiment and survey approaches was employed within a master course in Information Logistics at Jönköping University (Section 4.2.2), enabling the evaluation of further structural and documentation quality indicators contributing to usability and learnability. For this study, the goal was to evaluate several hypothesised effects of quality indicators, and thus, an experimental approach of testing these indicators (i.e., independent variables) against effects (i.e., dependent variables) was suitable, and in the context also feasible.

Finally, an evaluation of quality indicators believed to affect the reasoning performance over resulting ontologies was carried out by way of a literature study and via study of how indicators triggering such effects are displayed in ODPs published on the Internet (Section 4.2.3). The literature study method was selected based on the author's intent to expand the coverage of the quality model as efficiently as possible with regard to this rather important aspect of quality, and the prevalence of sufficient amounts of prior work within the area.

The resulting second generation of the quality model is the topic of the author's 2013 Licentiate thesis [76].

Third Iteration

The third generation of the quality model was developed by a process of evaluation and adaptation based on five data gathering activities: two surveys, and observations (in the form of researcher logs, interviews, and audio recordings of participant interaction) at three sets of modelling workshops.

The two surveys (with a total of over 100 respondents) studied user preferences regarding ODP documentation (including aspects such as preferred visualisation format, preferred example syntax, relative importance of different documentation fields, etc.). These surveys are presented in Sections 4.3.1 and 4.3.2. The use of surveys for data-gathering was determined by the author's wish to increase the credibility of findings, by grounding them in a larger dataset (in terms of participants) than the smaller studies from the previous iteration of the work.

The data gathered at the workshops primarily concerns similar aspects of ODP documentation quality, but also touches upon issues such as the need for quality assurance (of both ODP documentation and solution), and the trade-off between generalisability and usability. This work is covered in Section 4.3.3. As with the IMSK project, the choice to observe and participate in hands-on development work in workshop format was based on

the requirement to combine the author's research interests and the project participants' more practically oriented development needs.

3.2.2 Answering Research Question 2

Answering Research Question 2, *How can the features and functionality of ODP usage tools be improved to support inexperienced ontologists?*, involved a number of independent research processes (see Figure 3.4). This is because there are so many orthogonal aspects of ODP support tooling to consider, where the findings of a particular development or evaluation task does not necessarily contribute to any other aspect of the problem, and where marrying different results together into one joint theory is next to impossible. Consequently, the work on answering this research question (detailed in Chapter 5) consisted of several development projects all approaching the research question from different angles. The constituent deliverables, the motivation for their development, and the process by which they were evaluated, are summarised below:

- Section 5.2 describes how, through study of published uses of ODPs, a set of three previously unstudied *strategies for property specialisation* were discovered and formalised. The usage of these strategies in ODPs and ontologies published on the Internet was studied by way of downloading and analysing a large set of such ontologies. The reasoning performance effects of employing two of the strategies were evaluated by way of an experiment.
- Section 5.1 describes *CompositeSearch*, a search engine allowing ontologists to find suitable ODPs based on an input query string. The need for an improved ODP search feature such as this was observed in the IMSK project. The *CompositeSearch* solution was evaluated by way of an experiment comparing its performance to an existing standard system with queries sourced from the QALD (Question Answering over Linked Data) evaluation campaign.
- Section 5.3 describes a set of *template-based ODP instantiation heuristics*, allowing for the instantiation of ODPs into target ontologies by way of cloning of pattern design, rather than pattern concept subsumption. The need for such an alternate way to instantiate ODPs was observed in several projects, including VALCRI, IMSK, and E-care@Home. The template-based approach was evaluated by way of an experiment in the OSTAG project, featuring workshop participants testing and comparing both this approach and the more established specialisation-based approach to ODP instantiation.
- Section 5.4 describes the *XDP* (eXtreme Design for WebProtégé) plugin, supporting the above listed property specialisation strategies,

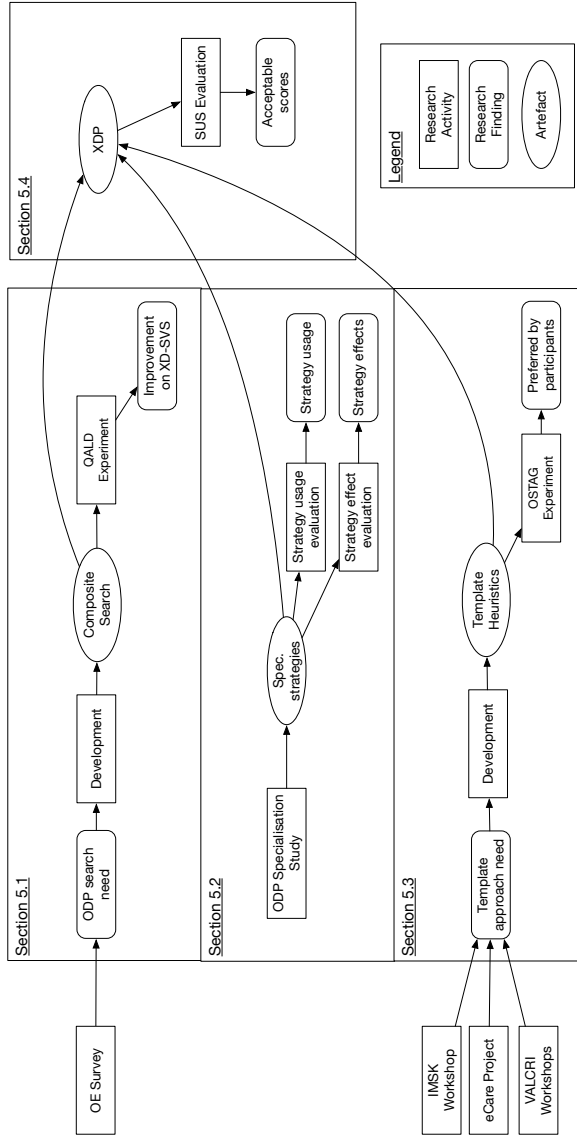


Figure 3.4: Overview of research process for answering RQ 2.

template-based ODP instantiation, *CompositeSearch*, and collaborative XD across geographical distances. XDP was evaluated using the well-known System Usability Scale.

3.2.3 Answering Research Question 3

Research Question 3, *How can ODP usage methodology be improved to support inexperienced ontologists?*, was answered by way of the process illustrated in Figure 3.5.

The existing eXtreme Design (XD) method for Ontology Engineering with ODPs was introduced in three different research projects. Observations of how participants in those projects then worked with or deviated from XD were made by way of participation and observation at development workshops, interviews with project participants, and researchers' logs on any XD method adaptations that were applied. The experiences of trying to use XD in these projects showed several shortcomings in the method. As prior research on ontology engineering methods did not sufficiently cover or resolve these shortcomings, a set of method improvement suggestions were developed.

Traditional eXtreme Design does not specify different roles within the XD project, such as “developer”, “release engineer”, “tester”, etc., with differing responsibilities. Instead, all developers are assumed to be able to perform the different tasks involved in the XD process. Findings from the VALCRI, OSTAG, and SSynAHD projects indicate that this situation is not typical. In these projects, there was a clear differentiation in ontology engineering skill or interest, motivating the need for defined roles, role allocation processes, and task-role mappings.

Further, traditional XD does not give any explicit guidance about reuse of existing ontological resources apart from ODPs. In every one of the projects the author was involved with, the issue of reuse was deemed highly important by management or developers. Consequently, traditional XD needs to be extended with recommendations on concrete methods for ontology reuse.

Finally, traditional XD assumes an ideal ontology engineering case that may not be typical. In the studied projects, challenges were observed regarding the distance (organisational and/or physical) to the customer, the geographic distribution of the development team, and the proficiency of that team. These findings support the development of project adaptation guidance, so that XD can more easily be adapted for use in such cases that are not ideal.

Based on these observations the author developed a set of guidelines, formalised into artefacts such as recommendations, checklists, decision trees, workflows, etc. (see Section 6.4 for a summary and the whole of Chapter 6 for details). These artefacts formalise many of the practical solutions to XD shortcomings that were implemented in the VALCRI, OSTAG, and SSyn-

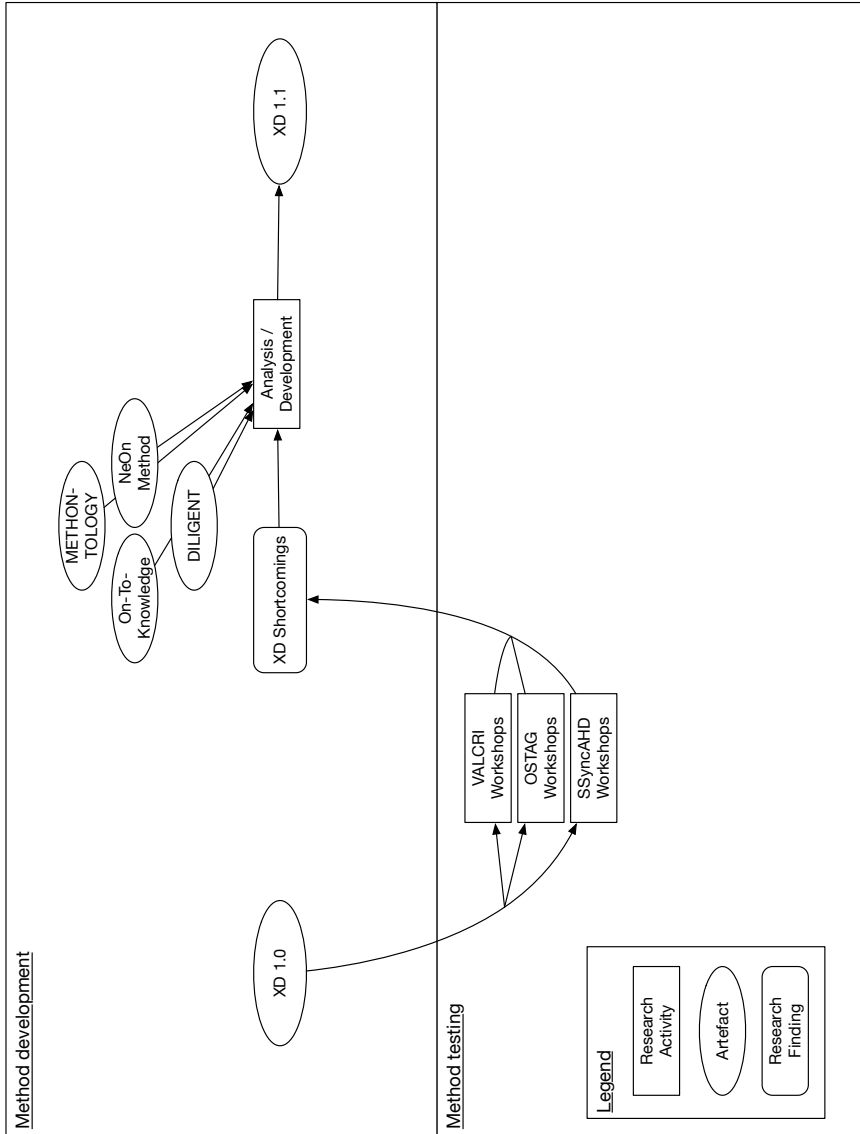


Figure 3.5: Overview of research process for answering RQ 3.

AHD projects, but it should be noted that they have not yet been thoroughly evaluated, neither in these projects nor in other project contexts.

3.2.4 Projects

Over the course of the PhD project, the author has been invited to observe and/or participate in five research projects where ontology engineering work took place. This has enabled the validation and evaluation of theories and artefacts through observational study, interviews, surveys, and experiments, with the projects as study cases. These five projects, and the author's participations within them, are introduced below.

IMSK

The goal of the IMSK project⁶ was to integrate technologies for security and surveillance to provide an easily reconfigurable system capable of providing area security for law enforcement agencies or security services. The author participated in development work within the project together with colleagues from a partner research institute (hereafter denoted RI⁷). The project work packages hosted 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 guidance on response force deployment.

The author was involved in two workshops at RI, both of which had the goal of developing prototype functionality mirroring the functionality in the complex event processing subsystem, but using Semantic Web-based technologies like ontologies and description logic reasoners. The reasons for hosting these workshops from RI's perspective were twofold: first, 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. Second, they believed that using ODPs as preconfigured modules of functionality to plug in and out of the system could support reconfigurability, particularly by less experienced users.

The first workshop, in late 2011, served primarily to introduce the author to the project in question, the staff involved, and the technology stack used, and did not result in much empirical material relevant to this PhD project. The second workshop, held in the middle of 2012, included joint modelling work, as well as a group interview. This workshop was recorded by audio and video. Notes on participant and group behaviour were taken by two independent observers. The total size of the resulting data (after audio transcription) was some 21 600 words, or roughly 85 pages of text (of which 16 are researcher notes, and 59 are audio and video transcriptions).

⁶Integrated Mobile Security Kit, EU FP7 project 218038.

⁷For reasons of integrity and confidentiality, the case description and published data have been partially anonymised.

VALCRI

VALCRI⁸ intends to develop a system providing Visual Analytics capabilities for law enforcement analysts, supporting investigative analysis (i.e., the solving of individual or strings of related crimes) as well as intelligence analysis (i.e., broader crime trends, guiding short- to medium-term force deployment as well as long term strategy). Linköping University is a participant partner in VALCRI, together with 17 other partners throughout Europe.

In VALCRI, ontologies are used for a variety of purposes, including data integration, provenance representation, and as vocabularies supporting event detection in data streams. The majority of the ontology engineering work has taken place in a work package specifically intended to develop an ontology library for use by the various software components that make up the finished VALCRI system. Therefore, a key requirement for those ontologies is that they need to be easily understandable by software developers. Another key requirement is that the developed system and its ontologies should be easy to modify for deployment in different contexts.

The author has participated in and contributed to a multitude of meetings and workshops within this project. The most important ones in terms of this PhD project were three two-day modelling workshops held from the middle of 2015 through the middle of 2016. At each of these workshops the goal was to develop an ontology or an ontology module supporting some required system feature. At one of the workshops, the entire modelling session was recorded and transcribed (yielding some 80 000 words, or 220 pages of text), whereas at the other two, the data gathered was in the form of the author's notes on participant behaviour, or survey responses.

E-care@Home

The E-care@Home project⁹ aims to improve home healthcare for the elderly by using ICT-rich environments to measure, record, and infer facts about people and their environment, which can be used for health recommendations, medication reminders, alerts to healthcare services, etc. This requires, among other things, the development of suitably non-invasive sensor systems and actuators, Internet of Things infrastructure supporting such sensors and actuators, integration with legacy healthcare systems (possibly requiring NLP and/or data mining technologies to extract information from patient records), and semantic integration between both hardware device output/input and health record systems. In this project, ontologies are used as a shared language to achieve such data integration.

The author has been involved in a variety of tasks within this project, providing data to different parts of the PhD project. These tasks include

⁸Visual Analytics for Sense-making in Criminal Intelligence Analysis, EU FP7 project 608142.

⁹<http://ecareathome.se>

requirements development and management work (which provided findings regarding the suitability of developed requirements management tooling), leading an ontology engineering tutorial and workshop (at which surveys on ODP features and tooling were answered by participants), and supporting another ontology developer in a quality assurance role (subsequent interviews with the developer giving insights into method issues).

The project runs from 2015 through 2020, but the work contributing to this dissertation primarily took place during the spring of 2016.

OSTAG

The goal of the OSTAG project (Ontology-based Software Test Case Generation) is to improve automation of software test-case generation and test data. The project attempts to achieve this by developing formal models (ontologies) of the system requirements specification of the system for which tests are to be developed, as well as the general domain of that system, and then to semi-automatically generate tests using novel programming techniques (inference rules, genetic programming, and ontology reasoning).

In this project, the author has helped project management establish methods for ontology engineering work, and tutored project participants in applying these methods and suitable support tooling for them. In so doing, the author has had the opportunity to observe participants performing ontology engineering during development workshops, which has provided input into the method development covered in Chapter 6. Additionally, an experiment studying the advantages of template-based ODP instantiation (discussed in Chapter 5) has also been performed within the project context. Finally, interviews with two ontology developers in the project were also performed, to confirm the validity of the author's observations earlier in the project. The ontology development workshops took place in the spring of 2015, while the experiment and interviews were performed in the spring of 2016.

SSyncAHD

The SSyncAHD project, lead by Sweden's National Veterinary Institute (Statens Veterinärmedicinska Anstalt, hereafter SVA), concerns the development of ontologies supporting animal health data and other related types of data in semantically interoperable ways, in turn supporting data integration for syndromic surveillance systems. Syndromic surveillance systems track early syndrome indicators associated with animal health crises (outbreaks of contagious diseases, environmental toxins, etc.), enabling oversight authorities such as the SVA to respond to these crises immediately, rather than after time-consuming formal lab test diagnoses have been performed. The input data to such systems (i.e., the syndrome signals or markers) varies greatly in structure, degree of formality, language, and assumptions. Additionally, not only does the input vary depending on input source or type,

but it also varies internationally, due to differences in legislative frameworks or agricultural practices and history—none of which are constant. Consequently, the goal of SSyncAHD is to develop an ontology standard integrating these different perspectives, that can be maintained and updated by the syndromic surveillance community of researchers and practitioners to supporting changing conditions.

The author was asked to tutor project participants on suitable ontology engineering practices and tools, and to this end, chaired two workshops at SVA, in the spring of 2015 and the spring of 2016 respectively. Both workshops were recorded, providing a rich empirical material (after transcription around 73 000 words, or 190 pages) concerning features and drawbacks of ontology engineering methods and tooling. At the latter workshop, participants also contributed to the PhD project by performing several exercises and participating in surveys on the tooling developed by the author.

3.2.5 Research Logs

In addition to recording participant interactions and discussions within modelling workshops in the involved projects, the author also kept his own logs of important observed behaviours and reflections on those behaviours, as well as broader reflections on the projects, and success or failure criteria of the projects. Per the definitions given in Section 3.1.5, these notes can be classified as observational notes and theoretical notes. Some methodological notes were also taken, but these were primarily used as mental “sticky notes” for the author, and have not been analysed further. In total these logs comprise around 11 000 words, or around 26 pages of text, covering all the projects discussed above.

Given the limitations inherent to researcher’s logs, as discussed in Section 3.1.5, this material has not been used to develop new theory. Instead it has been used to confirm the validity of the findings developed when analysing other qualitative empirical material.

3.2.6 Qualitative Data Analysis

The qualitative material collected in the above discussed projects initially consisted of both audio recordings of workshops and interviews, and written researcher logs. After transcription of the audio files into textual representation, both the transcripts and the researcher logs were analysed using qualitative methods as described in Section 3.1.3, through a process of fragment-based coding and analysis.

The recordings from the IMSK project were transcribed by the author, while most recordings from VALCRI, OSTAG, and SSyncAHD (except for those including sensitive content) were transcribed by an external company specialising in such work. The resulting transcripts consisted of a sequence of individual statements ranging from single word utterances through rather

extensive monologues. Each such statement was tagged with a speaker label and a timestamp.

Table 3.1: IMSK project qualitative analysis: 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		

The transcript and research logs were coded and analysed using different coding strategies, depending on the purpose of the analysis. For RQ1 (developing an understanding of ODP quality), an emergent coding strategy was used in the initial datasets from the IMSK project. The coding consisted of the author reading through the transcripts and organising a coding structure (i.e., a set of labelled tags) based on the content of the transcripts, before reading through the material again, this time applying those codes to the transcript content. Per this emergent approach, the codes were developed from the material in a grounded manner, which maps well to the need for developing a first set of indicators for the quality model. The distribution of codes to transcript fragments (each fragment includes one or more statements) is summarised in Table 3.1. The fragments were then grouped by code, and the collected material relating to each code analysed to see what conclusions could be drawn regarding participant experiences, opinions, and behaviour.

In the subsequent analysis of data from the VALCRI, OSTAG, and SSyncAHD projects, a predefined coding strategy was employed instead, where the codes used were developed from the partially developed ODP quality model. Specifically, the previously developed quality indicators were used as codes. However, new codes were added when this set was insufficient, and so we might in fact label the approach “partially emergent”. The final coding structure for the latter analysis is illustrated in Figure 3.6. Just as before, the fragments associated with each code were then grouped and studied, to develop trustworthy findings.

For answering RQ3 (concerning development of the eXtreme Design ODP usage methodology), an emergent strategy was again employed. The transcribed texts were studied, codes established, the texts were re-read, and the codes applied, before analysis based on the codes was finally performed. The emergent coding structure covering RQ3 issues is illustrated in Figure 3.7.

It should be noted that not all initially developed codes as displayed in Figures 3.6 and 3.7 are associated with results presented in Chapters 4 and 6: as discussed in Section 3.1, for findings developed using qualitative methods and analysis to be generalisable, triangulation of the findings may be required. For this reason, only such codes and themes as were identified in multiple project contexts were used in the subsequent analysis and solution development described in those chapters.

In addition to the thematic coding and analysis, in one case a more quantitative style—of analysis of speaker involvement or speaker role within two workshops—was also carried out (see the description of SSyncAHD project observations in Section 6.1.1). For this analysis, trivial one-word utterances were first filtered out of the dataset, before the remaining statements were grouped and summed by speaker, providing an image of the degree to which each of the involved speakers participated and drove discussion in the studied workshops.

3.2.7 Surveys Employed

In addition to the qualitative methods and analysis, three surveys have also been employed within the PhD project, to gather data of a more quantitative nature. In several cases, this data complements the gathered qualitative data, and in other cases validates the qualitative analysis (through triangulation, as discussed in Section 3.1).

In addition to the surveys detailed here, surveys using the System Usability Scale (SUS) have also been used to evaluate the XDP tooling developed within this project. The SUS scale is a very simple tool to rapidly evaluate usability of IT systems. While limited in scope, the ease with which an SUS survey can be deployed and analysed has lead it to see significant adoption in practice. More information about the SUS surveys employed is available in Chapter 5. For some respondent groups, these SUS evaluation surveys were integrated with or given simultaneously with either the second or third survey listed here.

Ontology Engineering Survey

The *Ontology Engineering Survey* was constructed and distributed in the autumn of 2014, after the second iteration of the ODP Quality Model had been developed, with the the goal of gathering data supporting both the further development of that quality model (i.e., RQ1) and the need for developments in ODP support tooling (i.e., RQ2). The survey contained a total of 44 questions, though six questions varied in such a manner that each respondent only responded to 38 questions—this was to ensure that users who responded that they had little or no experience of using ODPs were not asked about ODP usage, but about more generic ontology engineering instead.

The survey¹⁰ consists of six questionnaire pages (in addition to the opening and closing pages). The first page asks for background information about the respondent (age, academic degree, knowledge of semantic web standards, number of projects participated in, etc.). The second page asks about perceived industry uptake problems for semantic web technologies. The third page asks about the users' familiarity with and perceptions of ODPs. The fourth page asks about ODP usage challenges and user preferences regarding ODP tooling (or, in the case users have no experience of ODPs, similar questions about reuse of ontologies). The fifth page asks about user preferences about ontology visualisation, serialisation syntax, and what they use ontologies for. Finally, the sixth page asks about method issues, what type of tooling users prefer and what type of tooling they actually do use (these differ, as it turns out), and how their development teams are typically structured.

The survey was initially distributed during ISWC 2014, where it was marketed with an incentives program for participating, allocating (by lottery) Amazon gift cards to certain respondents. A total of 81 responses were received.

ODP Design Preferences Survey

The *ODP Design Preferences Survey* was developed as a follow-up to the *Ontology Engineering Survey*, with the goal of gathering more specific responses to questions from the previous survey with responses that proved to be interesting, but where the previous survey did not go into sufficient detail.

After the first page (gathering background information about the respondents), the survey contains one page with questions on user preference regarding how to find ODPs (what features users prioritise in an ODP search engine, what categories are most helpful in an ODP portal, etc.), followed by a page on how users prefer ODP documentation be constructed (which descriptive text fields are most important, which type of graphical notation is preferred, etc.).

The survey was distributed via projects that the author participated in from early 2015 through early 2016, and at an ODP tutorial the author tutored at the International Semantic Web Conference 2016. In total, it received 20 responses. Due to the relatively low number of responses, the resulting data was primarily used to elicit ideas and requirements for the development of ODP support tooling described in Chapter 5. However, some of the resulting data also feeds into the ODP Quality Model as discussed in Chapter 4.

¹⁰Available at <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-138724>

XD Workflow Survey

The last, and in terms of number of responses smallest, survey was the *XD Workflow Survey*. This survey sought to gather opinions from users who had applied the XD method, about how well they thought certain tasks in the XD workflow had worked in practice, how confident they were in performing those tasks, and how easy or difficult they found certain ODP features to be to work with. The survey consisted of eight questions with Likert scale answers. It was given in conjunction with modelling workshops within the VALCRI, E-care@Home, and SSynAHD projects, for a total of nine responses. Due to the relatively low number of responses, the data resulting from this survey was primarily used as supporting evidence confirming the validity of analysis and findings from other data gathering activities in the project.

3.3 Attributes of the Research Process

Section 3.1 introduces three attributes (*validity*, *reliability*, and *generalisability*) commonly used to evaluate research tasks. The subsequent section discusses the different types of research activities that have been performed, in terms of these attributes. While each type of activity is discussed separately below, it is important to keep in mind that the findings presented in this dissertation are typically sourced from multiple independent research activities that support one another, as detailed in chapters 4–6.

3.3.1 Workshop Observations

The workshop contributions consist of researcher logs and coded audio transcripts (for details, see Sections 3.2.4–3.2.6). These datasets have been used to develop theory regarding user opinions and requirements for ODPs and ODP quality, tooling, and methodology. The workshops have shared certain features, namely that they included relatively inexperienced users performing ontology engineering tasks, where the developed ontologies were intended to be used in some computational system. While a single observation from such a workshop might not achieve an acceptable degree of validity, the observations that have been presented and discussed in this dissertation as results are all based on recurring features or themes in the datasets, typically triangulated not only in terms of data source type (log or transcript) but also in terms of cross-case triangulation. Such triangulation guarantees an acceptable degree of internal validity for the findings.

The methods by which the workshops were performed and analysed are described in Sections 3.2.4 and 3.2.6, providing transparency to the research process and contributing to the reliability of the work. Unfortunately, owing to the sensitivity of some of the recorded materials and the ethical considerations of making semi-private conversations public, the source recordings

or transcriptions will not be made available to the world at large. While we might suspect that some findings from these workshops (e.g., emergent roles, challenges in reusing existing artefacts, etc.) could be generalised beyond the ontology engineering domain, such generalisability cannot be guaranteed without further study.

3.3.2 Surveys

The three surveys employed within this project, along with their contributions to answering the research questions, are presented in Section 3.2.7. The respondents to the surveys were anonymous and were not known to be close social acquaintances of the author, so they would have no obvious incentive to answer untruthfully. Consequently, the validity of the responses (i.e., the alignment of responses to respondents' actual opinions and experiences) is likely to be acceptable. Concerning the matter of reliability, the questionnaires that were used and the responses that were received have all been made available for download and validation by other researchers. However, for reasons of respondent integrity, the responses have been anonymised.

With regard to generalisability of survey results, we note that the largest survey in this PhD project (introduced in Section 3.2.7) elicited answers from the Semantic Web research community, and was marketed at research conferences and on the associated mailing lists. The respondents to this survey are thus likely to be representative of the Semantic Web research community, but this cannot be guaranteed. In total, this survey garnered 81 responses, which implies that for some questions, the margin-of-error may be around ± 12 percentage points. However, many questions had fewer responses, as did the other two surveys in this PhD project. While the findings that are based on survey responses are also supported by other evidence in most cases (quantitative or qualitative), the author cannot, based on the above, claim that the survey findings in total are statistically generalisable to the whole population of ontology engineers. Before employing methods or recommendations developed in this work, the reader is advised to test whether or not they appear valid in their usage context.

3.3.3 ODP Feature Studies

The ODP feature studies concern how certain performance-affecting indicators (Section 4.2.3) or specialisation strategies (Section 5.2) vary over a certain set of ODPs and ontologies that have been published on the web. The validity of these activities depends on design and quality of the employed measurement methods. Those methods have been thoroughly reviewed as part of prior publications¹¹—the former method has also been developed into an open source tool that is available for download and use¹². Given

¹¹Performance-affecting indicator measurement method via the author's Licentiate dissertation [76], specialisation strategy measurement method via [77].

¹²<https://github.com/hammar/OntoStats>

the availability of both the datasets and of the formally defined methods for measuring over the datasets, other researchers should be able to replicate the same findings—consequently, these findings can be considered reliable.

The results of these research activities are not inherently generalisable beyond the ODPs or ontologies that have been studied. However, it is likely that those results or effects which can be explained as resulting from formal logical structures in the OWL ontology language, can be generalised to other ontologies that use the same language.

3.3.4 Experiments

The experimental research activities performed within this project have attempted to test various hypotheses by evaluating them against the real world. Examples include testing whether certain ODP features correlate with improved user understandability (Sections 4.2.2 and 5.3), whether certain search strategies improve search engine results when operating over ODPs (Section 5.1), and whether the ODP specialisation strategy that is employed affects reasoning performance characteristics (Section 5.2).

Validity, in a quantitative experimental process, can be defined formally by way of statistical methods that measure the *p value*, that is, the likelihood of the observed result occurring by chance, rather than being caused by the experimental treatment. The experiments performed within this dissertation do not aspire to such degrees of validity. This is partly due to the relatively small scale at which the experiments have been performed (i.e., the dearth of participants or data), and partly due to an imperfect fit between the research problems being studied and the classical dependent/independent variable experimental structure. For instance, some experiments that show the feasibility of a method or an improvement over an existing method do not necessarily conform to this structure, nor do they generate large enough data sets for statistical analysis to apply.

The datasets used for each experiment, and the method used for the experiment, have all been provided for download or are included in this dissertation, so the work should be reliable enough for other researchers to replicate. However, the findings are not guaranteed to be generalisable beyond the experimental context—again, this is a consequence of the limited scale of the experiments in terms of subjects and datasets.

Chapter 4

ODP Quality Model

To answer the research question “*Which ODP features or qualities are important in supporting pattern understanding and use?*”, the author developed a quality model for Ontology Design Patterns, covering and including both general quality characteristics relevant to ODP development, and concretely measurable quality indicators contributing to those quality characteristics. Observing Hevner’s guidelines for Design Science Research (particularly *Guideline 6—Design as a Search Process*)¹, the quality model was developed over three iterations of evaluation and refinement, described in Sections 4.1–4.3. The final quality model is presented in Section 4.4 and its indicators are further detailed in Appendix A.

4.1 Initial Model

Rather than develop the quality model entirely from scratch, the author decided early on to take as input and inspiration established work on similar topics from neighbouring research fields, adapting that work to fit the specific characteristics and uses of Ontology Design Patterns. Such an approach grounds the initially proposed model in established theory and practice, and provides a solid starting point for further development. The following subsections describe this process, which contributed three distinct components to the quality model:

- A conceptual understanding or *metamodel* of quality as it applies to ODPs, influenced by the MAPPER [139, 32] metamodel (Section 4.1.1).
- A set of initial *quality characteristics* and subcharacteristics sourced from the ISO 25010 [94] Product Quality Model and from Thörn’s work on feature model quality [161] (Section 4.1.2).

¹See Sections 3.1.1 and 3.2 for more details and discussion on Design Science Research.

- An initial set of *quality indicators* contributing to the above quality characteristics, sourced from prior work on ER quality and ontology evaluation, as well as smaller studies with students at Jönköping University (Section 4.1.3).

4.1.1 Quality Metamodel Development

Many quality models have been proposed for various types of IT artefacts, as discussed in Chapter 2. 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 conceptualise and represent quality as it relates to whichever type of artefact they are intended to support. Thus, establishing such a metamodel structure upon which to build the ODP quality model was considered important from the outset, 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.

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 also needs to allow for the modelling of the IT artefact construction contexts in which the patterns are used.

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 [139, 32], introduced in Section 2.5.1. 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 from the one studied in this dissertation, the metamodel used is general enough to capture broader understanding of quality concepts in different fields. The most relevant perspectives taken from the MAPPER metamodel concern:

- The differentiation between measurable indicators and immeasurable general quality characteristics.
- The possibility of nesting different quality characteristics into a hierarchy.
- The idea that the metamodel is instantiated into a “filled out” quality model when applied.

- The importance of representing use case and context in modelling artefact quality.

These metamodel perspectives are compatible with the other quality models supporting this work, that is, the Thörn quality model for variability models [161] and ISO 25010 [94].

Development of the metamodel and the quality model with which it is populated was roughly sequential in that the metamodel was designed first and the quality model thereafter. However, some minor shortcomings in the metamodel, discovered during evaluation and development in the latter half of the work, were rectified and the metamodel was updated accordingly. The final metamodel, after these minor modifications, is presented in Figure 4.5 and in Section 4.4.1.

4.1.2 Initial Quality Characteristics

While the MAPPER deliverables influenced the metamodel design, the artefacts developed in that project were too different to from ODPs for the remainder of that framework to be reused. Instead, the initial quality characteristics were sourced from the ISO 25010 software quality standard [94] and from the PhD thesis “On the Quality of Feature Models” by Christer Thörn [161], as described below.

ISO 25010 Adaptation

The ISO 25010 [94] Product Quality Model (PQM) is introduced in Section 2.5.4. The majority of the quality characteristics of PQM are 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. 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 reasoning over 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 more closely to interoperability in terms of shared base concepts and lack of definition duplication, thus 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 can still perform that part correctly. However, in an ODP context this quality characteristic is redundant: a pattern can be considered correct only if it fulfils the requirements by which it is defined, that is, if it exhibits functional completeness.

- *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, they do not exhibit such behaviour.
- *Reliability*: As with the security characteristics, reliability and its sub-characteristics deal primarily with executable behaviour at runtime. 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 more closely to executable programs, and makes little sense in an ODP context.

Finally, the three ISO 25010 [94] PQM quality characteristics *maintainability*, *portability*, and *compatibility* display a certain overlap when translated to apply to ODPs. 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, and 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, the author found that the ones sorting under the top-level quality *compatibility* were most applicable to ODPs. The *portability* sub-qualities were either inapplicable to ODPs or could, in an ODP context, could be considered specialisations of other existing qualities. 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 further integrated 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*.

Thörn's Qualities

The Thörn quality model for feature models [161] is presented in Section 2.5.2. While the formal language used to create feature models is not as expressive

as Semantic Web ontologies and Ontology Design Patterns, the two types of models share certain characteristics (see Section 2.5.2) and intended usages, suggesting that quality characteristics for one should be applicable to the other also.

While the Thörn quality model does not include lower level quality characteristics, the preliminary first iteration of that model presented in [161] does define and argue for the existence of a set of more specific characteristics, several of which deal with issues that make them fit for adaptation and inclusion in the ODP quality model:

- *Accuracy*: “*This attribute describes how well the model represents the actual world*” [161]. While an ODP may be functionally complete and correct simply by fulfilling its design criteria (no matter what those criteria are), for ODPs, *Accuracy* such as proposed by Thörn would concern how consistent an ODP is with regards to the generally accepted understanding of the domain in question. In other words: is the pattern design criteria reasonable in the real world?
- *Consistency*: “*The attribute that determines the absence of contradictions in the model*” [161]. An ODP which holds or suggests conflicting axioms is obviously going to be difficult to apply in any real-world case that includes reasoning requirements, though it could still be useful for simple vocabulary tasks.
- *Stability*: “*This attribute denotes the perceived change expectation*” [161]. A stable ODP is developed with the intention of covering the foreseeable evolution of the modelled concepts with relatively few changes to the pattern.

Over the course of the PhD project, only rather minor modifications (some labelling and phrasing changes) were made to the initially developed list of quality characteristics. The final set of quality characteristics, including those modifications, is detailed in Section 4.4.2.

4.1.3 Initial Quality Indicators

At this stage of development, the ODP quality model held only quality characteristics, that is, abstract concerns or perspectives on ODP quality that were not directly measurable themselves (*Learnability*, *Reusability*, etc.). To make the model usable by scholars and practitioners, a set of measurable indicators affecting these quality characteristics was developed, drawn from existing literature on ER model quality [61, 120, 110] and ontology quality [56, 57, 105, 83], and from two smaller studies using university students, as described below.

Reuse of ER Model Quality Research

Several methods of evaluating ER models are introduced in Section 2.5.3. Some of these metrics and methods are also strong contenders for inclusion in an ODP quality model.

In [61] 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, the method by which the understandability and modifiability of models are gauged (via measuring the time taken to respond to a questionnaire on the functionality of the model, and the time needed to update said model), is easily transferrable to an ODP context. Consequently, *Functionality Questionnaire Time* and *Modification Task Time* measures were included in the initial ODP quality model as indicators for the learnability and modifiability sub-characteristics.

Both Moody and Shanks [120] and Lindland et al. [110] focus on the importance of reducing unnecessary content (that is, content that is not required for the model to be functional) in conceptual models. In an ODP context, unnecessary content would consist of entities that are not required to fulfil ODP design criteria (as formalised by competency questions). This type of *Minimalism* with regards to competency questions was also added to the model as an indicator.

Reuse of Ontology Quality Research

As has been covered in Section 2.6, a great deal of research has already been developed on formalising the quality of ontologies. A lot of this work is applicable to Ontology Design Patterns.

The combination of O^2 and $oQual$ by Gangemi et al. [56, 57] presented in Section 2.6.1 provides a comprehensive ontology evaluation method, including a set of indicators for measuring the structural dimensions of an ontology. While some of those indicators are unsuitable for use with ODPs (they measure features that are unlikely to occur in small or reuse-oriented ontology modules), many of them are also quite likely to be applicable to ODP evaluation, and were therefore added to the model. Listed by the quality characteristics they affect, those indicators are:

- *Usability*: Affected by *Subsumption Hierarchy Depth*, *Subsumption Hierarchy Breadth*, *Tangledness*, *Anonymous Class Count*, *Class/Property Ratio*, and *Annotation Ratio*.
- *Analysability*: Affected by *Size* and *Axiom/Class Ratio*.
- *Resulting performance efficiency*: Affected by *Class Disjointness Ratio* and *Tangledness*.
- *Compatibility*: Affected by *Tangledness*.

The effect of the amount of integrated pattern documentation (in the form of pattern comments) on usability is shown by Prechelt et al. [130], as described in Section 2.5.5. This measure is roughly translatable to the above listed *Annotation Ratio* indicator sourced from [57]. Prechelt et al. also report an experiment showing that this measure influences the maintainability of produced solutions. Accordingly, a positive effect of a high *Annotation Ratio* on maintainability was also added to the initial ODP quality model.

The findings on performance-related indicators associated with the use of certain types of design patterns by Lefort et al. in [105] discussed in Section 2.6.4 are natural candidates for inclusion in an ODP quality model. Those indicators (the *Terminological Cycle Count* present, and the *Complexity of Description Logic Language* used) were added to the initial quality model indicators that affect resulting performance efficiency.

In prior work [83], the author has discussed the effects of the number of `owl:imports` statements in an ontology. 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 easily become very significant. Because of this, an ontology or ODP that imports many other ontologies will likely require significant resources to classify using a DL reasoner. Furthermore, due to tooling limitations in managing the display of imported concepts, such an ontology can be difficult to visualise and work with. Consequently, the effects of *Transitive Import Count* on usability and reasoning performance were added to the initial quality model.

As mentioned in Section 2.6.3, the OntoClean [68] methodology is an established method for ontology evaluation. While applying OntoClean to larger ontologies is potentially a very time-consuming process, for Ontology Design Patterns this ought not be as big a problem. *OntoClean Adherence* would likely be a suitable indicator for pattern accuracy, and therefore this indicator was added to the quality model.

A master thesis by Lodhi and Ahmed [111], introduced in Section 2.6.5, finds that certain fields in ODP documentation are more important than others in supporting the learnability of the patterns. The author’s intuition is that in light of this, it is possible that the less important fields (of which there are quite a few) may be 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* was added to the initial quality model; this indicator 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 [111].

Student Studies

To elicit additional indicators for the initial quality model, the author also ran two small studies on the use of ODPs by students at Jönköping University. The students in question were all in the final year of a two-year master programme, with a bachelor in computer science or computer engineering

as an entry requirement. They all had some experience with semantic technologies or ontologies through the courses given in the programme, but none reported having worked with semantics prior to attending the programme. Several of the students had work experiences in entry-level programmer positions, in addition to their educational background.

The first study consisted of interviews with a student performing a master thesis project using ontologies and ODPs in the healthcare domain. The student was provided with two such patterns and references to several more, and the interviews were performed to gauge his understanding and opinion of these patterns.

The student strongly preferred task-oriented ODP documentation, preferably with clear and pedagogic examples included. He suggested that more than one usage example and corresponding context be included in pattern documentation, arguing that such pattern usages are often case-dependent and that an understanding of several types of application scenarios would often be beneficial. An indicator measuring the *Usage Example Count* was thus added to the initial ODP quality model.

In terms of the structure of the pattern documentation, the availability of graphic illustrations was also heavily emphasised. The student expressed a preference for 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 a pattern *Structure Illustration* and one or more *Usage Example Illustrations* should be included in ODP documentation. Both of these indicators were added to the initial model. The student also emphasised that the graphic illustrations need to be backed by descriptive text referencing their content rather than being entirely disconnected artefacts.

The second study took place in an Information Logistics course, where a group of 29 students were tasked with building an ontology using patterns and then filled out a survey regarding that pattern usage experience. Responses indicate that the *Competency Question Count* and the availability of graphical illustrations were most important in understanding how to use and apply the ODPs. When asked to suggest improvements to the patterns that were used, the most common recommendation from the participants was to add more usage examples. These findings are in line with those from the aforementioned student interviews.

A somewhat unexpected result of the 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 any such domains or 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. As a result, two indicators measuring *Property Domain Restriction Ratio* and *Property Range Restriction Ratio* were

added to the initial model. These indicators are also associated with lower reusability, given that restricting the use of ODP properties to only work with classes designed within that same ODP will reduce the ways in which that ODP can be integrated into other ODPs or ontology modules.

4.2 Second Generation Model

The second generation of the ODP quality model was developed via iterative evaluation and development of the initial model within three separate studies:

- In the IMSK project (Section 4.2.1) effects on ODP *Usability* and *Compatibility* were studied via a workshop exercise with project participants.
- In the ILOG Course Study (Section 4.2.2), effects on *Usability* were further studied via a combination of experiments and surveys with master student participants.
- Finally, in the Performance Indicator Evaluation (Section 4.2.3), possible ODP effects on *Resulting Performance Efficiency* were studied via a literature review and via study of how indicators triggering such effects are displayed in ODPs published on the Internet.

4.2.1 IMSK Workshop

The IMSK project (introduced in Section 3.2.4) provided the author with the opportunity to gather data on ODP usage via a two-day workshop exercise involving three participants developing ontology modules for project scenarios². In particular, the author aimed to learn about how the participants found and selected suitable ODPs to use, and which characteristics or features of ODPs the participants found helpful or harmful when employing ODPs. For the sake of establishing transparency and trust, this research motive was presented to the participants of the workshop at the outset, and they willingly accepted acting as subjects in such observations.

Participants

Three participants attended the modelling workshop, participants A, B, and C. They were all male, and at the time of the study were all between the ages of 35 and 55. All three were researchers (two PhDs, one MSc) in software engineering or conceptual and data modelling within the partner research institute, and all three had some experience of such modelling. B and C

²The ODPs and modelling scenario descriptions used in this workshop, and the interview manuscript used, are downloadable from <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-138724>.

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.

Workshop tasks

During the workshop the participants developed configurations for the ODP-based variant of the CEP system. Two scenario descriptions developed within the project were used to describe system deployment contexts. 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 been spotted within the last hour 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 by three different cameras.*

To aid them, the participants had a set of twenty Ontology Design Patterns, of which fourteen were selected from the ODP community portal³, and six were selected from other research projects or literature. They were not provided with any training in pattern use, and no particular development method was recommended to them, 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.

³<http://ontologydesignpatterns.org>

Data Collection and Analysis

Data was gathered by way of audio and video recordings of the modelling work in progress, photographs taken of ontology prototypes on the white-board, and notes 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 about 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 work process on which data was being gathered.

At the end of the second workshop day a semi-structured group interview was held in which the participants were queried about 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 where the subjects expressed diverging opinions. Such situations were instead noted and kept for analysis.

Upon completing the workshop, the recorded material was transcribed into text, yielding some 21600 words, or approximately 85 pages of text. This material was then analysed according to established transcript analysis methods [31], as discussed in Section 3.2.6.

Findings

The gathered data supports several of the initial quality model indicators, and supports the inclusion of one additional indicator.

Pattern Selection The single most important variable in initial 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 for their case. This motivated the addition of a quality indicator measuring *Name Appropriateness*. In terms of documentation fields and features, the participants then suggested that the presence of descriptive texts and the number of competency questions were important selection criteria that should be emphasised in an ODP catalogue. While the initial quality model already contained a quality indicator measuring the *Competency Question Count*, it contained no such indicator

for the presence of a descriptive text. Since several published ODPs actually do not have this rather vital documentation field filled out, an indicator indicating the presence of an *Accompanying Text Description* was added to the model.

The participants considered the possible negative consequences of applying a certain pattern to a problem to be of particular importance in selecting and applying patterns. The latter observation led to the addition of a new quality indicator concerning the presence of a *Common Pitfalls Description* in the ODP documentation.

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, containing 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 illustrations of pattern dependencies, and providing a semantic search engine across ODPs stored in the catalogue. The former suggestion was inspired by an illustration from the Core J2EE Patterns book [3] that the participants found helpful in deciphering pattern intent. Participant C in particular argued that such an illustration would help to clarify the structure and interdependencies of a set of ODPs. The latter suggestion was that a search engine be added which would allow users to search through concepts and properties present in ODPs in the catalogue, ideally including NLP techniques to match synonyms and related terms. This finding contributed to the development of the work presented in Section 5.1, which presents such search functionality.

Important ODP Features During both modelling and the subsequent interview, the issues of ODP *Size* and ODP *Transitive Import Count* were brought up. The participants initially expressed divergent opinions regarding the effect of OWL import statements in ODPs. Participant A considered imports to be 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 also on the grounds that the base concepts included in imported patterns may be incompatible with one's own world view, having been 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 that was 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. This observation is one of the underlying motivations for the development of the template-based approach to ODP instantiation introduced and discussed in Section 5.3.

In terms of ODP *Size*, during the interview session the participants emphasised 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 to 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.

4.2.2 ILOG Course Study

To further evaluate learnability- and usability-related indicators of the initial quality model, a study was set up in the context of a course in Information Logistics at Jönköping University in 2012⁴. The study aimed to evaluate the effects of six indicators from the initial model, with the hypotheses (based on effects proposed in the initial model) that:

- Patterns including *Usage Example Illustrations* are superior to ones with examples simply written in text, in terms of learnability.
- Patterns displaying *Documentation Minimalism* (i.e., limiting the documentation fields displayed to those found to be most important in

⁴The ODPs used, the surveys and task instructions employed, and anonymised datasets resulting from those surveys and tasks, are downloadable from <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-138724>.

[111]) are superior to patterns not displaying this feature, in terms of learnability.

- Patterns that display a high *Anonymous Class Count* (i.e., restrictions) are more difficult to apply than patterns that contain low numbers of anonymous classes.
- Patterns that have a high *Class/Property Ratio* are easier to apply than patterns that display a low ratio of this type.
- High *Property Domain Restrictions Ratio* and *Property Range Restrictions Ratio* in ODPs are beneficial to the learnability of said ODPs.

Setting

The study was performed at Jönköping University, within a master course in Information Logistics. This course took place during the second year of the master programme in *Information Engineering and Management*, and the students taking it had taken courses on knowledge modelling and knowledge management, database systems, and software engineering methods earlier in the programme. In these courses they had studied and used the Semantic Web and ontologies, as well as ER and UML models. Additionally, the programme entry requirements included at least a bachelor's 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 also be filled out 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. In **Survey 1**, the participants were presented with a randomised order of four ODPs and asked to answer a number of questions about them, measuring their comprehension of the pattern in question. Each pattern displayed was presented using a template mechanism, by which the two controlled variables, *Documentation Minimalism* and *Usage Example Illustrations*, were randomly adjusted for each participant. The questions were of two forms; firstly, the participants were asked to mark which out of five competency questions the pattern could answer, 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, to provide 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.

Table 4.1: ODPs used, their class to property ratios, and anonymous class counts.

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

In the **Tasks** portion of the study, participants were tasked to use ODPs to help model four different scenarios (using one ODP per scenario), and were timed on how long it took them to complete these tasks. 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.

Finally, in the concluding **Survey 2** portion of the study, the participants were surveyed on their opinions and impressions of the same ODPs and their features, now that they had used them in modelling. 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, harmful, or very harmful in understanding and using the patterns provided to them.

The patterns used in all three parts of the study are listed in Table 4.1. The ODPs in question were sourced from the ODP portal⁷ and were selected based on three criteria: they were of non-trivial size (i.e., containing a minimum of 10 classes including imports), they were intended to be generic as opposed to domain-specific, and they varied with regard to their *Class/Property Ratio* and *Anonymous Class Count*. For comparison, 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.

Data

The most interesting results from the first survey, regarding documentation-related effects on learnability, are summarised in Tables 4.2, 4.3, and 4.4. In all of these tables, the participant responses have been averaged and grouped by the indicator under study.

Table 4.2 indicates how well participants were able to match a pattern to five given competency questions. 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

⁵<http://protege.stanford.edu/>

⁶<http://www.topquadrant.com/>

⁷<http://ontologydesignpatterns.org>

Table 4.2: Competency question recognition correctness ratio (survey 1).

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 4.3: Class recognition correctness ratio (survey 1).

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 %

group. The presence of an illustrated example does not seem to correspond to any increase in correct responses.

Table 4.3 indicates how well participants were able to match a term in the description of a given scenario 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. The presence or absence of illustrated examples do not correlate with any such results.

Table 4.4 averages the time taken by participants to answer survey 1. 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 for illustrated examples.

Table 4.5 also reports 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 abstract” and “very easy” (scoring 1) through “very concrete” and “very difficult”, (scoring 5).

The times required to model the provided scenarios using the given pattern are detailed in Table 4.6. When cross-referencing this table against the ODP characterisations in Table 4.1, we can see that the two patterns having the lowest *Class/Property Ratio* (*Communication Event*, *Invoice*), are the ones for which the associated modelling exercise took the least time to complete.

Table 4.7 shows the most interesting results of the second survey on how

Table 4.4: Time required (in minutes) to answer questions (survey 1).

Group	Com. Event	Reaction	Invoice	Basic Plan
Minimal	14.8	7	4	13
Non-minimal	11.3	10	8.8	13
Illustrated	13.1	7.8	8.4	11.8
Non-illustrated	11	8.3	5.2	13.7

Table 4.5: Average user-reported concreteness and usage difficulty values (survey 1).

ODP name	Concreteness	Difficulty
Communication Event	2.55	3.27
Reaction	2.55	2.82
Basic Plan	2.91	2.73
Invoice	3.55	2.3

Table 4.6: Average time required (in minutes) to complete modelling tasks with ODPs.

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 participants perceived the studied indicators and their effects on the usability and learnability of the patterns in question. The indicators being studied (abbreviated in the table) were the presence of a *Usage Example Illustration* in pattern documentations, the use of *Property Domain or Range Restrictions* with properties asserted in the patterns, and the existence of property restrictions on classes in the pattern (the latter being a contributor to *Anonymous Class Count*). For the first three indicators, participants were asked how the presence of the related features helped or harmed in understanding the patterns, whereas for the last indicator, they were asked how the presence of this feature helped or harmed in using the patterns.

Findings

The low number of participants (12) implies that the generalisability of the findings to the greater population of ontologists is very limited. The data gathered thus gives some useful indications regarding the possible validity of the original hypotheses, but we cannot say that we have tested them in the statistical sense of the word.

The hypothesis that *patterns including Usage Example Illustrations are superior to ones with examples simply written in text in terms of learnability*

Table 4.7: Perceived usability and learnability effects of indicators (survey 2).

Effect	Illustrations	Range res.	Domain res.	Property 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 %

is partially supported by the observation that a large majority of study participants rank the presence of illustrated examples as *Helpful* or *Very helpful* (Table 4.7) 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 4.2, 4.3, and 4.4) gives 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 displaying documentation minimalism performed better than participant groups seeing non-minimal variants. However, the small number of participants and patterns tested severely limits the generalisability of the figures presented, so this should be considered as initial indications rather than evidence.

That *patterns that display a high Anonymous Class Count are more difficult to apply than patterns that contain low numbers of anonymous classes* is not supported by the gathered data. In fact, the data from survey 2 presented in Table 4.7 indicates that the opposite may be more correct, and that anonymous class definitions (when used to encode property restrictions) improve usability. As that table shows, a large majority of study participants found the presence of property restrictions in an ODP to be *Helpful* or *Very helpful* in terms of usability. The question on property 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. This finding lead to replacing the initial *Anonymous Class Count* indicator with a new indicator measuring *Property Restriction Count*.

The hypothesis that *patterns that have a high Class/Property Ratio are easier to apply than patterns that display a low ratio of this type* is partially supported by the data exhibited in Tables 4.6 and 4.1, which shows that the two patterns displaying the highest Class/Property ratio are the ones for which the modelling tasks were completed most quickly. 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 *High Property Domain Restrictions Ratio and Property Range Restrictions Ratio in ODPs are beneficial to the learnability of said ODPs* is supported by the results shown in Table 4.7. 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. The questions on these indicators were associated with open

questions, where participants indicated that they had given these responses because the restrictions helped clarify the intended usage of properties. As such restrictions are likely to have constraining effects on the reusability of patterns, it is important to study whether the usability gains they provide could instead be achieved by other means, for instance by improved pattern documentation pages or RDFS labelling and comments.

In addition to the original hypotheses, an unforeseen effect was observed that is relevant to the ODP quality model development. The participant-reported values for ODP concreteness and those for difficulty of ODP application presented in Table 4.5 seem to be inversely related. While the issue of the abstraction/concreteness of patterns was avoided in the initial quality model, this finding lead to the addition of a new quality indicator affecting usability: *User-reported Abstraction Level*.

4.2.3 Performance Indicator Evaluation

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. First, the indicators from the initial quality model that were believed 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 any 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 in online ODP portals, were studied and analysed to learn whether these indicators vary in practice, and how.

Literature review

Publications from 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. All papers matching the above criteria were downloaded, and their abstracts were 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.

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), *subsumption hierarchy indicators* (i.e., indicators related to the structure of the subsumption tree), and *axiom usage indicators* (i.e., general indicators related to the other logical axioms employed in an ontology). Each of these categories and the indicators found to be associated with them are discussed below.

Profile indicators Urbani et al. discuss the issue of scaling out description logic reasoning on parallel computing clusters using the MapReduce framework. They show in [165] that materialising the closure of an RDF graph using RDFS semantics can be performed using MapReduce, due to certain characteristics of the RDFS semantics. As shown in [164], the increased expressivity of OWL means that implementing such parallelisable reasoning over datasets based on OWL ontologies is significantly more difficult than when using RDFS. However, the authors present a solution that enables reasoning within the Horst fragment of OWL which is considerably faster than previous solutions.

In [91] 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 that have been published on the Web. The three profiles are subsets of OWL 2 intended for different usages. By limiting the semantics used, both in terms of allowed axioms and the positioning and use of those axioms, the OWL 2 profiles can guarantee computational properties that are suitable to different practical uses. Horridge et al. [91] find that relatively few ODPs fit these profiles, and that this may be due in part 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 (*Complexity of Description Logic Language*), 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 instead to consider the problem from a perspective of multiple indicators: a set of *OWL 2 Profiles Adherence* indicators and an indicator of *OWL Horst Adherence*.

Subsumption hierarchy indicators Kang et al. [98] perform a thorough evaluation of the effects of several 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 refer to as “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 is denoted *Tangledness* in the quality model. By showing that tree impurity contributes to reduced computational efficiency, Kang et al. [98] thereby also validate that *Tangledness* contributes to lowering the same quality characteristic.

In [107], LePendu et al. study the characteristics of both ontologies and data in the biomedicine domain, with a goal of improving reasoning performance. One of the metrics they find to have a high impact on reasoning performance is the *Subsumption Hierarchy Depth*. The explanation given for this is that for every asserted instance of a subclass, all the logic axioms pertaining to each and every superclass must also be calculated. For a deep ontology, this may be a quite significant number of entailments that need to be computed. Kang et al. [98] also study the depth indicator, and like LePendu et al. find that it contributes to slower reasoning performance. While the *Subsumption Hierarchy Depth* indicator was included in the initial quality model, it was not initially associated with such an effect on reasoning performance. This effect was thus added to the model.

Most of the metrics whose effects are studied by Kang et al. [98] are first defined by Zhang et al. in [174]. This paper provides an interesting reflection on the importance of including anonymous classes when computing values for subsumption hierarchy 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 indicators. Hence, both subsumption hierarchy indicators in the quality model would need be measured both with and without the effect of anonymous classes taken into consideration.

Axiomatic indicators As mentioned above, Kang et al. [98] evaluate the performance effects of a number of metrics (most of which are presented by Zhang et al. in [174]). They find eight indicators that show performance-altering effects, four of which can also be easily applied to ODPs, and which were thus added to the quality model:

- *Existential Quantification Count*: The number of existential quantification axioms in an ontology or ODP. This is most easily 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 when classifying said graph.

- *Average 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.
- *Average Class Out-Degree*: The inverse of the above indicator, that is, the average number of outgoing edges to classes in the ontology.

Indicator Variance in ODP Repositories

The second part of this performance indicator evaluation consisted of studying how the values of the proposed performance-related indicators were distributed in the ODPs available in the pattern repositories used by the community, and of identifying causes of such distribution that might contribute to the quality model.

For this purpose, 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 studied⁸. A plugin-based tool for measuring ontology or ODP metrics was developed⁹ specifically for this purpose. Plugins for the performance-related indicators under study¹⁰ were developed for this tool, and it was then executed over the downloaded pattern set.

In analysis, 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 all 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.

Findings

The results of the analysis, and its effects on the initial quality model, are detailed below.

⁸The ODPs used and the values of the indicators studied, are available at <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-138724>.

⁹<https://github.com/hammar/OntoStats>

¹⁰With one exception: it proved practically infeasible to develop software reliably measuring *cyclomatic complexity*, and rather than make assertions based on possibly faulty data, this indicator was left out of the analysis.

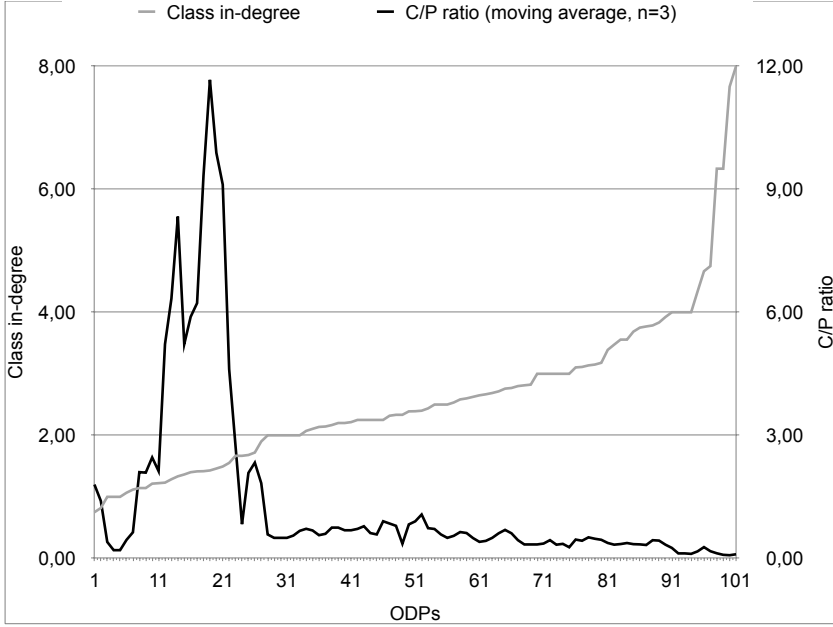


Figure 4.1: Class in-degree and Class to property ratio distributions.

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 4.1. 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 stand out with larger values.

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/Property Ratio* indicator from the ODP quality model were mapped against the values of the *Average Class In-Degree* indicator. The inverse of the former being a size-wise normalised measure of the number of properties in the ontology, if the observation holds then such a mapping should indicate the existence of an inverse correlation between the two mapped indicator value series.

The results, as shown in Figure 4.1, indicate that such an inverse correlation ($r=-0.375$) exists: the patterns displaying highest *Average Class In-Degree* have a lower *Class/Property Ratio* (i.e., contain more properties

per class as posited above), and many of the patterns displaying low *Average Class-In Degree* have a higher than average *Class/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 that 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 can be made to the effect of limiting the number of domain and range definitions used in performance-dependent ontologies, and the initial ODP quality model was revised accordingly. However, there may also be other as yet unknown causes besides domain and range definitions that give rise to high *Average Class In-Degree*, and given the observation on variability in this indicator in the observed patterns, including the indicator itself 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 3.67, with a median value of 2.75 and an average of 2.63. The distribution of indicator values over the whole pattern set is shown in Figure 4.2. The reason that all patterns exhibit a value of at least one is simply that all defined classes by definition have at least one outgoing `rdfs:subClassOf` edge to another class (either a direct ancestor, or an implicit link to the top-level `owl:Thing` class).

In studying some patterns displaying low or high values, it was observed that the patterns displaying a higher value seemed to be patterns in which property restrictions on classes were used extensively. Property restrictions are written as a class being asserted to be either a subclass of or equivalent to an anonymous restriction class, which would explain this observation—each `rdfs:subClassOf` or `owl:equivalentClass` axiom adds an outgoing edge, increasing the value of the indicator.

To test whether this explanation is supported by further evidence, the numbers of anonymous classes in the ODPs were plotted against the values of the *Average Class Out-Degree* indicator. The results are presented in Figure 4.2 which indicates a correlation ($r=0.441$) between these indicators.

Since the number of property restrictions has been shown in earlier chapters 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 justified.

Subsumption Hierarchy Depth As mentioned in previous sections, it can be important to measure both asserted and inferred versions of structural indicators. Due to the difficulty of measuring the inferred indicators

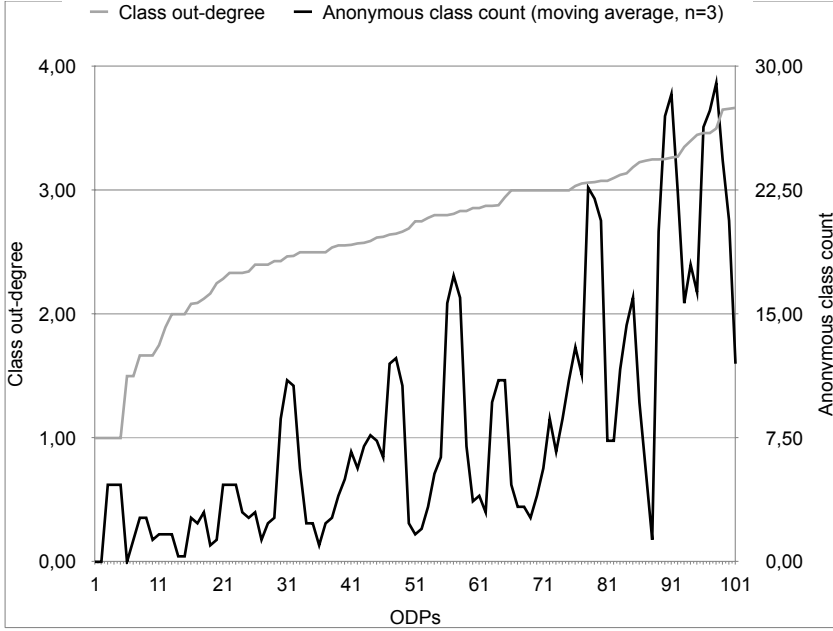


Figure 4.2: Class out-degree and Anonymous class count distributions.

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 `owl:Thing` if no other superclass is asserted within the same OWL file¹¹.

The *Subsumption Hierarchy Depth* of the patterns varies from 0 to 4.8, with a median value of 1.4 and an average value of 1.6. In other words, most of the patterns are not very deep. The distribution of values across the patterns studied is displayed in Figure 4.3. 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 be made up of two types of patterns. The first type consists of simpler domain specific vocabularies that do not employ much expressive logic, but rather act as schemas for simple datatypes that may be reused. Examples include patterns for species habitats, invoices, etc. The second type consists of very general patterns that define abstract or intangible phenomena without going

¹¹To study the code used to calculate the depth metrics, the reader is referred to <https://github.com/hammar/OntoStats/tree/master/plugins-structural>

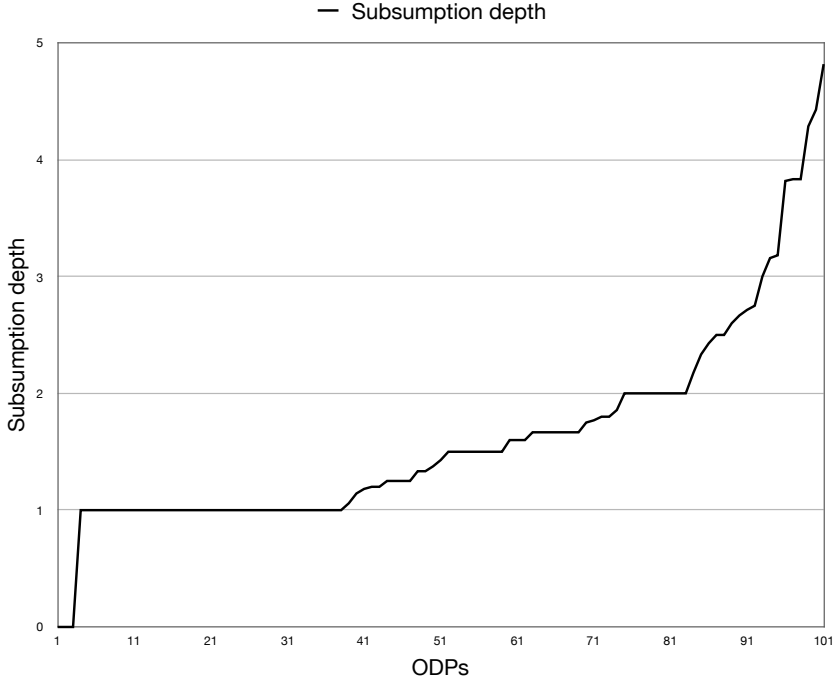


Figure 4.3: Subsumption depth indicator variance.

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, 49 of 101, contain no explicit existential quantification axioms. If cardinality restrictions are rewritten into semantically equivalent existential restrictions as suggested in [91], the number of patterns containing no existential quantification axioms drops to 41. Of the 60 patterns that contain such axioms

half, 31, contain one or two existential quantification axioms each. Studying these 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 axioms that were already asserted on their superclasses, and existential quantification used to assert the coexistence of two individuals where it seems that one individual might well exist on its own. These observed suboptimal uses of computationally expensive existential quantification axioms clearly justify the inclusion of the *Existential Quantification Count* indicator in the ODP quality model—it seems ODP developers need to be reminded not to take the use of this type of axiom lightly.

Tree Impurity / Tangledness Of the 101 studied patterns, only three display any degree of asserted tree impurity or *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 been infeasible to measure in this study for technical reasons, its effect on the performance of reasoning may be considerable, and for this reason the *Tangledness* indicator is kept in the ODP quality model.

4.3 Third Generation Model

The second-generation quality model was subsequently developed into a third-generation model based on findings from two survey studies, and on observations at a set of ontology engineering workshops, discussed in the following subsections:

- The Ontology Engineering Survey (Section 4.3.1) and the ODP Design Preferences Survey (Section 4.3.2) studied the effects of ODP documentation on ODP *Usability*, and user preferences on how to structure and present such documentation.
- The Ontology Engineering Workshops (Section 4.3.3) took place in three different projects. Observations from these workshops support quality indicators contributing to ODP *Usability* and *Maintainability*.

Table 4.8: Percentage of ODP-experienced respondents considering the respective component *Very important* or *Critically important* when evaluating the suitability of an ODP for reuse (29 responses).

	Some exp.	Confident	Expert	All
Example uses	100 %	75 %	88 %	86 %
Description	75 %	50 %	81 %	71 %
Competency Questions	75 %	25 %	75 %	61 %
Graphical Illustration	75 %	50 %	56 %	57 %
Title	0 %	13 %	56 %	36 %
OWL 2 Profile Adherence	25 %	13 %	13 %	14 %
Size in Classes	0 %	0 %	19 %	11 %
Size in Axioms	0 %	0 %	13 %	7 %

4.3.1 Ontology Engineering Survey

This survey (further detailed in Section 3.2.7) was carried out to gather knowledge about practices and problems in Ontology Engineering, particularly as related to the use (or lack of use) of ODPs¹².

The survey had a total of 81 respondents, who were asked to identify themselves based on their level of experience of Ontology Engineering using Semantic Web ontologies as ‘Expert’, ‘Confident’, ‘Somewhat experienced’ and ‘Novice’. In addition, respondents were asked how many years they had been working with Semantic Web ontologies. While self-reported skill level is a notoriously inaccurate measure, we found that the correlation between self-reported experience level group and reported years of experience was very clear, and have therefore elected to treat the self-reported experience level as trustworthy. There was only one respondent who self-reported as being a novice—to avoid this outlier skewing the results in analysis, that response has not been used. It should also be noted that as some respondents skipped some questions, not all tables of responses detailed below are based on the full set of 80 non-novice respondents. For legibility, the tables are presented using only percentages of respondents, while the number of respondents per question is included in each table’s label.

Table 4.8 reports the responses to the question “*In an ODP search engine or an ODP portal/catalogue, which fields or metadata about an ODP is most important when ascertaining the suitability of that ODP for reuse?*”, answered on a 5-point scale ranging from “Not important” through “Critically important”. Only respondents who reported having used ODPs to some extent were given this question, so the number of respondents is fewer than the number for the survey as a whole, at 28. Somewhat surprisingly, the title of the ODP was not considered particularly important in ascertaining the reuse potential of an ODP. This may be due to the way the question

¹²The survey questionnaire and a summary of all of the answers are downloadable from <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-138724>.

Table 4.9: Respondent opinion on the importance of graphical illustrations in understanding ontology meaning (65 responses).

Answer alternative	Percent of responses
Not important	2 %
Somewhat important	12 %
Rather important	23 %
Very important	45 %
Critically important	18 %

Table 4.10: Respondent preference regarding type of graphical illustration to use (61 responses).

	Strongly dislike	Dislike	Neither	Prefer	Strongly prefer
Graph-oriented	0 %	5 %	6 %	46 %	43 %
Tree-oriented	3 %	8 %	22 %	54 %	14 %
Venn diagram-oriented	8 %	31 %	41 %	16 %	3 %

was phrased; when selecting one ODP out of a list of many, the title is likely to be of more importance, whereas when studying only one ODP, the title is likely to matter less than other more descriptive fields.

It is interesting to note that three of the four top-most ranked fields are identical (though ranked slightly differently) to the top-ranked components from the similar survey performed by Karima and Hitzler [99]. The one field that does not match, *Description*, has no direct equivalent in that other survey. These findings support several indicator effects proposed in the ODP quality model, namely that the *Usage Example Count*, *Accompanying Text Description*, *Competency Question Count*, and *Structure Illustration* contribute positively to the quality characteristic *Appropriateness Recognisability*.

Table 4.9 reports the responses to the question “*When studying an ontology, how important is the use of a graphical illustration for understanding the meaning of the ontology?*”. While this question does not ask specifically about ODPs, the responses ought to be transferrable to an ODP context if we consider ODPs as mini-ontologies. The responses clearly indicate that graphical illustrations of the structure of an ontology contribute to the learnability of that ontology, or, in the case of the ODP Quality Model, that the presence of a *Structure illustration* contributes to the *Learnability* quality characteristic.

A follow-up question, presented in Table 4.10, further explored this indicator. The question asked, “*When studying an ontology, which type of illustration format do you prefer?*” (examples of each kind of illustration were provided alongside the question). The three types of formats provided represent three different perspectives often found in ontology literature: the

Table 4.11: Ranking of documentation field utility in ascertaining ODP reusability.

Documentation Field	Position Score
Intent	5.91
Competency Questions	4.64
Name	4.09
Solution	3.82
Scenarios	3.45
Domains	3.18
Consequences	2.91

ontology as graph (either an RDF graph or a more high-level structure), the ontology as a subsumption tree, and the ontology as a Venn diagram of super- and sub-concepts. As the table shows, the respondents expressed the greatest preference for graph-style illustrations, followed by trees, and finally Venn diagrams.

4.3.2 ODP Design Preferences Survey

This survey (described in Section 3.2.7) queried for ontologists’ preferences and requirements on tooling to support ODP use. The survey was answered by participants in projects and at academic conferences the author attended, with a total of 20 responses collected in all¹³. The respondents vary in academic background (ranging from MSc degrees through full professorships), and in self-reported Ontology Engineering experience (ranging from novices to experts). The large majority of respondents (16) work in academia or research institutes, with a minority (4) working in industry or government agencies.

Table 4.11 reports a summary of responses to the question *“Please rank the following ODP documentation fields in terms of how important you think they are to understanding whether an ODP is suitable for reuse in your project”*. As this question was accompanied by a ranking widget in the survey questionnaire, respondents (unlike in the survey discussed in the previous section) had to prioritise between the different provided alternatives and provide a preferred ranking of the seven options. The overall position score for each option was calculated by multiplying the score value of each position (7 for first place, 1 for last) with the percentage of respondents who placed the field at said position.

Some of the fields provided are more specific and narrow than those listed in the previous survey, as they are in fact candidate subfields for

¹³The survey questionnaire can be downloaded from <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-138724>. However, due to assurances given to participants, the individual responses will not be published online, though they are retained in accordance with Swedish legislation on archiving research data.

Table 4.12: Percentage of respondents preferring documentation minimalism (responses grouped by respondent self-reported experience level).

ODP	No experience, novice, some experience	Confident, Expert
Agent role	57 %	17 %
Componency	64 %	17 %
Reaction	57 %	17 %
Average	60 %	17 %

inclusion under that survey’s Description documentation field. We again note that competency questions are deemed highly important in ascertaining the reusability of a given ODP. We also note that ODP name is considered quite important in recognising the suitability of an ODP, lending further support to the value of the *Name Appropriateness* indicator. Finally, we note the importance of the ODP intent field in ascertaining reusability. In the ODP portal the defined purpose of this field is, somewhat ambiguously, to “[describe] the goal of the *Ontology Design Pattern*”¹⁴. Because of these findings, a recommendation was added to the ODP quality model concerning the structure of the descriptive text provided with an ODP.

To evaluate the possible effects of the *Documentation Minimalism* indicator on ODP learnability, the respondents were asked to compare two different documentation perspectives on the same ODP, for three different ODPs. The first perspective included all the documentation fields presently used in the ODP community portal, whereas the latter one displayed a documentation-minimal view (i.e., including only those documentation fields found by Lodhi and Ahmed [111] to be most important for Learnability). Summarising and analysing the responses, we note two things: firstly, that 18 of 20 respondents indicated the same preference regardless of which ODP was displayed (i.e., these preferences seem to be quite strongly held), and secondly, that user preference on documentation minimalism correlates clearly to self-reported experience level, as shown in Table 4.12. Experienced users seem to prefer non-minimal documentation, whereas inexperienced users are somewhat more in favour of limiting the number of documentation fields displayed. This is consistent with the findings by Lodhi and Ahmed [111].

The survey also queried respondents on their preferences regarding the type of notation used to represent ODPs graphically, providing them with pairwise comparisons of the recently developed VOWL notation [112] versus four other notations supported in Desktop Protégé, TopBraid Composer, or used in the ODP portal, and asking them to select which out of every pair they found most intuitive or easy to understand. The responses are reported in Table 4.13. Each notation is illustrated in Figure 4.4 (note that in the

¹⁴<http://ontologydesignpatterns.org/wiki/Property:HasIntent>

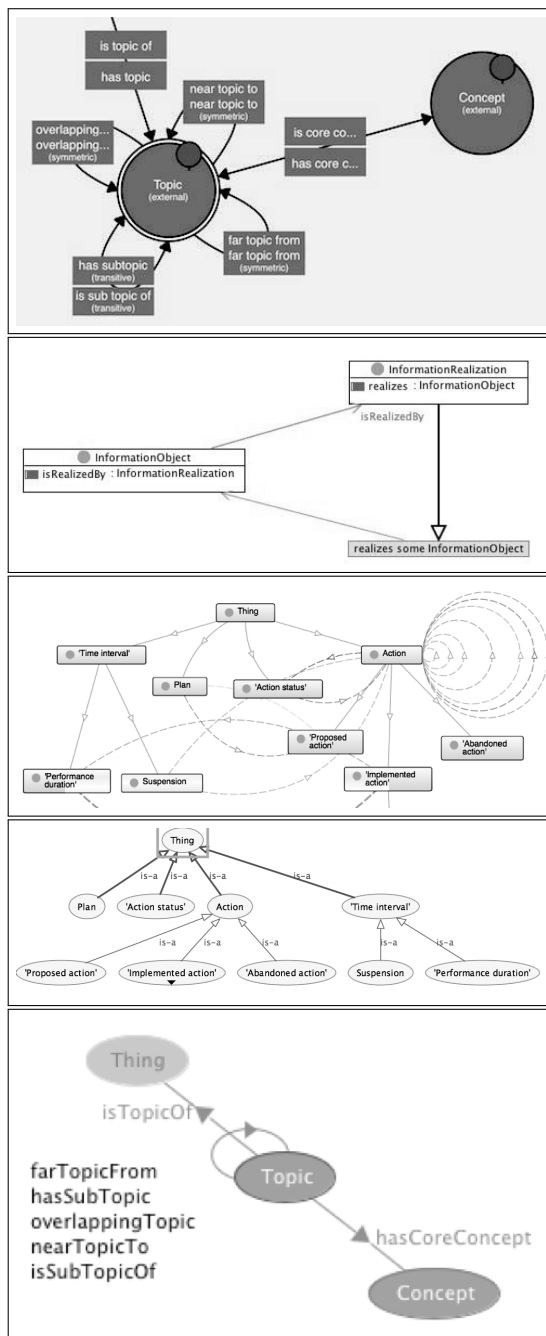


Figure 4.4: Ontology visualisation formats, from the top: VOWL, TopBraid Composer Graph view, OntoGraf, OWLviz, Unknown name (used in the *Topic* ODP in the community ODP Portal).

Table 4.13: Results of pairwise comparison of VOWL notation versus other common OWL visualisation notations.

Alternative	Prefers VOWL	Prefers Alternative
TBC Graph View	11	9
OntoGraf	18	2
OWLViz	12	8
<i>Topic</i> ODP Style	13	7

survey questionnaire, unlike in this figure, each pair of notations displayed the same ODP).

As the table indicates, VOWL syntax was consistently preferred among the respondents. The closest second was the Graph view provided in TopBraid Composer, while the other three representations are less preferred. This is perhaps not surprising given the amount of information that each representation conveys: both the TopBraid Graph View and the VOWL representation include much more detailed information about the ODP concepts than the other representations.

4.3.3 Ontology Engineering Workshop Observations

Several ontology engineering workshops were arranged within the VALCRI, SSynAHD, and OSTAG projects introduced in Section 3.2.4. Audio was recorded at each of these workshops, and researcher logs of participant behaviour were also maintained, as further discussed in Sections 3.2.5 and 3.2.6. Below, the findings on ODP quality from those workshops are presented, grouped by the artefact or action that they most directly measure—the ODP documentation, the ODP model, or the ODP when used by human ontology engineers. These findings confirm some of the quality indicators and effects from the second iteration ODP quality model, but they also support the addition of several new quality indicators for the model.

Documentation Indicators

The participants in the SSynAHD project asked on several occasions about which ODPs were most frequently used in practice, and whether usage of ODPs was somehow tracked, so that they might learn from such usage the best way to operate the ODPs they were studying. This observed behaviour is in line with and supports the effect proposed in the ODP quality model concerning ODP *Usage Example Count* (and corresponding *Usage Example Illustrations*). The participants also stated that such usage metrics would be helpful in evaluating the suitability of an ODP for use, suggesting that the indicator contributes to the *Appropriateness Recognisability* quality characteristic. They also indicated that in the absence of such usage tracking, it would be beneficial to beginner ontologists if ODPs in online portals

(e.g., <http://ontologydesignpatterns.org>) at least displayed some sort of stamp or other proof that a validation of its quality had been performed. A similar observation was made within the VALCRI project, where a key ontology developer was very reluctant to make use of that ODP portal, as it appeared relatively stagnant and as it contained no ODPs that had passed through a quality assurance process. Hence a new quality indicator, *Quality Approval Stamp* was added to the ODP quality model.

The SSyncAHD participants also observed and discussed usage challenges relating to the textual description fields accompanying ODPs. One participant remarked that negation phrases within the Solution Description field played havoc with the ODP search engine they used—that is, if an ODP description included a phrase such as “*This pattern does not support time-indexing*”, then that ODP would show up erroneously when the user searched for time-related ODPs. Such negations should thus not be used outside of the Consequences documentation field, and that field should not be searched by search engine tools unless explicitly called for by users.

Another issue relating to the textual description fields observed within SSyncAHD concerns the coverage of ontology entities. On more than one occasion the participants were perplexed by the presence of classes or properties in the accompanying OWL file or in the ODP structure illustration, that were not mentioned in the textual documentation or in the competency questions. This observation motivates the addition of a *Documentation Completeness* quality indicator; measuring the degree to which individual ontology entities are described in the ODP-wide documentation (i.e., not only in entity-specific annotations).

Lastly, within both the VALCRI and SSyncAHD projects, users found OWL 2 property chains to be simultaneously powerful for modelling purposes and potentially confusing when navigating the ontology. In SSyncAHD in particular, the participants suggested that if their ontology were to be reused in whole or part by other parties, then it is vital that the use of property chains is transparent and documented such that those users do not experience any unforeseen reasoning consequences. Thus, a recommendation to this effect was added to the ODP quality model.

Model Indicators

Experiences from both VALCRI and SSyncAHD strongly support the ODP quality model’s prediction that a high *Transitive Import Count* will have a negative effect on ODP usability. In VALCRI, ontology engineers and programmers who were planning to use an ontology expressed displeasure that the ontology included many imported concepts that they found irrelevant to the tasks they were performing. They repeatedly suggested that tooling user interfaces should be developed that would allow the masking or filtering of imports by the user, while keeping the perceived beneficial effects of those imports “under the covers”. Similarly, in the SSyncAHD projects, the participants repeatedly stated their displeasure with the presence of high-level

concepts in their ontology, resulting from `owl:imports`. In particular, they found the presence and use of the `Situation` class derived from the DUL¹⁵ ontology to be difficult to understand.

In all three projects, VALCRI, SSyncAHD, and OSTAG, the users expressed opinions on, and researchers made observations concerning, entity naming. In VALCRI developers emphasised the trade-off between using IRIs that are readable by humans (which simplifies debugging and communication) and using opaque or otherwise concept-agnostic IRIs based on UUIDs (which might improve subsequent modifiability). These effects lead to the inclusion of the indicator *Human-Readable Entity Names*.

In both SSyncAHD and OSTAG participants had trouble deciding on new entity names, particularly for object properties. Again, a trade-off was observed: entity names that are too long (e.g., `carHasRegisteredCompanyOwner`) in effect encode semantics into the naming itself, reducing future modifiability, whereas entity names that are too short (e.g., `has`) lack in usability. A reasonable “sweet spot” that worked well in practice was to use a noun-verb-noun or noun-adjective-noun combination, such as `personOwnsCar` or `carRegisteredtoPerson`. Naming practices that helped clarify the subsumption hierarchy were also beneficial (e.g., entities should have longer or more specific names than their super-entities, such as `TaxiCar` being a subclass of `Car`). Finally, in the VALCRI project, the participants emphasised the need to avoid homonyms in entity naming; this was particularly evident in the case of an annotation ontology, where the object property `hasBody` was intended to link instances of `Annotation` to instances of `AnnotationBody`. As the project concerned policing, a domain in which the term “body” often carries a rather more concrete meaning, this object property had to be renamed to avoid the risk of the two meanings being confused. In the ODP quality model, the above observations support the inclusion of an indicator and recommendations on *Entity Naming Structure*.

The ODP quality model suggests that high *Property Domain Restrictions Ratio* and *Property Range Restrictions Ratio* values support ODP usability. This effect was observed during the workshops in the OSTAG project, where the participants were at one point uncertain about the intended use of an object property that lacked such restrictions (even though they had themselves defined that same object property at a prior workshop). Similarly, the users were displeased that domain and range from superproperties were not displayed for subproperties in ontology IDE user interfaces without reasoning turned on, making the use of those subproperties less intuitive. These issues were resolved to some degree by adding domain and range restrictions to the affected properties. In so doing however, the participants came across another challenge, in that their intuition about the semantics of RDFS domain and range restrictions differed from the specification. Per the specification, multiple domain or range restrictions on a given property implies that the combined domain or range is the intersection of the classes

¹⁵<http://www.ontologydesignpatterns.org/ont/dul/DUL.owl>

in those restrictions. As the UIs of all major ontology IDEs allow the insertion of multiple domain or range restrictions independently of one another, it would be quite natural for the developer to instead assume that the restrictions are indeed independent and that the combined domain or range should therefore be the union of classes from those restrictions. To avoid unexpected inferences, users should therefore avoid multiple domain or range restrictions on a given property in most cases, and a recommendation to this effect was added to the ODP quality model.

In-Use Indicators

One group of participants within the VALCRI project emphasised that the ontologies under development in that project needed to display consistent semantic distances for certain question answering algorithms to work as intended. Semantic distance is the inverse of semantic relatedness; these measures indicate similarity (or lack thereof) of concepts within some information system. For instance, we might say that the concept *Car* has a higher semantic relatedness to *Bus* than to *Space rocket*. These measures are often used in conjunction with lexical methods in information extraction and natural language processing tasks [30]. Typically, such software would make use of manually curated natural language resources known to be well-structured with regard to semantic distance (e.g., WordNet [119]). These measures are thus highly dependent on human interpretation. The need for consistency with regard to distance/relatedness observed in the VALCRI project lead to the inclusion of a new quality indicator, *Semantic Distance Consistency*, contributing to *Functional Suitability*.

4.4 Summary: Resulting Quality Model

The ODP Quality Model resulting from the work presented in Sections 4.1 through 4.3 consists of three major components, presented below: an ODP Quality Metamodel (Section 4.4.1), a hierarchy of ODP Quality Characteristics (Section 4.4.2), and a listing of ODP Quality Indicators and effects (Section 4.4.3). Certain indicators contribute negatively to some quality characteristics but positively to others—a summary of some trade-offs of this kind is included in Section 4.4.4. Finally, Section 4.4.5 closes the chapter with a brief discussion of the qualities that were studied less thoroughly in this dissertation, and the importance of future work.

4.4.1 Quality Metamodel

The ODP quality metamodel is displayed in Figure 4.5. The key concepts it encompasses are:

- *Usage context*: Represents the context in which an ODP is used to construct an ontology. Component concepts of this context are the skills

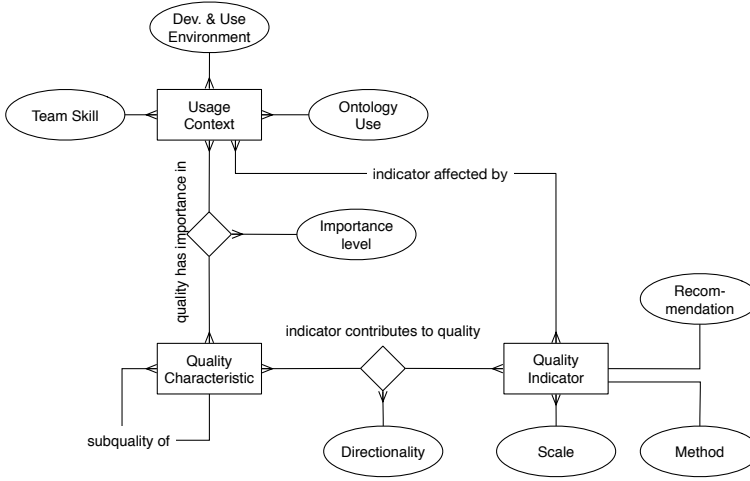


Figure 4.5: Quality Metamodel.

of the ontology engineer(s), the intended use for the resulting ontology, and the social or business environment in which the development and subsequent use of the ontology will take place (the latter including aspects such as development method, team distribution, customer relationship, etc.).

- *Quality characteristic*: Denotes a particular aspect of ODP quality that may be of greater or lesser importance in a particular usage context. Quality characteristics can 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. They are however affected (positively or negatively) by indicators that are measurable using some metric. The term *quality characteristic* is borrowed from and used similarly to the definition given in the ISO 25010 standard [94] on software quality.
- *Quality indicator*: Individually measurable properties of an ODP that contribute to increasing or decreasing some quality characteristic(s) (similarly to the concept *quality property* in ISO 25010). Indicators have scales of measurement, in accordance with the definitions established by Stevens [152] on nominal, ordinal, interval, and ratio scales. They also have recommendations that give guidance regarding relevant values for indicators to assume, to the extent that this can be given, and method definitions for how indicator measurements can be sampled, to the extent that it is not obvious from the indicator name itself. These different aspects of an indicator can be affected by, and vary depending on, usage context.

Table 4.14: ODP Quality Characteristics (for indicator names, see Section 4.4.3).

Quality characteristic	Sub-Characteristic	Indicators
Functional Suitability		IUI5
Usability	Functional Completeness	IUI4 DI4, MI1, MI5, MI8, MI11, MI18, MI20-22, MI24, IUI6 DI1, DI3, DI6, DI7, DI9, IUI3 DI3, DI5-8, MI16, MI17, MI19, IUI1
	Functional Appropriateness	
	Consistency	
	Accuracy	
Maintainability	Appropriateness recognisability	DI2
	Learnability	
	Operability	
	User error protection	
Compatibility	User interface aesthetics	MI1
	Accessibility	
	Modularity	
	Analysability	
Resulting performance efficiency	Modifiability	MI4, MI19 MI8, MI10, IUI2 MI10
	Testability	
	Stability	
	Interoperability	
Resulting performance efficiency	Reusability	MI1, MI22 MI16, MI17, MI24
	Co-existence	
	Interoperability	
	Interoperability	
Resulting performance efficiency		MI2, MI3, MI6, MI7, MI9, MI12-18, MI21-24

4.4.2 Quality Characteristics

The ODP Quality Model quality characteristics are presented in Table 4.14, and discussed in detail below. Table 4.14 also includes references to the quality indicators that have effects on each quality characteristic.

While most of the developed quality characteristics are associated with one or more quality indicators, several are not. This does not imply that those characteristics are unimportant, but rather that more work is required to find suitable indicators to measure them.

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).

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 learned by users that are 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 to 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-based ontology module (or the ODP itself) 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 that require modification.
- *Modifiability*: Degree to which an ODP-based ontology module (or the ODP itself) 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 expectation of changes to the ODP - high stability denotes that a low degree of change is expected, and vice versa.

Compatibility

Degree to which an ODP can be successfully 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 ontology, or in building other assets.
- *Co-existence*: Degree to which an ODP can coexist with other ODPs as independent modules in an ontology, without detrimental impact.
- *Interoperability*: Degree to which an ODP shares definitions of concepts that occur in other published ODPs.

Resulting performance efficiency

Reasoner or system performance efficiency over ontologies created using the ODP. Can be discussed in terms of the average time taken, or of the system resources required, to materialise inferences of ontologies using this ODP.

4.4.3 Quality Indicators and Effects

Tables 4.15 and 4.16 list the set of quality indicators and corresponding effects on quality characteristics in the ODP quality model. The former table holds those indicator effects that have been observed within this PhD project. In this table, the third column references the section of this dissertation that presents said observation. The latter table lists those indicator effects that have been hypothesised based on prior work or analysis of OWL features, but which have not been observed within this dissertation. In that table, the third column references the prior work that lead to the inclusion of the hypothesised effect, or the section in this dissertation where the indicator is discussed and motivated. The indicators are labelled and grouped into three categories:

- *Documentation indicators* (DI) concern the textual and graphical description of an ODP, typically provided in the form of an accompanying web page or other document. Such indicators contribute mostly to usability-related ODP qualities, such as appropriateness recognisability, learnability, operability, etc.

Table 4.15: Observed indicator effects.

Nr	Indicator	Affected characteristic	Section(s)
DI1	Accompanying Text Description	Appropriateness recog.	4.2.1, 4.3.1
DI2	Common Pitfalls Description	User error protection	4.2.1
DI3	Competency Question Count	Appropriateness recog.	4.2.1, 4.3.1, 4.3.2
DI3	Competency Question Count	Learnability	4.1.3
DI4	Documentation Completeness	Usability	4.3.3
DI5	Documentation Minimalism	Learnability	4.2.2, 4.3.2
DI6	Structure Illustration	Appropriateness recog.	4.2.1, 4.3.1
DI6	Structure Illustration	Learnability	4.3.1
DI7	Usage Example Count	Appropriateness recog.	4.3.1, 4.3.3
DI7	Usage Example Count	Learnability	4.3.3
DI8	Usage Example Illustrations	Learnability	4.2.2, 4.3.3
DI9	Quality Approval Stamp	Appropriateness recog.	4.3.3
MI5	Class/Property Ratio	Usability	4.2.2
MI8	Entity Naming Structure	Modifiability	4.3.3
MI8	Entity Naming Structure	Usability	4.3.3
MI10	Human-Readable Entity Names	Modifiability	4.3.3
MI10	Human-Readable Entity Names	Testability	4.3.3
MI16	Property Domain Restrictions Ratio	Learnability	4.2.2, 4.3.3
MI17	Property Range Restrictions Ratio	Learnability	4.2.2, 4.3.3
MI18	Property Restriction Count	Usability	4.2.2
MI19	Size	Learnability	4.2.1
MI24	Transitive Import Count	Reusability	4.2.1
MI24	Transitive Import Count	Usability	4.2.1, 4.3.3
IUI3	Name Appropriateness	Appropriateness recog.	4.2.1, 4.3.2
IUI5	Semantic Distance Consistency	Functional Suitability	4.3.3
IUI6	User-reported Abstraction Level	Usability	4.2.2

- *Model indicators* (MI) measure different aspects of the reusable OWL building block that makes up the core of an ODP, or its underlying RDF/RDFS model. These indicators contribute to qualities that are important in terms of how an ODP or an ODP-based ontology is used together with other ODPs, ontologies, or in information systems, including maintainability, compatibility, and reasoning performance.
- *In-use indicators* (IUI) measure properties of ODPs that cannot be captured using quantitative methods over the ODP itself, but rather require the use or evaluation of that ODP by a human ontology engineer. These indicators contribute primarily to those qualities that are contextually bounded, such as maintainability, usability, and functional suitability.

The full list of indicators is presented in Appendix A, together with suggested measurement methods and scales for each indicator, and in some cases, recommendations on suitable values.

4.4.4 Quality Trade-offs

In the presented model, there are indicators that contribute negatively to some quality characteristics while contributing positively to others. The presence of such tensions implies that ontology engineers who are developing or using ODPs may need to make trade-offs between qualities that they consider most important in their project. Organised by quality characteristic,

Table 4.16: Hypothesised indicator effects.

Nr	Indicator	Affected characteristic	Reference
MI1	Annotation Ratio	Maintainability	[130]
MI1	Annotation Ratio	Usability	[56, 57]
MI2	Average Class In-Degree	Resulting perf. efficiency	[98]
MI3	Average Class Out-Degree	Resulting perf. efficiency	[98]
MI4	Axiom/Class Ratio	Analysability	[56, 57]
MI6	Class Disjointness Ratio	Resulting perf. efficiency	[56, 57]
MI7	Cyclomatic Complexity	Resulting perf. efficiency	[98]
MI9	Existential Quantification Count	Resulting perf. efficiency	[98]
MI11	Minimalism	Usability	[120, 110]
MI12	OWL 2 EL Adherence	Resulting perf. efficiency	[91]
MI13	OWL 2 QL Adherence	Resulting perf. efficiency	[91]
MI14	OWL 2 RL Adherence	Resulting perf. efficiency	[91]
MI15	OWL Horst Adherence	Resulting perf. efficiency	[164]
MI16	Property Domain Restrictions Ratio	Resulting perf. efficiency	Section 4.2.3
MI16	Property Domain Restrictions Ratio	Reusability	Section 4.1.3
MI17	Property Range Restrictions Ratio	Resulting perf. efficiency	Section 4.2.3
MI17	Property Range Restrictions Ratio	Reusability	Section 4.1.3
MI18	Property Restriction Count	Resulting perf. efficiency	Section 4.2.3
MI19	Size	Analysability	[56, 57]
MI20	Subsumption Hierarchy Breadth	Usability	[56, 57]
MI21	Subsumption Hierarchy Depth	Resulting perf. efficiency	[107]
MI21	Subsumption Hierarchy Depth	Usability	[56, 57]
MI22	Tangledness	Compatibility	[56, 57]
MI22	Tangledness	Resulting perf. efficiency	[56, 57, 98]
MI22	Tangledness	Usability	[56, 57]
MI23	Terminological Cycle Count	Resulting perf. efficiency	[105]
MI24	Transitive Import Count	Resulting perf. efficiency	Section 4.1.3
IUI1	Functionality Questionnaire Time	Learnability	[61]
IUI2	Modification Task Time	Modifiability	[61]
IUI4	OntoClean Adherence	Accuracy	[68]

the trade-offs that have been found so far include *Resulting Performance Efficiency* versus *Functional Suitability*, *Resulting Performance Efficiency* versus *Learnability*, *Interoperability* versus *Usability*, and *Learnability* versus *Reusability*.

The trade-off between *Resulting Performance Efficiency* and *Functional Suitability* stems from the fact that employing the full logic expressivity of the OWL language is computationally expensive, hence the development of reasoning-friendly OWL subsets and the OWL 2 profiles (captured in indicators MI13–MI16). The most reasoning-efficient OWL model imaginable would be a null model that makes no assertions whatsoever. As a model comes closer and closer to representing some real-world domain (i.e., as the model’s functional completeness increases), the number of axioms (and most likely also the variance of OWL constructs used) in it also increases, which in turn decreases reasoning performance over the model. One concrete example of this is the use of existential quantification axioms (indicator MI10), which may well be required to accurately model the domain, but which are associated with poor reasoning performance [98].

When applying ODPs in scenarios where the resulting ontology will be used for reasoning and where the time or resource consumption characteristics of that reasoning matter, it is recommended that developers consider carefully whether the ODPs they use are compliant with their reasoning requirements. The simplest way of going about this is to ensure that the ODPs used comply with a suitable OWL 2 profile. However, it is also important to also ensure that in the process of specialising and applying those ODPs, no OWL constructs are introduced that are not supported by the chosen profile. Thankfully the two common ontology development environments Protégé and WebProtégé both include features that allow easy checking of the profile adherence of an ontology under development, simplifying this work. If a task-suitable ODP exists but does not adhere to a required reasoning profile, the author encourages the developer who notices this to reimplement and release that ODP as a new profile-compliant version.

The trade-off between *Reasoning Performance Efficiency* and *Learnability* is caused by the same underlying mechanism—several OWL constructs that increase learnability of the model are associated with poor reasoning characteristics. This includes the aforementioned use of domain and range restrictions, which typically help users understand the intended use of ODP properties. It also includes the use of class restriction axioms that can be similarly helpful in illustrating how classes and properties in an ODP are to be reused, but which in several cases decrease reasoning performance [98]. When constructing an ODP it may be beneficial to consider increasing the scope of the ODP metadata or documentation to better describe the ODP’s features and their intended uses, rather than relying on the above discussed performance-affecting constructs for this purpose (unless required to realise some actual functionality of the ODP).

The trade-off between *Interoperability* versus *Usability* relates to the

Transitive Import Count indicator (MI24). For an ODP to be as interoperable as possible, it should be aligned with and share foundational concepts with as many well-established ODPs as possible—which would be indicated by MI24 having a high value. However, for an ODP to be as easy to use as possible, it should not require the developer to understand a large number of phenomena that are outside of the scope of the problem that the ODP is intended to solve, that is, the MI24 value should be low.

Finally, the trade-off between *Learnability* versus *Reusability* concerns indicators MI17 (*Property Domain Restrictions Ratio*) and MI18 (*Property Range Restrictions Ratio*). As discussed above, the use of property and range restrictions improves *Learnability*, so an ODP that has higher values for these indicators would be easier to understand and learn than one that has lower ones. However, for an ODP to be as reusable as possible it should make as small an ontological commitment as possible while still fulfilling its design goals. Consequently, for reusability MI17 and MI18 should measure as low as possible, as the properties defined in the ODP should not make assumptions about the classes with which they will be used, but should rather be reusable outside of the exact ODP scope if need be. This is particularly important as the semantics of OWL property domain and range definitions imply that if a property has multiple asserted ranges or domains, the resulting inferred domain or range of that property is the intersection of the set of domain or range class expressions. In other words, it is not possible for a child ODP to extend a property domain or range from an imported parent ODP, only to constrain it further.

4.4.5 Notes on Unstudied Qualities

The ODP Quality Model contains several quality indicators that contribute to *Usability* and *Resulting Performance Efficiency*, but fewer that contribute to *Functional Suitability*, *Maintainability*, and *Compatibility*. This is a consequence of the difficulty of studying the latter quality characteristics. Understanding *Maintainability* requires studying a real ontology engineering project as it progresses beyond the initial development phases and into deployment and maintenance phases. By contrast, most research projects are formulated to focus primarily on the development phase, and consequently we do not have enough studies on deployment and maintenance yet. *Functional Suitability* is difficult to study rigorously as it is a very context-bounded quality characteristic, making it quite difficult to generalise any findings. Studying ODP *Compatibility* requires that a sufficient number of high quality ODPs from different sources or designed in different styles already exist, so that compatibility can then be gauged against them. This has not been the case until recently.

The lack of indicators for the above-mentioned quality characteristics is particularly unfortunate as these characteristics have several sub-qualities that are important in supporting ODP adoption outside of academia. *Main-*

tainability is perhaps the most obvious candidate for improvement—without understanding how ODP and XD use in ontology engineering affects the need for maintenance and support of developed ontologies, we will not see industry deploying these methods to any greater extent. Specifically, ODP *Testability* and *Analysability* are critical qualities to understand when developing ODP-based ontologies that will need to be maintained (possibly by a team that does not include the initial developers) in the future. *Testability* is also highly relevant in supporting the eXtreme Design methodology, which depends on tests to ensure that component modules of an ontology project do not break during integration and refactoring [131]. In addition to understanding these quality characteristics, tooling to aid in constructing and executing tests, or in analysing ODP-based ontologies, would also need to be developed.

Chapter 5

ODP Tool Support Improvement

This chapter attempts to answer the research question “*How can the features and functionality of ODP usage tools be improved to support inexperienced ontologists?*”. In order to provide solutions to simplify ontology engineering with ODPs, we first need to understand the types of challenges that users face. Each of the first three sections in the chapter discusses some particular challenge with which ontology engineers may need guidance, and then provides and evaluates a suggested solution to said challenge. Section 5.1 discusses challenges in finding the right ODP to use, and suggests improvements to ODP search engines that may simplify this. Section 5.2 concerns the choices users must make when specialising entities (particularly object properties) in ODPs for a particular use case. Section 5.3 discusses the usability challenges inherent in employing ODPs by way of such specialisation, and suggests an alternative method based on cloning ODP structures. The fourth and final section of this chapter, Section 5.4, discusses the need that users have for new ontology engineering tooling that both supports the use of ODPs and enables simple collaboration on ontology engineering tasks by geographically distributed teams. This section also introduces the *eXtreme Design for WebProtégé* (XDP) tooling that the author has developed which, in addition to fulfilling both of these needs, also integrates the solutions that are developed and discussed in Sections 5.1–5.3.

5.1 ODP Search

The following section discusses the difficulty of finding an appropriate ODP to reuse for a given modelling problem using currently available search engines, proposes a method of search that aims to improve the quality of ODP

search engine results, and evaluates an initial version of that method with promising results.

5.1.1 Motivation

As discussed in Section 2.4.2, the eXtreme Design method frames the task of finding an appropriate ODP for a particular problem as a matching problem where a local use case (the problem for which the ontology engineer needs guidance) is matched to a general use case (the intended functionality of the pattern) encoded in the appropriate ODP’s documentation. In order to perform this matching, the general use case needs be expressed in a way that enables matching to take place. Typically, this is done by encoding ODP functionality as Competency Questions, either described in the ODP documentation, or embedded as annotations to the ODP reusable building block itself (or both).

There are then two methods for finding appropriate ODPs for a particular modelling challenge—users can do matching by hand (by consulting an ODP repository and reading ODP documentations one by one), or they can employ the ODP search engine included in the XD Tools plugin for NeOn Toolkit (now discontinued) to suggest candidates, by way of rudimentary keyword-based searches over the ontologydesignpatterns.org community portal. As shown in Section 4.2.1, as soon as the list of available ODPs grows to a non-trivial number (such as in the aforementioned portal), users find the task of manually browsing through them to be challenging to perform correctly, particularly if the ODPs are not structured in a way that is consistent with their expectations.

In the case that an ODP search engine is employed, the signal-to-noise ratio of the results is often discouragingly low. In initial trials using the search engine included in the XD Tools plugin for NeOn Toolkit the author found that with a result list displaying 25 candidate ODPs, an actually correct and suitable ODP was included in less than a third of the cases. In order to guarantee that at least one suitable ODP was included, the search engine had to return more than half of the ODPs in the portal, essentially negating the point of using a search engine!

This difficulty in finding the correct ODP to reuse is further supported by answers to the Ontology Engineering Survey (introduced in Section 3.2.7)—nearly two thirds of respondents (19 of 31) stated that they found it *Difficult* or *Very difficult* to find suitable ODPs for reuse in modelling.

5.1.2 Proposed Solution

In order to improve recall when searching for suitable ODPs, the author suggests making use of two pieces of knowledge regarding patterns that the existing XD Tools ODP search engine does not consider: firstly, that the core intent of the patterns in the index is codified as competency questions, which are structurally similar to the types of queries that an end-user might pose,

and secondly, that patterns are general or abstract solutions to a common problem, and consequently, the specific query that a user inputs needs to be transformed into a more general form in order to match the indexed patterns level of abstraction.

The first piece of knowledge can be exploited by using string distance metrics to determine how similar an input query is to the competency questions associated with a pattern solution. The second piece of knowledge can be exploited by reusing existing language resources that represent hyponymic relations, such as WordNet [119]. By enriching the indexed patterns with synonyms for disambiguated classes and properties in the pattern, and by enriching the user query using hypernym terms of the query, the degree of overlap between a user query (worded to concern a specific modelling issue) against a pattern competency question (worded to concern a more general phenomenon) can be computed.

The author has developed a method of indexing and searching over a set of ODPs based on these ideas. The method, *CompositeSearch*, combines three different search methods, each of which generates a confidence value between 0 and 1, and these confidence values are added together with equal weight to generate the final confidence value which is used for candidate pattern ranking. The first search method employs a Semantic Vectors Search [171] of stemmed and normalised query terms over a Lucene ODP index including fields for all RDFS labels, local IRI fragments, and embedded ODP documentation annotations¹. The second search method uses a standard Lucene search with query terms that have been enriched with the WordNet hypernyms, and executes against an index including the same indexed information, but enriched with WordNet synonyms. Finally, the third search method ranks all candidate ODPs by the smallest Levenshtein edit distances [108] between the input query as a whole (assumed to be formulated as a competency question) and the competency questions embedded as annotations on the ODPs in question.

5.1.3 Evaluation

While the approach requires further work, early results are promising, as shown in Table 5.1.

The dataset used in testing was created by reusing the question sets provided by the *Question Answering over Linked Data*² (QALD) evaluation campaign. Each question was matched to one or more ODPs suitable for building an ontology supporting the question. This matching was performed independently by two senior ontology experts, and their respective answer sets were merged. The two experts reported very similar pattern selections in the cases where only a single pattern candidate existed in the

¹As expressed using the ODP community Content Pattern Annotation Schema, <http://www.ontologydesignpatterns.org/schemas/cpannotationschema.owl>

²<https://qald.sebastianwalter.org>

Table 5.1: Recall improvement for ODP search.

	XD-SVS	CompositeSearch
R10	6 %	22 %
R15	8 %	31 %
R20	9 %	37 %
R25	14 %	41 %

pattern repository that was compliant with a competency question (e.g., the Place³ or Information Realization⁴ patterns), but for such competency questions where multiple candidate patterns existed representing different modelling practices (e.g., the Agent Role⁵ or Participant Role⁶ patterns), their selections among these candidate patterns diverged. Consequently, the joint testing dataset was constructed via the union of the two experts' pattern selections (representing the possibility of multiple correct modelling choices), rather than their intersection⁷. Recall was defined as the ratio of such expert-provided ODP candidates that the automated system retrieves for a given input question.

As shown in the table, the average recall within the first 10, 15, 20 or 25 results is 3-4 times better using CompositeSearch than using the existing XD Tools Semantic Vectors Search (XD-SVS). It should be noted that while CompositeSearch also increases the precision of the results compared to XD-SVS by a similar degree, the resulting precision is still quite poor. The search engine user will consequently see a lot of spurious results using either of the approaches. This is understood to be a potential usability problem, and an area for further study.

A factor that is believed to limit the success of this method is the fact that resolving ODP concepts and properties to corresponding concepts and properties in natural language resources (in this case WordNet) is an error-prone process. This is largely due to the ambiguity of language and the fact that concepts in ODPs are generally described using only a single label per supported language. If pattern concepts were more thoroughly documented, for instance using more synonymous labels, class sense disambiguation would likely work better, and consequently ODP search would also work better. Additionally, WordNet does contain parts of questionable quality (both in terms of coverage and structure), the improvement of which could lead to

³<http://ontologydesignpatterns.org/wiki/Submissions:Place>

⁴http://ontologydesignpatterns.org/wiki/Submissions:Information_realization

⁵<http://ontologydesignpatterns.org/wiki/Submissions:AgentRole>

⁶<http://ontologydesignpatterns.org/wiki/Submissions:ParticipantRole>

⁷Testing datasets and code are available at <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-138724>.

increased quality of results for dependent methods such as the one presented here.

5.2 ODP Specialisation Strategies

As discussed in Chapter 2.4.2, a key step in the eXtreme Design method is the “*Reuse and integrate ODPs*” task, where the developer specialises a selected ODP (which represents a general reusable solution) to a specific modelling scenario. In the following section, the choices that the ontology engineer makes when performing this task are studied, and we explore the consequences that these choices may have for the ontology in question.

In order to understand how ODPs are in fact being specialised, a two-part method was employed. Initially, a set of ODP-using ontologies were studied in order to extract commonalities or strategies regarding how ODPs were specialised. Section 5.2.1 describes this process, and the ODP specialisation strategies discovered by using it. Subsequently, the usage of those specialisation strategies among ontologies were evaluated, along with the consequences of such use. These latter evaluations are described in Section 5.2.2.

5.2.1 Understanding ODP Specialisation Practices

Ontologies making use of ODPs first had to be located and downloaded. For this purpose, a method was employed that combined several different sources of ontologies. The initial set of ODP-using ontologies was retrieved using the Google Custom Search API⁸. This API was queried repeatedly, using all known ODP IRIs; the results were downloaded, filtered based on type, and only such that held both one or more instances of `owl:Ontology` and one or more references to known ODP namespaces were kept. Additionally, the LODStats⁹ list of RDF dataset vocabularies, the Linked Open Vocabularies¹⁰ dataset, and the known uses and instantiations of ODPs from OntologyDesignPatterns.org¹¹ were added to this set (the same criteria for filtering were employed). This resulted in 22 ODP-using OWL ontologies being found and downloaded. Additionally, a set of 19 such ontologies originating with the IKS¹² project were added to the set¹³.

From these 41 ontologies, 107 *specialisation mapping axioms* were extracted, that is, subsumption or equivalence axioms linking a class or property defined in the ontology to a class or property defined in a known ODP. These mapping axioms were analysed for recurring patterns based on the

⁸<https://developers.google.com/custom-search/>

⁹<http://stats.lod2.eu/>

¹⁰<http://lov.okfn.org/>

¹¹<http://ontologydesignpatterns.org/>

¹²<http://www.iks-project.eu/>

¹³The ontologies and scripts used are available via <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-138724>.

features of the ODP class or property being specialised, and based on the type of mapping properties used. Table 5.2 summarises the type of mappings used in the gathered data. As the table shows, simple mapping using `rdfs:subClassOf` and `rdfs:subPropertyOf` predicates against ODP named classes is the most common, together accounting for 85% of all specialisation axioms. In all but a handful of these uses, ODP classes and properties act as superclasses and superproperties to specialised classes and properties. Equivalency mappings against named ODP concepts are used less often; no uses of `owl:equivalentProperty` are observed at all, and `owl:equivalentClass` is only used in a few cases.

Table 5.2: ODP specialisation mapping axioms summary.

Mapping axiom type	Occurrences
<code>rdfs:subClassOf</code> against named ODP class	32
<code>rdfs:subPropertyOf</code> against named ODP property	59
<code>owl:equivalentClass</code> against named ODP class	3
<code>owl:equivalentProperty</code> against named ODP property	0
<code>rdfs:subClassOf</code> against restriction over ODP property	7
<code>owl:equivalentClass</code> against restriction over ODP property	2
<code>rdfs:subClassOf</code> against restriction over ODP class	4
<code>owl:equivalentClass</code> against restriction over ODP class	0

The use of existential or universal quantification restrictions involving ODP classes and properties is worth noting. In the studied set of ontologies, such restrictions are used to constrain the uses of ODP object properties, locally emulating domain or range axioms; for instance, a `WeatherForecast` is defined as being equivalent to the union of two restrictions, one using a project-defined vocabulary, and one on the `WeatherForecast` being an information realisation (i.e., `EquivalentClass (WeatherForecast objectSomeValuesFrom(informationRealization:realizes WeatherInformation))`).

Such a use of restrictions to constrain the local semantics of object properties can be seen as a form of specialisation of a more general model, or ODP, for a particular modelling case. This observation leads us to consider how this type of specialisation strategy differs from the more common strategy of specialising subproperties with defined domains and ranges, as supported by the existing XD Tools. In order to develop tool support for the use of property restrictions in ODP specialisation, the consequences of applying this type of modelling need to be studied, such that users can be informed of the potential effects of applying either the traditional (hereafter denoted “*property-oriented*”, due to the use of subproperties) strategy for ODP specialisation, or the alternative restriction-based strategy (hereafter denoted “*class-oriented*”, due to the use of property restrictions on subclasses).

Specialisation Strategies Described

The following section discusses and provides examples of the two initially discovered strategies for ODP specialisation. The possibilities and the consequences of combining the two strategies in a third, hybrid strategy, are also discussed. Two things are important to note. Firstly, an *ODP specialisation* is defined here as the set of specialisation mapping axioms that together specialise a single ODP for use in a target ontology. In most cases each ODP specialisation will consist of several specialisation mapping axioms, each specialising different classes or properties of the ODP. Often this set of axioms will be held in an ontology module that is imported into the target ontology, an *ODP specialisation module*. Secondly, in discussing the relative usage frequency of these strategies, here we only compare the specialisations that modify the semantics of object properties defined in the original ODP (simple class taxonomies without any object property specialisation have been filtered out), and we also include specialisations of ODP specialisations; as OWL imports are transitive, this type of layered structure is not uncommon. This gives a total of 20 ODP specialisations in the dataset that is being studied, the distributions of which over the specialisation strategies are summarised in Table 5.3.

Table 5.3: ODP specialisation strategy use.

Specialisation strategy	Occurrences
Property-oriented	9
Class-oriented	6
Hybrid	5

Property-oriented Strategy The property-oriented strategy is the most common type of ODP specialisation seen in the originally studied set of ODP specialisations, being used in 9 out of 20 cases. This may be due to the fact that OWL tools and tutorials tend to emphasize properties as basic language features, and the construction of property subsumption hierarchies that specialise those properties as fundamental modelling tasks.

The process by which an ODP is specialised in accordance with the property-oriented strategy is illustrated and exemplified in Figure 5.1 (the example is taken from an ontology developed in the IKS project). In the figure, the `ce:` namespace prefix indicates that classes or properties are defined within the CollectionEntity ODP¹⁴, whereas the `cc` namespace prefix indicates that classes or properties are defined within the ODP specialisation module ContentCollection. We see that the higher level class definitions defined in the ODP or OWL language itself (`ce:Collection`, `owl:Thing`)

¹⁴<http://ontologydesignpatterns.org/wiki/Submissions:CollectionEntity>

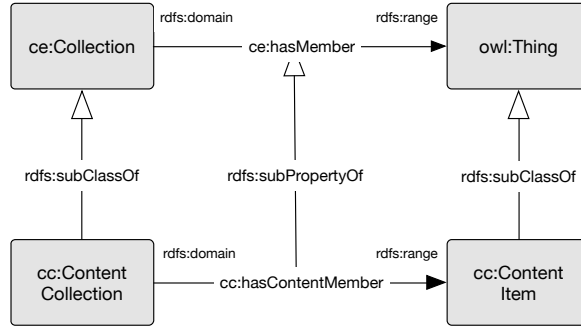


Figure 5.1: Property-oriented ODP specialisation strategy (rounded rectangles denote named classes).

are specialised for the modelling problem at hand using newly defined subclasses (`cc:ContentCollection`, `cc:ContentItem`), and that the usage of `ce:hasMember` is specialised to apply to these new class definitions via a new subproperty `cc:hasContentMember` with corresponding domain and range declarations. As shown by [100], the `rdfs:subPropertyOf` definition implies that domains and ranges of subproperties must be subsets of the domains and ranges of their superproperties.

It should be noted that this specialisation strategy does not necessarily need to be fully instantiated in all parts; there are several cases where only one subclass is created, and where either the domain or range of the created subproperty is therefore defined over a more general term. Indeed, that is the way in which the `CollectionEntity` ODP itself is structured in Figure 5.1, with `ce:hasMember` having a range of `owl:Thing`. The important thing about this strategy, and the key differentiator from the class-oriented strategy, is the definition of a subproperty.

Class-oriented Strategy The class-oriented strategy is a little less common in the studied set of ODP specialisations, being seen in 6 of 20 cases. The fundamental idea of this strategy is to avoid creating specialised subproperties, by instead reusing the object properties of the original ODP, and locally constraining the usage of those properties by way of property restrictions on specialised classes. As shown by Horridge et al. [91] the definitions `SubClassOf(A ObjectAllValuesFrom(someProperty B))` imposes a local range of B on the property `someProperty` for the class A. Using the same approach, we can represent a local domain of A over the property `someProperty` where the target of that property is B, by defining `EquivalentClass(A ObjectSomeValuesFrom(someProperty B))`.

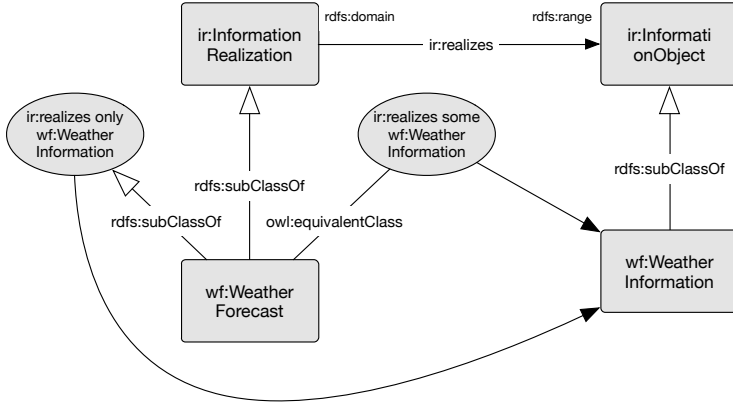


Figure 5.2: Class-oriented ODP specialisation strategy (rounded rectangles denote named classes, ovals denote property restrictions).

The concept is illustrated in Figure 5.2. The namespaces used in the figure are `ir:`, denoting the Information Realization ODP¹⁵, and `wf:`, denoting the Weather Forecast ODP specialisation module. In this example a `wf:WeatherForecast` class is defined in terms of how its member individuals are connected via the `ir:realizes` property to members of the `wf:WeatherInformation` class. In layman's terms, the `owl:equivalentClass` property restriction imposes the condition that any individual that is connected via `ir:realizes` to some other individual that is a `wf:WeatherInformation`, must itself be a `wf:WeatherForecast`; this restriction corresponds to the use of a `rdfs:domain` definition over a specialised property in the property-oriented strategy. Similarly, the `rdfs:subClassOf` property restriction imposes the condition that only individuals that are `wf:WeatherInformation` may be linked via `ir:realizes` from an individual that is a `wf:WeatherForecast`; this corresponds to a `rdfs:range` axiom in the property-oriented strategy.

Note that the above use of the term *corresponds to* does not imply that the described uses of property restriction axioms are logically equivalent to the use of domain or range axioms; merely that both modelling strategies allow for describing similar phenomena, via expressing constraints on which types of entities object properties can connect.

Hybrid Strategy In addition to the pure property-oriented or pure class-oriented strategies for ODP specialisation, the combination of the two in a hybrid strategy approach also occurs in the set of ODP specialisations: 5 of the 20 ODP specialisations use such a hybrid strategy. In these speciali-

¹⁵http://ontologydesignpatterns.org/wiki/Submissions:Information_realization

sations, subproperties with domain and range declarations are defined, and the classes involved are also defined using universal and/or existential quantification restrictions ranging over the newly created subproperties. The latter restriction axioms are possibly redundant if they are defined in this manner over properties that have themselves got domains and ranges. However, they could be helpful from a usability perspective in a large ontology of taxonomic nature, i.e. with a large class subsumption hierarchy, where one wants readers of said ontology to be able to easily grasp how classes are intended to be interconnected without having to study the property subsumption hierarchy.

5.2.2 Strategy Usages and Effects

As seen in Section 5.2.1, the number of published ODP specialisations in which object property specialisation takes place is limited; only 20 such cases were found. This is too small a dataset to base conclusions on. This section broadens the scope from just ODP specialisations to ontologies in general. If the same strategies are also observed in a larger set of ontologies, this strengthens the notion that ontology engineers need tool support for these strategies. Similarly, if the effects of applying these strategies can be observed on ontologies in general, ODP specialisation tooling needs to take this into account and guide the user accordingly when constructing ontologies using ODPs. The following section describes a study on the extent to which the strategies are employed in ontologies published on the web, discusses the effects of using the property-oriented or class-oriented strategies, and details a benchmark-based evaluation of the reasoning performance effects of such use.

Strategy Use

The ontologies studied were gathered in the same manner as described in Section 5.2.1. While we previously selected only those downloaded RDF files that held instances of `owl:Ontology` and that had references to known ODP namespaces, in this case the latter selection filter was dropped, and all downloaded graphs containing OWL ontologies were kept. This resulted in 423 ontologies for study.

Algorithm 1 was then executed over the ontology set. Per the algorithm, the number of specialised object properties that have domain or range definitions are compared against the number of properties that occur in a restriction emulating a local domain or range, and the ontology as a whole is classified based on the most commonly occurring type of structure. In the case that an overlap exists between these two sets, that is, that there are individual object properties that are specialised in both ways, the ontology is classified as employing the hybrid strategy. The results of this classification are summarised in Table 5.4 (again, simple class taxonomies have been filtered out).

Input: graph = An RDF graph containing an owl:Ontology

```

1 restrictionProperties = List();
2 hasDomainOrRange = List();
3 for subProperty defined in graph do
4   if hasDomain(subProperty) then
5     Append(hasDomainOrRange,subProperty);
6   if hasRange(subProperty) then
7     Append(hasDomainOrRange,subProperty);
8 for class defined in graph do
9   for superClass in getSuperClass(class) do
10    if hasPredicate(superClass, owl:allValuesFrom) then
11      restrictionProperty =
12        getObject(superClass,owl:onProperty);
13      Append(restrictionProperties,restrictionProperty);
14    for equivalentClass in getEquivalentClass(class) do
15      if hasPredicate(equivalentClass, owl:someValuesFrom)
16        then
17        restrictionProperty =
18          getObject(equivalentClass,owl:onProperty);
19        Append(restrictionProperties,restrictionProperty);
20 if Overlap(Set(hasDomainOrRange),Set(restrictionProperties)) > 0
21   then
22   Return(Hybrid strategy);
23 if Size(hasDomainOrRange) == 0 AND Size(restrictionProperties)
24   == 0 then
25   Return(No property specialisation occurring);
26 if Size(hasDomainOrRange) > Size(restrictionProperties) then
27   Return(Property-oriented strategy);
28 else
29   Return(Class-oriented strategy);

```

Algorithm 1: Detects ontology property specialisation strategy.

Table 5.4: Ontology specialisation strategy use

Property specialisation strategy	Occurrences
Property-oriented	193
Class-oriented	33
Hybrid	23
No property specialisation occurring	98

These results indicate that all three object property specialisation strategies discovered in Section 5.2.1 also occur to some extent in ontology engineering where ODPs are not used. The results also indicate that the property-oriented strategy is the most commonly used strategy by a large margin. Comparing this against the previously studied ODP specialisations (Table 5.3), we see that the class-oriented and hybrid strategies are used less frequently in ontologies (13 % vs 30 % of cases for the former, 9 % vs 25 % of cases for the latter). This suggests there may be a difference in how ontology engineering is performed when using ODPs as compared to in the general case. However, given the small size of the ODP specialisation sample set, and given that several of those ODP specialisations originated within the same project, further study would be required to validate this observation.

Strategy Effects

An advantage of the property-oriented strategy is that it creates new subproperties, which can themselves be dereferenced and annotated or typed as needed. For instance, such a specialised subproperty could be defined to be transitive or functional, without this definition affecting the parent property. Another advantage is that this type of modelling is accessible from a usability perspective; the simple tree view of the property subsumption hierarchy as used in many tools enables the ontology engineer or end-user to get an at-a-glance understanding of how the properties are organised and intended to be used. Yet another advantage is that, given that domains and ranges are defined, inferring the type of individuals connected via the property is a fast operation, when compared to the class-oriented strategy.

The main advantage of the class-oriented specialisation strategy is that, rather than creating subproperties, the original parent properties are reused. This allows RDF datasets that are expressed in accordance with an ontology using this strategy to be natively interoperable with other datasets using the same property, without the need for reasoning. This is particularly relevant in an ODP context, where the ODP and its properties are intended be reused extensively. Such interoperability can have many advantages, including in querying, where SPARQL triple patterns will often define only the predicate used and leave subject and object variables unbound. Furthermore, this

strategy allows for modelling of typing based on property links, much like *duck typing*¹⁶ in programming; such an approach can have advantages in situations where the ontology engineer does not control the RDF predicates used in data creation or extraction, but rather has to deal with what they are given.

There are also downsides to this strategy, most noticeably in terms of reasoning performance. As pointed out by Horridge et al. [91], universal quantification axioms, used in this strategy to emulate `rdfs:range` definitions, are disallowed in the computationally friendly OWL 2 EL profile. As illustrated by Urbani et al. [164], using property restrictions to infer typing requires multiple joins between large sets of candidate entities, greatly complicating reasoning, particularly when dealing with large datasets. The results of Kang et al. [98] (also discussed in Section 4.2.3) also indicate that there may be performance penalties associated with this type of reasoning. Their predictive model for reasoning performance includes eight ontology metrics categorised as impacting or strongly impacting reasoning performance; of these, three (the number of existential quantifications, average class out-degree, and subsumption tree impurity) are increased by employing the class-oriented strategy, as opposed to the property-oriented one. The strongest impact factor of any metric in their model is the number of existential quantification axioms, which are heavily used in this modelling strategy.

Reasoning Performance Evaluation In order to evaluate the reasoning performance effects of the class-oriented and property-oriented specialisation strategies, an experiment was set up using the well-known LUBM¹⁷ and BSBM benchmarks¹⁸. The hypothesis was that, due to the above-mentioned characteristics of the two strategies, the execution of reasoning tasks on datasets using ontologies that adhere to the property-oriented strategy would be faster than on the same datasets using ontologies adhering to the class-oriented strategy.

Each of the two benchmark suite ontologies were adapted to both the property-oriented and class-oriented strategies, via replacing domain and range axioms with universal and existential quantification restrictions or vice versa (in the case of BSBM, as an OWL ontology is not provided, said ontology first had to be created from scratch using the BSBM Dataset Specification). Datasets of non-trivial size (LUBM: 1053084 triples, BSBM: 334479 triples) were then generated using each benchmark suite’s data generator. In order to make the performance evaluation tasks that include

¹⁶An approach to programming in which the suitability of some object for some purpose, for instance as input to a method or function, is determined not by its formal typing, but by the methods or properties it exhibits; the name is thought to derive from the idiom “If it looks like a duck and quacks like a duck, it’s a duck”.

¹⁷<http://swat.cse.lehigh.edu/projects/lubm/>

¹⁸<http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/>

inferring typing axioms non-trivial, RDF typing axioms generated by the benchmark data generators were removed.

The datasets were then used together with the property- and class-oriented benchmark ontologies as input for two leading OWL reasoners, Pellet (version 2.3.1) and HermiT (version 1.3.8). The operations performed were consistency checking (ensuring that no contradictory axioms exist in the datasets and ontologies), and realization (finding the most specific class that a named individual belongs to). As the HermiT reasoner does not support performing realization from the command line, it was only used to perform consistency checking over the two datasets. The experiments were executed on a quad-core 2.6 GHz Intel Core i7 machine with an 8 GB Java heap size, running Mac OS X 10.9.3.

Table 5.5: Specialisation strategy reasoning performance effects.

Reasoning task	Benchmark	Reasoner	PO time	CO time
Consistency check	BSBM	Pellet	1.27 s	1.90 s
Consistency check	BSBM	HermiT	1.98 s	27.19 s
Consistency check	LUBM	Pellet	8.23 s	42.89 s
Consistency check	LUBM	HermiT	10.10 s	46 min
Realising individuals	BSBM	Pellet	2.39 s	9.48 s
Realising individuals	LUBM	Pellet	1.80 s	4+ hours

As illustrated in Table 5.5, the hypothesis holds for the generated datasets. In all of the reasoning tasks performed, the use of class-oriented ontologies resulted in slower execution than the use of property-oriented ontologies. In most cases the effects were severe; in one case execution of the reasoning task was halted when no results were reported after 4 hours of continuous execution. It should be noted that the inferred axioms of the reasoning tasks were equivalent, regardless of which strategy the ontology in question used.

5.3 Template-Based Instantiation

As discussed above, a key step in employing the XD method lies in the instantiation and adaptation of an ODP building block into an ontology module fit for solving the concrete modelling problem at hand. Per established practice this step typically consists of importing the building block into the target module using the `owl:imports` predicate and then, based on the existing classes and properties within that building block, specialising it by creating subclasses and (possibly, depending on the chosen specialisation strategy) subproperties that encapsulate domain semantics [131]. In the following, this way of instantiating ODPs will be referred to as *specialisation-based*.

The specialisation-based approach is generally well-known and understood by ODP researchers. However, in three of the projects this PhD project follows, the author has noted sets of users who have different preferences regarding the way that ODPs should be reused—for these people the use of the specialisation-based approach reduces the usability of the resulting ODP-based ontologies. An alternative approach to ODP instantiation is that of using the ODP building block as a template that is instantiated into the target ontology module by way of copying and renaming its constituent classes and properties [131]. This *template-based* ODP instantiation approach carries with it advantages and disadvantages, that may make it more or less palatable depending on the intended XD development context and ontology users.

The following subsections discuss the experiences from the three aforementioned projects (which demonstrated the need for supporting template-based approaches), presents a method by which template-based instantiation can be carried out, and provides an initial evaluation of this method.¹⁹

5.3.1 Motivation

In the three projects discussed, a recurring theme is the difficulty that less experienced ontologists (who typically exhibit a high degree of other technical skills, or skills in other types of conceptual modelling) have in understanding or agreeing with ODP-level entities that are imported into their models.

VALCRI Project

In VALCRI, ontologies are used for data integration purposes. A key requirement for those ontologies is that they need to be easily understandable by software developers. Another key requirement is that the developed system and its ontologies should be easy to modify for deployment in different contexts.

The initial versions of the VALCRI ontologies were developed mainly by partners with extensive experience of Ontology Engineering work. ODPs were instantiated into the target ontologies by import and specialisation. Challenges arose when these initial ontologies needed to be communicated to the less experienced ontology engineers in the project, such as professional software developers and researchers within other fields of computer science, who would then need to extend the ontologies. These developers all had extensive experience with data modelling, yet they found the developed models unintuitive and poorly designed. Their complaints primarily concerned two things:

1. Many foundational entities that are brought in via transitive imports from ODP building blocks do not immediately make sense in the tar-

¹⁹The work presented in this Section has previously been published in [81]

get domain. For instance, criminal involvement in crimes was modelled using the Participant Role ODP²⁰, which in turn depends on the Situation ODP²¹. As a consequence of this choice, the `Situation` class was brought into the target ontology as a high-level superclass. Developers expressed confusion and irritation with this design, which they argued added unnecessary complexity, as the requirements do not call for representing a general theory of situations.

2. ODP-level classes and properties, while they are helpful in solving the modelling problem at hand, are not named or labelled in a cognitively relevant way for the target domain. For instance, even after being instructed in the design of the set of reused ODPs, the developers still expressed dissatisfaction that entities in these ODPs had generic names such as *Agent* rather than the domain specific term *Nominal*²² that they were used to.

These complaints indicate that the initially developed ontologies do not fulfil the above discussed requirement of being easy to understand by software developers. Neither were the developers comfortable with modifying these ontologies for new uses or deployment contexts, seemingly due to not being confident that they properly understood the initial ontology design.

IMSK Project

As discussed in Section 4.2.1, at the IMSK workshop, three participants discussed the effects of `owl:imports` statements in ODPs, agreeing that the import feature and the large import closures it leads to create a tension between reuse and applicability. In particular, one participant criticised the use of imports, arguing that the base concepts included by imported patterns could be incompatible with one's own world view:

"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.

The participants in this workshop were not required to adhere to the XD method, but were free to use ODPs in whatever way they deemed most suitable to solve the provided modelling problem. Without explicit guidance,

²⁰<http://ontologydesignpatterns.org/wiki/Submissions:ParticipantRole>

²¹<http://ontologydesignpatterns.org/wiki/Submissions:Situation>

²²A UK Police term indicating a person who is reported as being involved in a crime in some role, including both suspected perpetrators and victims.

the intuitive way in which the developers instantiated ODPs was consistently to start by studying an ODP's design documentation, then to draw that ODP on a whiteboard, then modify that drawing to suit the specific modelling problem, before finally attempting to encode that illustration into a solution using Protégé. In spite of the participants having extensive experience in working with Semantic Web ontologies, the use of `owl:imports` to instantiate ODPs was never tried.

E-care@home Project

In reviewing an ODP-based sensor ontology that was developed in the E-care@home project, the author noticed that just like in the above two projects, certain classes and properties were included in the target ontology which, strictly speaking, were not necessary in the target context. When questioned about this design and about whether tooling or methods to support reuse without the need to import such classes would be useful, the developer that was responsible for the ontology in question responded:

"Definitely useful. I spent a considerable amount of time to find top-level classes that provide the required links to already designed ones. The lack of such tools is sensed. It can also decrease the rate of errors or inconsistencies in our design."

5.3.2 Proposed Solution

Benefits of Template-Based Instantiation

Before describing the approach, the author would like to emphasise that the idea underlying template-based ODP instantiation is not new; similar approaches are discussed in prior work [131, 47, 137, 93]. However, such template-based approaches have not achieved much adoption in the ODP community and the effects of their use have not been explored sufficiently.

The cases described above illustrate how certain types of ontology developers and users find the structures generated by the more common specialisation-based approach to be unintuitive, and that they consequently find ontologies constructed using this approach to be difficult to understand and to modify. For these users, a template-based approach in which ODPs are instantiated by copying and adapting ODP entities to the target domain semantics (and general ODP-level concepts are left out of the resulting model) would be advantageous. Additionally, a template-based ODP instantiation approach carries with it other advantages (and disadvantages) detailed below.

Template-based ODP instantiation does not require the target ontology to import semantics from namespaces outside of the project scope, so it reduces the risk of the target ontology breaking due to unforeseen changes outside of the project scope. This self-containedness also simplifies tool support implementation in tooling that does not support the addition of

`owl:imports` axioms (i.e., WebProtégé) or that needs to be able to work in an offline mode.

Furthermore, when performing template-based instantiation, the ontology engineer in question becomes more familiar with the resulting module than when they reuse a whole block of ontology functionality as-is, per the specialisation-based approach. Consider the analogy of program code examples taken from the web—rarely do developers reuse such code straight off, rather they more typically use such examples in parts, adapting them to the target context, and as part of this process, begin to understand and grow comfortable with the code. The same should hold for ODPs; by copying and adapting an ODP to the target domain, ontology engineers make it their own code, and this reduces the complexity of and barrier to entry of subsequent debugging or refactoring tasks.

Another advantage of the template-based approach is that the lack of ODP-level entities in the target ontology makes it easier for domain experts who are not knowledge engineers to validate that target ontology, as the number of entities with domain-irrelevant names decreases.

There are, of course, also disadvantages to this approach, or rather, advantages to the specialisation-based approach that the template-based approach does not share. In specialisation-based instantiation, the direct reuse of ODPs by `owl:imports` means that target ontologies that reuse the same ODPs (and RDF data that is expressed according to these ontologies) are immediately interoperable, not only in a conceptual sense, but also in that the same namespaces are used for shared concepts, which may simplify ontology integration and data sharing. This advantage could also be achieved in template-based instantiation by aligning local cloned ODPs to the original source ODPs, but it would then require OWL-level reasoning to be executed in order to achieve the same result, which may not be suitable in all cases. Also, in specialisation-based instantiation, higher-level classes and properties are defined only once, while in template-based instantiation pattern entities will be instantiated into the target ontology multiple times—which could complicate maintainability, and, particularly if done manually, could increase the risk of inconsistency.

Proposed Method of Instantiation

In order to implement tooling that supports template-based instantiation, a concrete list of tasks or steps to be taken must be established, along with the order in which these should be performed. The list of steps provided below has worked well in initial trials with some commonly used patterns. Due to the variance in structure of published ODPs, the below steps do not transfer all semantics from the source ODP (including its entire transitive import closure) to the target model in all cases—in some cases, additional modelling steps may be required. In the following “*copy*” implies cloning entities and associated class restrictions into a new namespace.

1. Copy ODP leaf classes into subclasses of `owl:Thing` in the target module. If two leaf classes in the source ODP have some shared parent classes below the `owl:Thing` level, also copy the least common subsumer into the target module as a shared parent to the copied leaves.
2. Copy those object or datatype properties that have as their domain or range such classes as were copied above into the target module. For object properties: narrow a potential unmatched half of the domain/range to the least common subsumer or (in the case that one does not exist) to the leaf class level.
3. Copy (and similarly to above, narrow if it is an object property) any properties involved in class restrictions on classes copied in the first step above and use these copied properties to create equivalent restrictions in the target module.
4. Merge the resulting structure with existing entities in the target module, using ontology matching techniques to find candidate matches.

As previously mentioned, the steps proposed above have worked well in initial testing, but there are some cases where they are insufficient and further manual work will be needed. These cases include but are not limited to:

- ODPs where leaf classes may need to be instantiated twice (e.g., the `Place` class in the `Place ODP`²³, or the `Object` class in `Time Indexed Part Of ODP`²⁴).
- When higher-level (non-leaf) classes from a parent ODP are reused and specialised in a child ODP the situation can arise that the child ODP concepts are sibling leaves to parent ODP concepts, and consequently when instantiating the child ODP, some leaf nodes can exist that are not intended to be instantiated.

5.3.3 Evaluation

As discussed above, ontologies built using template-based ODP instantiation are likely to be easier to understand and easier to modify for a certain group of inexperienced ontologists than those built using specialisation-based instantiation. In order to evaluate this proposition, a small study was run within the OSTAG project. The study included five participants, all academics within Computer Science or related topics. The participants had all used ontologies, but only two participants had actually constructed ontologies themselves, and these two were beginners to the task, this project being

²³<http://ontologydesignpatterns.org/wiki/Submissions:Place>

²⁴<http://ontologydesignpatterns.org/wiki/Submissions:TimeIndexedPartOf>

Table 5.6: Instantiation approaches ease-of-use comparison.

	Task 1	Task 2	Task 3
Template-based easiest	4	2	3
Equally easy/difficult	1	2	0
Specialisation-based easiest	0	0	0
Correct answer rate	83 %	81 %	

the first project where they had done such work. None of the participants had worked with XD or ODPs previously.

For evaluation two sets of ontology requirements were constructed, and for each of these, two ODP-based ontology variants were then built (one template-based, and one specialisation-based)²⁵. The specialisation-based variants were constructed using the traditional specialisation-based XD method, while the template-based variants were constructed strictly adhering to the template-based method proposed above. The workshop participants were given a tutorial on ODPs, XD, and the specific ODPs that the ontology variants had been constructed from. They were then given three tasks to perform individually:

1. Task 1: For requirement set A, out of seven competency questions provided, determine which ones can be answered by each of the two ontology variants.
2. Task 2: For requirement set B, out of nine competency questions provided, determine which ones can be answered by each of the two ontology variants.
3. Task 3: For the requirement set A, modify the two ontology variants by adding four object properties, specialising some of the more generic properties already in place.

After each task, the users were surveyed on which of the two ontology variants they found easiest to understand or to modify, or whether they found the two equally easy/difficult. The results of these surveys and the total percentage of correct answers to Tasks 1 and 2 given by the participants are provided in Table 5.6. Note that since not all users finished all tasks within the workshop time-frame, the answer frequency drops in Tasks 2 and 3.

Obviously, a study as limited as this is insufficient to conclusively evaluate the utility of the template-based approach, and the generalisability of the findings is limited. All the same, it's interesting to note that the proposed method can be followed in practice to generate ontologies that work, and that of the experiment participants, not a single one reported that the

²⁵Available at <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-138724>.

ontologies constructed using the traditional specialisation-based approach to ODP instantiation were the easiest or most helpful. These findings, albeit early, clearly demonstrate the need for additional study of instantiation method effects, along with the development of ODP tools supporting template-based instantiation.

5.4 Summary: eXtreme Design for Protégé

The following section introduces and discusses the features of eXtreme Design for WebProtégé (XDP), an extension to the well-established WebProtégé ontology engineering environment that supports the key ODP instantiation tasks in the eXtreme Design workflow, supports the new solutions developed and discussed in the previous sections of this chapter, and (due to its WebProtégé heritage) supports collaboration on ontology engineering by multiple simultaneous users over the internet. XDP is open source²⁶, works in any browser, and has passed initial usability evaluation with acceptable results.

5.4.1 Motivation

As discussed in Section 2.4.2, the XD method consists of a number of tasks, some of which are performed jointly by a development team, and some of which are performed by the individual developer(s). The latter include ODP *import* and *specialisation* (or in the case that template-based instantiation is employed, *cloning*), and *composition*. Using ODPs in an efficient manner requires tool support for performing these tasks.

Such tool support was developed within the NeOn project in the form of a set of plugins for the Eclipse-based NeOn Toolkit²⁷ ontology engineering development environment. The XD Plugin provided a number of components to this environment that simplified pattern browsing, selection, adaptation, and use. The browsing and selection components made use of the `ontologydesignpatterns.org` pattern repository, and allowed users to browse the patterns in this repository, or search over them using competency questions. The adaptation component contained a number of wizards that allowed for easier specialisation of generic patterns for use in specific projects. The assistant component provided warnings and suggestions to modellers based on known good or bad practices (i.e., patterns or anti-patterns). While the plugins are still available and to the author's best knowledge still work, the NeOn Toolkit lost development traction and stopped being updated soon after release—consequently, developers who want to use ODPs to develop ontologies that are compliant with the more recent OWL versions, or who want to develop in more widely used IDEs, have had no appropriate tool support available to them. Additionally, the

²⁶Available for download or study at <https://github.com/hammar/webprotege>

²⁷<http://neon-toolkit.org/>

Table 5.7: User preferences on communications methods for ontology engineering (65 responses).

Method	Percentage who prefer
Face to face conversation	58.5 %
Drawings	53.9 %
Joint development space	43.1 %
VMS commits	43.1 %
Email correspondence	41.5 %
Issue tracker / forum	35.4 %
Screen sharing	32.3 %
Video conference	27.7 %
Chat conversation	20.0 %
Phone conversation	15.4 %

XD plugin was built on the understanding of ODPs and ODP practices that was the state of the art at the time the NeOn project ran, i.e. 2006–2010. Since then, several advances in the field have been made, including those described in Sections 5.1–5.3, motivating further development of tooling to support these new findings.

XD is technically constructed as a fork based on the existing ontology engineering environment WebProtégé. This is motivated by the collaboration features that WebProtégé possesses, which the Ontology Engineering survey (introduced in Section 3.2.7) indicates would be appreciated by certain users. Specifically, Table 5.7 shows the answers to the question *“In communicating about modelling choices within an ontology engineering project, which communications methods would you prefer to use?”*. Users were allowed to select as many of the provided options as they wished. As the table indicates, a common preference among the respondents was to use a joint ontology development workspace, such as WebProtégé or a shared triple store, as a communications aid or method in ontology engineering. However, when asked which communications methods the respondents actually do use (Table 5.8), such a joint development workspace is not very frequently employed. While the gap between users who prefer such tooling (43 %) and the users who do use such tooling frequently (33 %) is not very large numerically, it should be noted that other communications mechanisms that are much less preferred (e.g., phone calls and chatting) are used to a much larger degree. If users are, as the latter would indicate, communicating about ontology engineering in ways that they *do not like*, then it is reasonable to assume that new tooling that enables these old methods to be replaced (such as a shared ontology engineering environment), would be appreciated by many users.

Table 5.8: Reported use of communications methods for ontology engineering (61-65 responses per row).

Method	Never	Rarely	Occ.	Often	Always
Email correspondence	1.5 %	1.5 %	13.9 %	49.2 %	33.9 %
Face to face conversation	1.6 %	6.4 %	23.8 %	44.4 %	23.8 %
Drawings	4.8 %	6.5 %	30.7 %	35.5 %	22.6 %
Chat conversation	12.9 %	14.5 %	17.7 %	43.6 %	11.3 %
Phone conversation	16.4 %	13.1 %	23.0 %	36.1 %	11.5 %
Video conference	26.2 %	8.2 %	19.7 %	31.2 %	14.8 %
VMS commits	21.3 %	13.1 %	21.3 %	29.5 %	14.8 %
Screen sharing	24.6 %	18.0 %	16.4 %	32.8 %	8.2 %
Joint development space	26.7 %	20.0 %	20.0 %	23.3 %	10.0 %
Issue tracker / forum	23.0 %	23.0 %	21.3 %	23.0 %	9.8 %

5.4.2 Developed Solution

WebProtégé [163] is a web application for ontology engineering. Its back-end is built using proven Protégé technologies, so it supports most of the recent W3C ontology standards. The user interface is constructed using the Google Web Toolkit (hereafter GWT) technology stack, which enables full-screen web applications that to a rather large degree look and act like desktop software. WebProtégé has fewer features than the full desktop Protégé ontology editor (there is no built-in visualisation, nor any ontology reasoner included by default), but it requires no local installation and it adds novel collaboration and discussion features (i.e., simultaneous online editing of the same ontology by two partners, embedded issue tracking on class-by-class level, etc.) enabling new usage scenarios that desktop Protégé does not support.

The use of GWT is of particular importance, as most existing OWL infrastructure code (desktop Protégé, the OWL API, etc.) is constructed in Java. GWT is, at its core, a Java to JavaScript transpiler—it takes existing Java code²⁸, and translates it into JavaScript that can be executed in a web browser. This enables developers to easily use a consistent Java-based development toolchain from the back-end through the front-end of their system. Additionally, it enables developers to write only one version of their web application, and GWT then generates JavaScript code that works in all major browsers (previously, browser-specific hacks were a common nuisance for web developers).

At the time of writing, WebProtégé does not have a plugin infrastructure. Consequently, the author’s work in adding XD support required forking the main WebProtégé code base, and subsequently, keeping the forked XDP edition up-to-date with mainline WebProtégé. In order to simplify such maintenance, the XDP integration into the existing WebProtégé code base

²⁸Not all Java features are supported in this process, but a very large subset are supported.

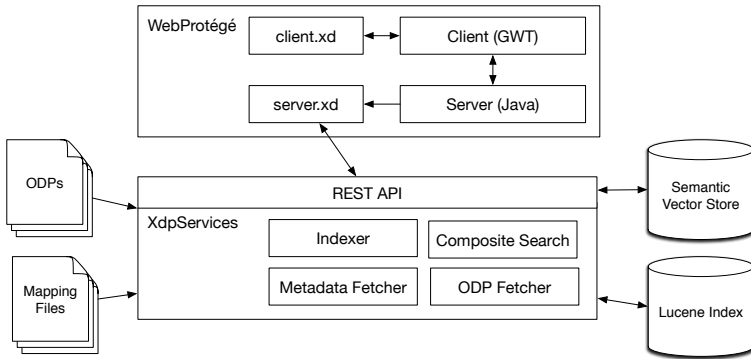


Figure 5.3: eXtreme Design for WebProtégé architecture diagram.

is designed to be as small and independent of other WebProtégé features (and therefore as maintainable) as possible. The XDP architecture has consequently been designed to consist of two loosely connected subsystems, illustrated in Figure 5.3:

- Integrated in a fork of the main WebProtégé code are several client UI components and the necessary server-side components needed to persist an ODP to a WebProtégé project. These components extend existing WebProtégé base GUI and persistence components. Likewise, they communicate with one another by using extensions to the WebProtégé standard GWT-RPC-based dispatch mechanisms.
- Supporting these components, a separately maintained and loosely coupled REST service enables clients to search for ODPs, to browse ODPs by category, and to fetch the documentation and OWL representation of a given ODP. This back-end service queries Lucene indexes built from a set of input ODPs and mapping metadata extracted from the community ODP portal²⁹, using the CompositeSearch method introduced in Section 5.1.

XDP’s user-facing components are housed in a WebProtégé UI tab titled “Design Patterns” (see Figure 5.4). This tab houses an ODP Selector component, and an ODP Details component. The former provides an interface where the user can browse for ODPs by category, or search over all ODPs using a query string. Browsing or search results are displayed in the same component; when browsing they are listed alphabetically, and when searching they are listed by search engine confidence score. Upon selecting an ODP from the result list, the illustration and documentation for that ODP is displayed in the ODP Details component.

²⁹<http://ontologydesignpatterns.org>

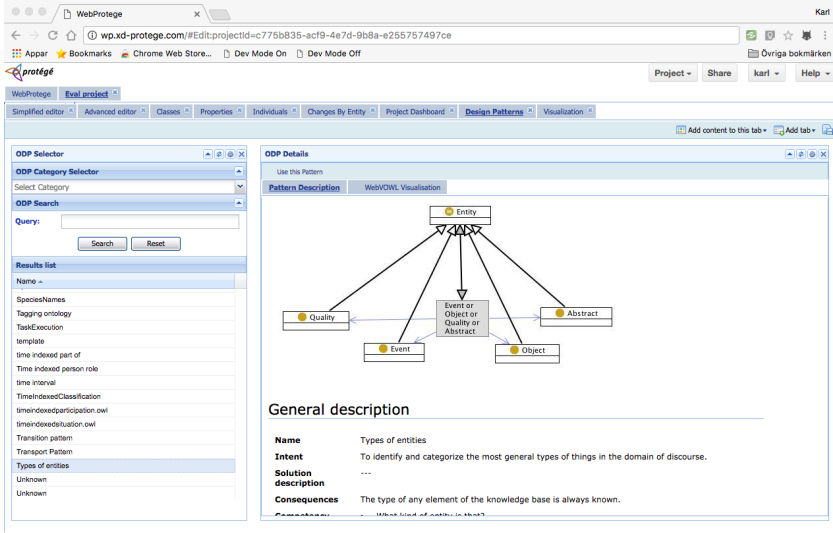


Figure 5.4: eXtreme Design for WebProtégé design patterns tab.

When the user has found an appropriate ODP they can run the ODP Wizard component, which guides them through instantiating the ODP into their target ontology. In the initial version 1.0 release, this wizard supported selecting a specialisation strategy to use (based on the work presented in Section 5.2), specialising ODP classes and/or properties by subsumption, constraining the semantics of property specialisations, aligning the resulting model with the existing ontology, and persisting the ODP specialisation into the existing ontology project. In the subsequent version 1.1 release support was also added for template-based ODP instantiation, using the method and heuristics presented in Section 5.3.

XDP version 1.1 also features integrated ontology visualisation using the VOWL notation [112] (see Figure 5.5). As shown in Section 4.3, users consistently prefer this notation over other available representations. Since the VOWL reference tooling called WebVOWL³⁰ is open source and built using web technologies such as JavaScript and JSON, integrating this technology with XDP was relatively straightforward.

5.4.3 Evaluation

XDP usability has been evaluated using the System Usability Scale (hereafter SUS). SUS, developed by Brooke [29], is a quick and easy tool for gathering users' opinions of the usability of some artefact, such as a function of some piece of software, a website, or some larger IT system. It consists of

³⁰<http://vowl.visualdataweb.org/webvowl.html>

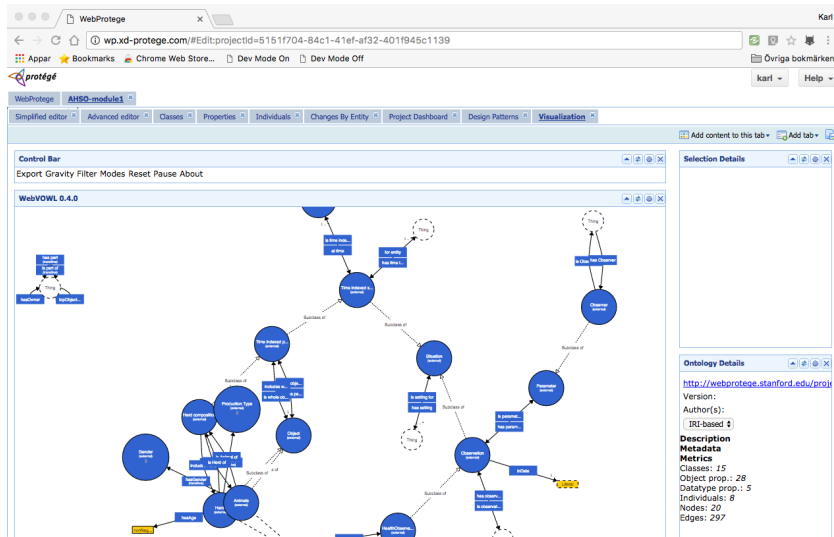


Figure 5.5: eXtreme Design for WebProtégé visualisation tab.

1. I think that I would like to use this system frequently
2. I found the system unnecessarily complex
3. I thought the system was easy to use
4. I think that I would need the support of a technical person to be able to use this system
5. I found the various functions in this system were well integrated
6. I thought there was too much inconsistency in this system
7. I would imagine that most people would learn to use this system very quickly
8. I found the system very cumbersome to use
9. I felt very confident using the system
10. I needed to learn a lot of things before I could get going with this system

Figure 5.6: System Usability Scale assessment questions (source: [29]).

ten Likert scale questions [109], each asking the respondent to grade on a five-step scale, ranging from “*Strongly disagree*” to “*Strongly agree*”, how much they agree with statements about the usability of the system (illustrated in Figure 5.6). The questions are designed in such a manner that they should ideally provoke some reaction in the respondent, such that answers would skew to either of the two extremes (“*Strongly agree*” or “*Strongly disagree*”), rather than recording ambivalent answers. Positive statements about the artefact under evaluation are alternated with negative ones in order to prevent the respondent from simply clicking through the same answer to all questions based on some overall assessment of the artefact in question.

Once all questions in a SUS evaluation are answered, the individual question responses are scored, summarised, and scaled, to produce a composite score between 0 and 100 representing the total usability of the artefact being evaluated. In the first step, a score contribution of 0–4 is assigned to each question’s answer. For questions with positive statements (1, 3, 5, 7, and 9 in Figure 5.6), the score contribution is the scale position minus 1, i.e., 0 indicates complete disagreement, and 4 indicates complete agreement. For questions with negative statements (2, 4, 6, 8, and 10 in the figure) the score contribution is 5 minus the scale position, i.e., 0 indicates complete agreement, and 4 indicates complete disagreement. The scores are summarised and multiplied by 2.5 to obtain the overall 0-100 score.

It is important to note that SUS is designed to evaluate the usability of systems in some given context, i.e., with similar (ideally the same) respondents applying the scale to multiple artefacts, or to multiple versions of the same artefact. Given the subjective nature of the questions, it is not guaranteed that respondents with different backgrounds, skills, or interests would necessarily reply in a consistent manner for some particular artefact. Consequently, Brooke [29] warns against generalising SUS results beyond the context and the population in which the SUS evaluation(s) took place.

That being said, Bangor et al [7] have collected a large number of SUS evaluations (2324) across a multitude of studies (206), and analysed these surveys quantitatively. Based on their analysis, and based on a subsequent study correlating SUS scores to overall evaluations of usability (based on adjective terms like “Excellent”, “OK”, “Poor”, etc.), they attempt to answer the question *What is an Acceptable SUS Score?* They find that a reasonable rule-of-thumb is that SUS scores below 50 are cause for significant concern (adjectives “Worst imaginable”, and “Poor” belong to this category), that scores ranging from 50-70 indicate an artefact of marginal usability (adjective “OK”), and that scores above 70 indicate an acceptable level of usability (adjectives “Good”, “Excellent”, and “Best imaginable”). It should be mentioned that Bangor et al consistently use the term “Product” rather than “System” in their surveys, which may skew the results somewhat, given the increased expectancy of usability that respondents are likely to have regarding what is perceived as a finished product, rather than a prototype system or software in development.

Table 5.9: SUS Evaluations of XDP 1.0.

Respondent	Score
R1-1	40
R1-2	45
R1-3	72.5
R1-4	77.5
R1-5	77.5
R1-6	52.5
R1-7	85
R1-8	65
R1-9	52.5
Mean	63.1

Table 5.10: SUS Evaluations of XDP 1.1.

Respondent	Score
R2-1	67.5
R2-2	30
R2-3	52.5
R2-4	75
R2-5	82.5
R2-6	75
R2-7	67.5
R2-8	47.5
R2-9	77.5
Mean	63.9

XDP SUS Evaluation

SUS evaluation of XDP was carried out with users within the VALCRI, SSyncAHD, and eCare@Home projects (version 1.0, Table 5.9), as well as at a tutorial held in conjunction with the International Semantic Web Conference (ISWC) 2016 (version 1.1, Table 5.10). These different contexts imply that the findings are not necessarily comparable with one another across versions, and the relative scarcity of responses (nine for each version) imply that the usability of the system has by no means been exhaustively tested. However, the primary goal of this evaluation was not exhaustive testing of XDP for the purpose of academically validating its usability. Rather, the author wanted, based on prior experiences of academic prototypes, to verify that an at least acceptable level of usability was reached for the first released version of XDP, and that this level of usability was more or less maintained as the XDP was updated with a variety of new features in version 1.1.

The mean score for version 1.0 was 63.1, with a median score of 65, and the mean score for version 1.1 was 63.9, with a median of 67.5. These sets of scores both fall within the upper half of the marginally acceptable range per Bangor et al [7], and indicate no immediate cause for concern that the new features added in version 1.1 reduce the usability of the tooling.

Additional User Feedback

In conjunction with the above SUS evaluations, respondents were also offered the opportunity to submit free-text comments with improvement suggestions regarding different aspects of the tooling. The comments (shown in Table 5.11) indicate that users seem quite satisfied with the presentation of ODPs, but that further training, UI features, or documentation may be needed to better support them in using the ODP search features and the instantiation wizard.

Table 5.11: XDP improvement suggestions from users.

XDP Feature	Comments
ODP Category Selector	<ul style="list-style-type: none"> • “<i>Maybe number of ODP in each category in parentheses?</i>” • “<i>It is not clear whether the browse/search features are intended to be used together - I initially thought it was a matter of filtering to a category and then running a search - but I don’t think this is the case. Some categories have no entries - making it difficult to see how the ‘browse’ is meant to work.</i>”
ODP Search	<ul style="list-style-type: none"> • “<i>More filtering, e.g. on OWL 2 profile. Perhaps filtering on top-level ontology applied (in case there is more than one).</i>” • “<i>Maybe if ODPs could be tagged, you could have predictive text in the search feature that suggest tags existing in the database. For example typing ‘memb’ could suggest ‘member’ or ‘membership’.</i>” • “<i>A bit of advice as to whether full CQ should be entered or simply 1-2 keywords.</i>”
ODP Details View	<ul style="list-style-type: none"> • “<i>I actually thought this was very good as it is. It was better than I would have expected, and I really appreciated the integration with OWL visualization.</i>” • “<i>I really liked that feature.</i>” • “<i>This was quite impressive - especially the integration of the visualization facilities.</i>”
ODP Wizard	<ul style="list-style-type: none"> • “<i>I think the wizard was great and worked very well for us. I am just not sure that I would have figured out directly what to do if I was on my own, using the wizard for the first time without help. So maybe it would be good to just have a little available help (on a pop up of icon), in particular to explain the use of ‘specialize’ and the use of ‘modify/delete’.</i>” • “<i>I think this may need to be explained in simpler terms for the non-expert.</i>” • “<i>This involved quite a lot of fairly complex choices - perhaps a bit more by way of guidance would be useful here? Also the delete/modify/etc functions can apply to each of a number of lines/options.</i>”

Chapter 6

ODP Methodology Development

This chapter presents the author's work to answer the research question "*How can ODP usage methodology be improved to support inexperienced ontologists?*". The author has, through involvement in three different ontology engineering projects, observed shortcomings in the eXtreme Design ODP-based ontology engineering methodology. These shortcomings include an absence of role descriptions and role allocation guidance, an absence of non-ODP reuse guidance, and an absence of guidance on how to adapt XD to non-optimal project contexts. The following chapter describes these observations in more detail and suggests improvements to the XD methodology to overcome them.

6.1 Project Roles

An important success factor in any project, but particularly in technically advanced projects such as those where ontology engineering is involved, is the matching of the right project member to the right tasks, such that those tasks are performed efficiently and correctly. This is particularly important if inexperienced developers are involved, who are likely to make a greater number of mistakes than more experienced users, and whose work thus requires a greater degree of quality assurance prior to release.

While role and task allocation might emerge as a project progresses, based on trial-and-error experiences within the project scope, such allocation is likely to be far less efficient and take far longer than if some decisions on roles and responsibilities had already been made at the project outset. While eXtreme Design suggests that each project begin with a project initiation and scoping meeting at which staffing and role issues should be discussed, the methodology does not give any advice as to what constitutes suitable

roles for an XD project, nor which types of tasks in the XD workflow ought to be staffed with which roles, that is, which responsibilities are associated with each role.

In the following sections, observations and experiences from three projects are presented, concerning the challenges of differing proficiency levels among developers, emergent roles, and user acceptance of formally assigned roles. Based on these reports, a suggested set of ontology engineering roles, role responsibilities, and role assignment principles are then presented.

6.1.1 Observation: Role and Task Challenges

VALCRI

Within the VALCRI project, several universities and industry partners came together to build a software system in which ontologies and other technologies were used to integrate and visualise data in the criminal analysis and investigations domain. As the project progressed it became apparent that the competencies and interests of the participants working on ontology engineering tasks differed widely, and that consequently, the work processes and task allocation had to be adapted to account for this. There were at least four different types of participants involved, roughly corresponding to four partners in the project:

- *Ontology Researchers*: Academics with deep knowledge of ontology technologies and ontology engineering processes, but little or no initial knowledge of the domain.
- *Software Developers*: Programmers from industry partners tasked with implementing a system making use of ontologies. Typically had experience of conceptual modelling using technologies other than Semantic Web ontologies (e.g., ER or UML models), and by virtue of being involved in requirements elicitation for the software system as a whole, had some knowledge of the domain.
- *Provenance Researchers*: Academics studying how to capture and formally represent information provenance such as where and how information originated, and how users have interacted with or modified that information. Had deep knowledge of part of the domain and experience of conceptual modelling, but lacked ontology engineering knowledge.
- *Policing Domain Experts*: Industry partners who work with and train police analysts. Had deep knowledge of several parts of the domain, including aspects such as criminal profiling, access and confidentiality requirements on data, chain of custody issues, etc. Also had experience of conceptual modelling, but lacked ontology engineering knowledge.

While the ontology engineering work package was lead by the ontology engineers, the initial plan was that each of the groups would work semi-independently and contribute modules or sub-ontologies based on their areas of expertise. In order to bootstrap the three groups of non-ontologists to become proficient in OWL ontology engineering, a set of training materials was produced and distributed by the ontology researchers, consisting both of self-study portions and hands-on sessions with a trainer. An initial work process based on eXtreme Design was developed and communicated. The team was geographically distributed and communication was maintained via weekly telephone calls.

It became apparent rather quickly that the initially envisaged work processes were not sufficient to produce results of high enough quality for use in the software system under development. Three observed challenges in particular concern the roles and tasks of the project:

1. Training software developers or domain experts in ontology engineering tasks to a level at which they can work on real world modelling challenges is difficult and time-consuming, even when the trainees have prior experience of conceptual modelling. In the VALCRI project, even after training, the participants needed extensive hand-holding from the ontology researchers.
2. Quality assurance and release engineering tasks for ontologies are difficult and require a particular set of skills that not all ontology developers have, or can easily develop. Reconciling incompatible design choices in a way that both fulfils requirements and pleases all involved parties may require the kind of understanding of both the domain and problem at hand, and the tradeoffs and consequences of design choices, that only comes from having participated in a couple of projects and learned through experience. In the VALCRI project the initial plan was that the ontology developers, after training, would be able to integrate partial modules. This never worked in practice.
3. Domain experts cannot always provide useful requirements on their own; input from the software developers who build the system is needed to capture the full set of constraints restricting the ontology. In the VALCRI project, we found that the participation of software developers in designing requirements was critical in concretising the competency questions that would need to be applied to the ontology, and also in prioritising which requirements to treat first, depending on the system development schedule.

As a consequence of these and other challenges, the ontology engineering work stalled for quite some time. Acceptable results were eventually developed, but only after the academic ontology engineers had taken over responsibility for developing and formalising the ontologies, the other three groups had reverted to providing documentation and other materials describing

their respective domain concepts (e.g., instructions, definitions, diagrams, etc.), and the system requirements had been significantly scaled back. This solution, in addition to providing a less fully featured ontology, required initially unanticipated investments of time and effort, including additional communications overhead (some of which required staff to travel across Europe) and additional work to resolve ambiguities or other issues caused by the added communications complexity. Additionally, this solution depended on high availability of expert ontology engineers, the number of which were limited in this project. For these reasons it is unlikely that this solution would have scaled well to a more ambitious ontology engineering project.

SSyncAHD

The core development team of the SSyncAHD project consisted of three people, all with backgrounds in veterinary epidemiology, that is, the study of how disease spreads among animals (in this case primarily farm animals). None of them had prior experience of participating in ontology modelling projects. As the workshops with these three developers progressed it became obvious that while their domain expertise was comparable, the developers differed with regard to their understanding of ontology engineering and with regard to which parts of the project they were most interested in. This was evident in the complexity level of the questions posed, the results of individual modelling exercises that were performed, and the degree to which the individual developers participated in and drove the discussion, as illustrated in Table 6.1¹.

Table 6.1: Statements expressed per participating developer (each statement consists of one or more uninterrupted spoken sentences; trivial one-word utterances are excluded).

Speaker	Workshop 1	Workshop 2
Karl Hammar	131	571
Developer J	82	480
Developer K	76	382
Developer L	33	95

Studying the content of these statements, the three participants can be characterised as follows:

- *Developer J*: Displayed a good understanding of ontology engineering practice, especially considering lack of training. Asked practical questions about how to implement some feature correctly or how one might best model some specific concept when whiteboard prototyping.

¹Participants other than the author have been anonymised as required by the conditions of the author's participation.

Lead (i.e., held the pen and lead the discussion) most such prototyping work. Also discussed workflow challenges such as software support, tooling for reusing existing ontologies, IRI minting, etc.

- *Developer K*: Displayed a lower level of understanding of ontology engineering practice but a more extensive understanding of the domain in question. Discussed how to correctly understand or interpret concepts from the domain when whiteboard prototyping. Occasionally modified or corrected solutions by J.
- *Developer L*: Displayed a lower level of understanding of ontology engineering practice. Less active throughout the sessions, but seemingly not due to any social reasons (interactions during the workshop breaks were comparable to J and K). Asked about methodology and workflow, and about the possible intellectual property consequences of ontology reuse.

These characterisations are of course simplifications, but they give a passable overview of how the participants interacted. J was quite clearly the most proficient modeller, K was more knowledgeable about the domain and helped J capture it correctly, and L was a bit quieter and seemed to take in and analyse a lot of knowledge before contributing to the discussion (though when they did, those comments were of some consequence).

As the second workshop (the longest and most ambitious in terms of development goals) progressed, the author noted that the three developers spent a lot of time discussing the merits and drawbacks of different ways of modelling relatively simple concepts. While there was little feature or scope creep (thanks to the use of competency questions and the eXtreme Design methodology), the participants were nonetheless unable to finish modules on time and progress to the subsequent requirements stories. This was likely the result of three factors: firstly, with four people present in the room, discussions had a tendency to drag out more than needed, to capture every perspective on the problem from every participant present. It is likely that pair-based development as suggested by XD would be more efficient. Secondly, the situation lacked a designated project manager or project leader, someone who could have put their foot down and decided that a particular module was good enough to commit to the ontology under development. Thirdly, due to the lack of training, too much time was spent on discussing basic ontology modelling issues, which delayed work.

When the author suggested that the developers consider establishing the emergent roles described above more formally, and associating certain XD tasks to the different roles, they immediately latched on to the idea. They suggested that J take on a role which included quality assurance and release engineering, whereas K and L would study up on and subsequently work with more basic ontology engineering tasks. The developers also brought up the value of employing role structures in the case that the project become an open source collaboration (which was a potential future goal of

the project)—in such a scenario, they argued, they would need some way to differentiate between the core developer team and occasional external contributors.

OSTAG

There were three main developers in the OSTAG project, all of whom were academics in the ontology engineering and Semantic Web fields: one associate professor, one senior researcher (with a PhD), and one junior researcher (a PhD student). The junior researcher had limited ontology engineering experience, whereas the other two developers had worked with ontologies for some time.

The associate professor developed the software components that made use of the OSTAG ontology. The other two researchers worked jointly to build the ontology, based on documents describing the domain that were provided by industry partners (typically hardware specifications and requirements specifications documents). To this end, they would meet in an office or meeting room and do modelling, both on a whiteboard and using a shared computer. The parts of the ontology that were easy or obvious to formalise into OWL were immediately entered into Desktop Protégé on the computer, while any non-trivial modelling challenges initiated a round of back and forth modelling on the whiteboard, comparing the pros and cons of different designs, before a consensus was reached and the selected solution input into the computer.

While there were no formal roles assigned in the development process, a working order emerged early on in the project whereby both ontology developers took turns at the whiteboard, and the junior developer was responsible for managing input of the resulting design into the computer. While the senior researcher initially directed the work, and had the final say about the designs, the junior researcher's competence rapidly increased to the point that they both made roughly equal contributions to the work, improving on and correcting each other's work. Work progressed at a reasonable pace, further aided by the fact that the provided specifications and requirements documents were rather unambiguous and consequently relatively easy to formalise into an ontology.

Issues arose, however, when the first iteration ontology was released to the associate professor, who needed to implement a rather complex piece of code-generating software that made use of the developed ontology. While the developed ontology fulfilled the formal requirements that had been elicited from the documentation from industry, the associate professor found it difficult to use in practice, due to poor usability and lack of documentation. Furthermore, the first iteration ontology lacked a fine-grained representation of some concepts that the software components required. This necessitated the development of a second iteration ontology that fixed these issues. Had the associate professor been involved more frequently with the development

project, these issues would likely have been caught earlier, which would have saved development time.

6.1.2 Suggestion: XD Roles and Responsibilities

Given the challenges observed and discussed above, a set of recommendations on roles, the responsibilities of those roles, and role allocation principles would likely have proven beneficial to the discussed projects (and ideally also to other projects of similar nature).

While many existing ontology engineering methodologies (e.g., METHONTOLOGY [49], On-To-Knowledge [157], etc.) entirely ignore this facet of ontology engineering there are some that do not, notably DILIGENT [129, 128] (discussed in Section 2.3). Unfortunately, DILIGENT lacks some detail: it does not describe the suggested roles beyond their labels (*domain experts*, *users*, *knowledge engineers*, and *ontology engineers*), it does not discuss role allocation, and it does not discuss the responsibilities of the respective roles in the initial ontology construction phase of a project (the method emphasises subsequent analysis, revision, and update tasks instead). Furthermore, DILIGENT suggests that a *control board* should have the final say with regard to updates under development, when in fact, as discussed above, collective responsibility for release engineering work is not always appropriate and can instead slow down development. Due to these limitations, direct reuse of DILIGENT roles and role allocation is unsuitable. Instead, the author proposes the following role descriptions and role allocation principles, based on experiences from the projects discussed above:

- *Project Manager*: The project manager bears the overall responsibility for the success or failure of the project. In a business context, the project manager role is typically assigned based on business hierarchies outside of the XD project scope. In an open source project, the role can be assigned via other methods, including voting by project members. Responsibilities of the project manager include providing the required infrastructure for project members, establishing collaboration practices within the team and between the team and the *Requirements Source*, allocating roles and development pairs, and ensuring that project members adhere to the XD process.
- *Requirements Source*: At the root of any ontology engineering project lies the need to perform some particular task. The *Requirements Source* is the party that has this need. Depending on context, this might be a paying customer of the development team, it might be a different department or function within the same organisation as the development team, or it might even be the development team itself, when constructing internal support systems. Assigning this role is a matter of investigating who originally initiated the development

project and for what purpose. The responsibility of the requirements source includes provision of requirements stories per the XD process.

- *Domain Expert*: Typically and ideally, the *Requirements Source* would also be a domain expert, carrying with them all the knowledge required to formulate system and/or ontology requirements. However, there may be cases where additional analysis or guidance regarding the domain may be needed in order to formulate such requirements. In these cases, requirements elicitation would benefit from further domain expertise. *Domain Experts* might be technical or business consultants or subject-matter experts hired to facilitate development, or they might represent agencies concerned with regulatory compliance, etc. The responsibility of the domain expert is to work with the requirements source to ensure that the elicited requirements stories are consistent with the real-world domain requirements and any regulatory compliance demands.
- *Software Developer*: Ontologies are rarely used in isolation, rather they typically form part of or interact with some software system. Any person(s) developing such software systems should be assigned the *Software Developer* role. The responsibility of the software developer in an XD project is twofold. Firstly, to work with the *Requirements Source* to ensure that the ontology requirements take into account any requirements that may arise from software implementation, which might not be obvious to a non-developer. Secondly, to act as a liaison between the XD project and the software development team, and in particular to inform the latter of how to work with the ontology in practice (implications concerning libraries and tooling used, testing practices, etc.)
- *Ontology Developer*: This role encompasses any project member who is assigned to develop ontology modules within an XD project. Developers may be more or less experienced—in the latter case, they should receive tutoring and support by an *Ontology Expert* (possibly, by virtue of such training, attaining enough experience and confidence to fulfil that role themselves). When working in pairs, such training and competence transfer can be supported by pairing a more experienced developer with a less experienced one, or ideally, a less experienced developer with an *Ontology Expert*. Also, the more experienced and competent the developer, the less likely they are to be over-dependent on pattern support when modelling. Consequently, it is beneficial to let a more experienced *Ontology Developer* model such requirements stories where no appropriate ODPs can be found and manual development work is needed.
- *Ontology Expert*: This is a senior *Ontology Developer* who is very well-versed in ontology engineering tasks and technologies. The *Ontology*

Expert is distinguished from the *Ontology Developer* by the fact that the former does not require additional training in ontology engineering, and that they can perform all of the XD development tasks with confidence. In addition to normal development work, the responsibilities of this role include tutoring *Ontology Developers* and supporting the latter when they encounter difficulties in modelling. In the case that the development team is geographically distributed, it is important that *Ontology Experts* are available to support all developers. Ideally this would be achieved by ensuring that there is at least one *Ontology Expert* at each development site. In the case that this is infeasible, teleconferencing solutions might be employed instead.

- *Ontology Release Engineer*: This is the person who, within an XD ontology engineering project, merges the partial results or modules from *Ontology Developers* into a coherent joint ontology (possibly refactoring either the module or target ontology), tests the resulting ontology, and releases it for use. Ideally and typically this role will be filled by an *Ontology Expert*, though with sufficient tool support and training, a less experienced *Ontology Developer* could also fill the role.

These roles may (and in fact in most cases are very likely to) overlap, such that 2-3 roles are filled by a single individual. However, the development team should take care not to leave any of the roles unallocated, as this may lead to development being delayed or otherwise performed in an inefficient manner.

6.2 Ontology Reuse

There are many modelling issues and domains for which ontologies have already been constructed at a substantial expenditure of time and effort, using ontology standards such as OWL that facilitate their reuse. There are good reasons to try to reuse such ontologies if possible, including possibly significant savings in development time, and the reuse of established good practices embedded in them. Ontologies can be reused in several different ways, e.g.: by direct reuse of the entire ontology using `owl:imports`, by copying the required structures from the reused ontology into the target ontology namespace, by aligning the target ontology entities to the reused ontology using subsumption or equivalence mappings, by breaking apart the original ontology into reusable modules, etc.

Of these reuse approaches, the XD methodology only supports the last. While this approach is easy to reconcile with the overall XD methodology and the use of ODPs, it also has drawbacks that may make it unsuitable in some cases. Firstly, the additional work required to extract and formulate an ODP from the source ontology (which includes generalising the solution, documenting it, and ideally publishing it publicly) is arguably not justifiable

if the solution is not going to be reused more than once. Secondly, this approach results in an ODP that uses a different namespace from the original ontology it is based on, which negates several of the advantages of reuse (i.e., compatibility with other ontologies and/or systems).

Ontologists who, for these or other reasons, wish to reuse whole ontologies without first breaking them apart into ODPs, are left without sufficient guidance from the XD methodology. This lack of guidance has given rise to challenges and less-than-optimal results in several projects the author has been involved with. The following sections present and discuss these challenges, before proposing a set of questions to aid the ontologist in choosing among several suitable ontology reuse strategies.

6.2.1 Observation: Ontology Reuse Challenges

VALCRI

The VALCRI ontologies covered issues and content such as crimes, suspects or perpetrators of crimes, vehicles, observations, arrests, intelligence reports, objects (and object taxonomies) discussed in police reports, etc. While large parts of the work in these domains were greenfield, that is, there was little prior work to consider or adapt to, some sub-sections of the domain were relatively well explored. The sections for which ontologies already existed included annotations on documents or reports (Open Annotation Collaboration²), document provenance (W3C PROV³), and standards for constructing classification schemas or taxonomies (W3C SKOS⁴).

Table 6.2: Degree to which imported ontology entities were used (the Open Annotation Data Model is a RDF vocabulary and thus does not differentiate between object and datatype properties).

	PROV	SKOS	Open Annotation
Classes	4 / 51 (8 %)	2 / 4 (50 %)	3 / 29 (10 %)
Object properties	4 / 60 (7 %)	1 / 17 (6 %)	
Datatype properties	0 / 9 (0 %)	0 / 1 (0 %)	
RDF properties			1 / 21 (5 %)

Developers reused these ontologies by importing them into their target ontology using the `owl:imports` predicate. The imported ontology concepts were either aligned to pre-existing ontology concepts by subsumption, or they were directly used in terminological (e.g., class restrictions, domain/range restrictions, etc.) or assertional (e.g., individuals used in SKOS collections) axioms. A problem with this approach is that it brought along a large number of entities that were both unneeded and potentially

²<http://www.openannotation.org>

³<https://www.w3.org/TR/prov-overview/>

⁴<https://www.w3.org/2004/02/skos/>

confusing to the developers. Table 6.2 illustrates how few of the imported ontology entities were used in axioms in the target ontology—there is very clearly a large overhead of unused entities. For at least one of the resulting ontologies, developers other than the original author had difficulties understanding how it was in fact constructed, as they had trouble locating the VALCRI-specific additions in the class and property subsumption trees.

Another challenge the author observed in VALCRI concerns the difficulty of correctly aligning to entities defined in existing ontologies. These ontologies typically model non-trivial domains, and are likely to have evolved through a process of refinement within some project group until they were deemed complete enough to be released to the general public. In other words, they tend to be rather complex, either in terms of modelling design or in terms of the ontological assumptions that they encode. Furthermore, while they may be designed with reuse in mind, they are rarely designed with reuse *through extension* in mind. Their documentation typically assumes the ontology will be reused as-is, and focuses on examples of such reuse, rather than examples of extensions and modifications to suit neighbouring domains. In VALCRI the initial annotations ontology exhibited several flaws as a consequence of the responsible developer misinterpreting the intended usage of the OpenAnnotations ontology entities and aligning their own classes to the wrong Open Annotations parent classes. While the developer’s solution was internally consistent, the subtly incorrect alignment to Open Annotations would have broken compatibility with that ontology (had they not been discovered), essentially negating the benefits that motivated reuse in the first place, and possibly triggering unexpected bugs and issues in subsequent deployment. Had another approach to reuse been employed (e.g., copying or cloning the design but without maintaining alignment through subclassing) such “hidden” problems would have been avoided.

Yet another challenge observed in VALCRI concerns *when* in the XD process reuse questions should be decided on and reuse implemented. Certain decisions regarding reuse (whether of ontologies or non-ontological resources) that concern foundational concepts could be quite expensive to modify at a later time. In VALCRI the initial ontologies were constructed in part based on database schemas provided by the end-user organisations, from which an ontology was constructed that simply mirrored the database schemas used by the police organisations. Later on, in an effort to abstract away from these specific database schemas and build a more generic policing domain model, we found that this required us not only remodel to the directly affected ontology, but also, due to assumptions that we had made concerning key concepts, required extensive modifications of several other dependant ontologies. This might have been avoided had the reuse decisions and their trade-offs been more thoroughly discussed early on in the project.

SSyncAHD

The SSyncAHD project domain, veterinary epidemiology, borders and overlaps with the biomedical domain, which is rich with prior work covering a variety of content that is highly relevant to the project: genetics, species, diseases, organ systems, etc. Naturally, the project participants were very keen on reusing as many of these existing resources as possible—using ODPs as a kind of “glue” to be used with existing resources to connect those resources into one joint ontology. The participants were less certain about how to go about such reuse in practice and expressed concern that the XD methodology did not cover or discuss ontology reuse challenges (apart from reuse of ODPs). As one of the participants put it, when trying to reconcile the notion of ODP reuse with that of ontology reuse:

“Let’s just say that there exists a pathology ontology, PATHOG, that exists and that is maintained by people. Let’s say it’s not entirely suitable—let’s say it covers 500 % of what we need. So we only need 20 % of what’s in PATHOG, put in as a little box. So what’s the practice? Or, what if it’s more or less deprecated, it doesn’t really get used or maintained, is it appropriate form to take that and use it within your own ontology? Or what if it has a public address on the web but was never really published externally? What’s the right way to go about putting these boxes in our own ontology framework?”—Developer K

After some discussion (in which the author took care not to intervene), the participants suggested four different approaches to ontology reuse that could be evaluated and considered for use within the SSyncAHD project:

- *Importing*: Reuse an existing ontology wholesale using the `owl:imports` property, making all of the reused ontology’s assertions hold also in the target ontology, essentially making the reused ontology a part of the target ontology.
- *Remote references*: Reuse individual OWL classes and properties in constructing axioms in the target ontology without importing the reused entity definitions, but rather by referring to them by their individual entity IRIs. For example, defining a local class `Human` that that is defined to be equivalent in extension to `foaf:Person`⁵.
- *Partial cloning*: Copy the structure of the needed parts of the reused ontology (not necessarily the whole of it) into the target ontology, assigning the copied entities new IRIs within the target ontology namespace.

⁵From the Friend-of-a-Friend vocabulary, <http://xmlns.com/foaf/spec/>.

- *Slicing*: Copy the needed parts of the reused ontology verbatim into the target ontology, maintaining the original entity IRIs (the approach is named to illustrate how the users in this manner extract certain *slices* of the reused ontology).⁶

While the participants understood (at least in theory) how they could apply the above reuse approaches, they were rather uncertain about some of the practical consequences and implications of this selection. Some issues that they considered of particular importance concerned intellectual property implications, and the possibilities of retaining provenance links from the developed ontology to the reused source ontology. Questions relating to the former issue included: who owns the copyright of an ontology structure, does that copyright still apply if the structure is cloned into a new namespace, and if so, how large or small a portion can legally be cloned or reused from an IP perspective, etc? Questions relating to the latter issue included which ontology or RDF/RDFS structures should be used to indicate provenance of reused entities, and how one might indicate the provenance of structures that are larger than individual entities. If the participants had had at their disposal some sort of decision tree or check list providing guidance in these matters, they would likely have been much more confident in deciding which resources to reuse and how.

6.2.2 Suggestion: An Ontology Reuse Checklist

The idea of reusing existing ontologies in an XD process is not new—the NeOn methodology from which XD stems (introduced in Section 2.3.3) discusses ontology reuse extensively. [154] introduces the NeOn methodology and discusses on an overarching conceptual level how ontology reuse fits into it. Per this methodology a distinction should be made between the scenarios in which an ontology can be reused as-is, and the scenarios where some re-engineering of ontology resources needs to take place and/or several ontology resources need to be aligned or merged. [50] goes into greater depth in describing the steps prescribed by the NeOn methodology to be taken to find and to reengineer a reused ontology. The former task is described in some detail, but the latter task is only covered very briefly—it is described as consisting of steps to prune the reused ontology of unnecessary content, enrich that ontology with needed content, translate the ontology into a suitable format, adapt to naming conventions and other design criteria of the target ontology, and finally, to evaluate the resulting ontology module. None of these steps are described on a level that is applied enough to provide sufficient guidance to a less experienced ontologist. Furthermore, none of the steps are discussed in terms of the three levels of ontology reuse granularity also presented in [50]: whole ontology reuse, ontology module or part reuse, and reuse on ontology statement level. Given the observations discussed

⁶This approach can be simplified by the use of a tool such as MIREOT [36].

in Section 6.2.1, the author believes it would benefit the XD community if more practical guidance on ontology reuse in XD could be developed. In the following section, an attempt at providing such practical guidance is presented.

To the four approaches to ontology reuse discussed by participants in SSynAHD (*Importing*, *Remote references*, *Slicing*, and *Partial cloning*) we can add a fifth approach: *Patternising* the reused ontology, that is, extracting the needed parts of the ontology in the form of modules⁷, constructing new reusable ODPs from those extracted modules, and then instantiating these new ODPs into the target ontology.

All of these approaches have some merits and some drawbacks. In order to select one, the ontology engineer would need to evaluate these merits and drawbacks and their relative weights in the context of their own project. For the purpose of simplifying such an evaluation, the author has developed a list of attributes of ontology reuse approaches:

- **Implementation Complexity:** How much effort is required to apply the approach? *Importing* requires very little effort to implement, consisting only of adding a single `owl:imports` axiom (though further alignment of the imported entities may add complexity). *Remote references* is also rather low effort, requiring the addition of alignment axioms as needed throughout the ontology. *Slicing* requires more work, in terms of having to select the specific parts of the source ontology to reuse, and then isolating them (though as mentioned above, tools exist to streamline this process). *Partial cloning* requires even more work, as it necessitates the minting of new IRIs for copied entities. Finally, *Patternisation* is the most expensive approach, as it requires generalising a particular solution, enriching it with metadata, and subsequently instantiating it again into the target ontology.
- **External Reasoning Dependencies:** When executing reasoning over the ontology that results from reuse, would any external references need to be resolved in order to derive all intended inferences? The *Importing* and *Remote references* approaches depend on ontologies external to the project namespace(s), so the answer for these approaches would be yes. The answer for the *Partial cloning* and *Patternisation* approaches, in which reuse is performed within the project namespace, would be no. The *Slicing* approach also makes use of external namespace(s), but redefines these to appear local, so resulting ontologies would therefore not depend on external ontology resources (given that the slicing approach was carried out carefully).
- **Content Overhead:** To how large a degree would the ontology that results from reuse contain non-essential content? This of course depends on the degree to which the reused ontology matches the problem

⁷[41] provides an overview of several ontology modularisation techniques that may be suitable for this task.

space, but certain generalisations can be made. The *Remote references* approach adds no content at all beyond the problem domain, so has zero overhead. The *Slicing*, *Partial cloning*, and *Patternisation* approaches have low overhead, as generally only relevant content would be extracted from the reused ontology. The *Importing* approach could lead to potentially large content overhead, depending on the ontology used.

- **Reused Content Modifiability:** To what degree can the reused content be modified when refactoring further along in the project? The approaches that result in ontologies with external dependencies (*Importing*, *Remote references*) generally do not support content modifiability, while approaches that result in ontologies that do not have such dependencies (*Partial cloning*, *Patternisation*) do support modifiability. As discussed above the *Slicing* approach redefines remote references to appear local, so in theory, it would allow modifiability. In practice, however, modifying entities in a namespace one does not control entails risks of inconsistency, particularly if the resulting ontology is to be reused by other parties who are not aware of this design choice.

Based on these characteristics the author has developed a set of questions that the ontologist may ask themselves in order to narrow down the suitable choices for a particular ontology reuse scenario:

- *How much extraneous content does the ontology to be reused include?* If the source ontology contains a large amount of extraneous content then the developer should avoid *Importing* the reused ontology, as this will carry over all of that extraneous content to the target ontology, which may impact usability and computability characteristics of the resulting ontology.
- *What is the purpose of the reuse — alignment to known good practices, or structure reuse?* If the purpose of the reuse is primarily to align to practices (i.e., vocabularies or ontologies that are well-established in some community or context) then the *Remote references* approach is viable. If the purpose of reuse is to copy and reuse design structures, then this approach is unsuitable.
- *Are labels and IRIs in the reused ontology cognitively relevant in the target domain or use case?* If the answer is no, a reuse approach should be selected in which new IRIs are minted and labels applied to reused structures (e.g., *Remote references*, *Partial cloning*, or *Patternising*).
- *Will the reused portions of the reused ontology be used only once, or repeatedly (in this project or others?)* If the reused portions will be reused multiple times, the *Patternising* approach should be selected.

- *At what stage of the development process is the project presently?* At the early stages of a development project it is easier to commit to reuse of large structures (*Importing*) or structures that should not be changed (*Remote references*, *Slicing*). Later in the project, approaches with smaller impact and which are more flexible with regard to modification and adaptation (*Partial cloning*, *Patternising*) are likely to be more appropriate.
- *Will the target ontology subsequently be reused outside of the original deployment scope or system?* If the answer is yes, future users would probably benefit from the developed ontology being as homogenous and self-contained as possible, that is, if the use of multiple namespaces or ODP modules (i.e., *Slicing*, *Patternising*) were avoided and generally the use of external resources (*Importing*, *Remote references*) minimised as far as possible.

6.3 Context-Based Methodology Adaptation

The eXtreme Design methodology, as presented in [131] and [22], makes certain assumptions about the context in which an XD project is run (though not all of these assumptions are necessarily explicitly stated). The party acting as the project's initiator and requirements source is assumed to be involved in the project in a manner resembling a project member, rather than a customer, aiding the team in with prioritisation of requirements and testing of results throughout the project. As XD does not emphasise training or skill development it is assumed that the project members are proficient ontology engineers to begin with. The developers are assumed to be located in such a manner that they can work in pairs (a defining feature of XD), and the team they operate in is assumed to be large enough that ontology complexity would need to be encapsulated into semi-independent modules of functionality that would be released into the project once a pair has finished iterating over a certain set of requirements.

While the above listed assumptions may hold for certain classes of projects, the author has also found the XD methodology of breaking apart and managing ontology requirements and engineering complexity to be useful in projects that do not adhere to this somewhat idealised mould. In such projects, XD can benefit from certain practical adaptations to the project context. The following section describes the author's experience of using XD in such projects, and suggests tools for characterising and adapting XD to such projects.

6.3.1 Observation: Real World XD Project Contexts

As previously discussed in Section 6.1.1, eliciting and formalising proper requirements was a challenge in VALCRI. The persons involved in the on-

tology engineering work packages represented domain experts and software developers, but there was no representation of the intended end-user organisations who were the actual requirements sources. The lack of such customer participation meant that requirements (written in several different formats) had to trickle through the project upper management and the project software developers before being delivered to the ontology engineers, who then reformatted these requirements into the user stories that they worked with. The ontology engineers were unable to verify the completeness or veracity of these user stories against the end-user organisations who were the actual project customers. This work process caused several start-overs on the ontology engineering work, and a lot of wasted effort. While the process was clearly not optimal, it is not far removed from the reality of many software engineering projects; getting the customer as involved in the project as an agile process like XD requires, is a decidedly non-trivial exercise [115, 160].

Similarly, as mentioned in Section 6.1.1, in the OSTAG project the developed ontology was to be used by software developed by a senior researcher who was not himself involved in the day to day development of the ontology. This inaccessibility of the requirements source led to an extra iteration to rectify problems in the initially released version of the ontology, which could easily have been avoided had the senior researcher been briefed more frequently about the development progress. If the XD methodology is to be employed in project contexts such as VALCRI and OSTAG where the customer is not an active participant, additional tasks may need to be performed early in the project to account for this.

Another challenge observed in multiple projects concerns the distribution of the development team preventing pair development. In the VALCRI project the developers were widely distributed in terms of physical location, time zone, and employers. One developer was located in Belgium, three in two different cities in Sweden, two in the United Kingdom, one in Poland, one in Germany, and one in the USA. In total, the developers were spread over three time zones, with a total of 9 hours of time difference, at six different companies or institutions. None of the developers worked exclusively on VALCRI, but all had other responsibilities in parallel. In the SSyncAHD project the situation was similar: the three main developers were located in Sweden, in Switzerland, and in Canada, at different employers, and across five time zones. Hence in both these projects it was very difficult, even with modern teleconferencing solutions, for the developers to work in pairs as prescribed by XD. In VALCRI a lot of effort was spent on developing communications practices, distributed ontology engineering tooling, requirements management tooling, etc. In the end, in spite of all this effort, the project never achieved a situation where the distributed ontology engineering worked as intended. In SSyncAHD large travel expenditures were required to put developers in the same room, which would become infeasible if the project were to scale beyond the initial prototyping stage at which it remained while the author was involved. Clearly more work is needed in

this area.

As discussed in Section 6.1.1, the VALCRI participants had varying degrees of knowledge about ontology engineering, ranging from quite experienced to entirely inexperienced. Some required training in order to become proficient developers. Others did not become proficient developers in spite of receiving such training, possibly because the training material was developed based on incorrect assumptions about trainee proficiency, or because the initial training was primarily in the form of self-study. In SSyncAHD the author observed similar differences in ability, and similar challenges regarding how to develop tutoring and mentoring to support developers with less experience. The XD methodology does not provide any guidance on how to set up and train an XD team for increased chances of success—such guidance would have benefitted both of these projects.

An interesting observation from the OSTAG project concerns the manner in which the two ontology developers worked with regard to modules. While the developers largely adhered to the XD process (breaking apart the problem domain into user stories, formalising these stories into competency questions, solving the questions one by one), they deviated from standard XD practices with regard to how they implemented their solutions. Firstly, due to the lack of suitable ODPs, most of the ontology engineering work was in fact from scratch, not reusing ODPs. Secondly, while the work was performed in an iterative manner, the developers soon realised that releasing each iteration as a self-contained module, and then refactoring it with the previously developed ontology, was a waste of time, given that there was only one development team. Instead, they decided to work on the main development branch and forego modularity throughout the project. This adaptation to XD would probably not work in larger projects with more development pairs whose work needed to be integrated in a controlled process, but for the purposes of this small project, it worked very well and saved the developers a lot of time and effort.

6.3.2 Suggestion: Project Adaptation Questionnaire and Recommendations

In order to allow non-ideal ontology engineering projects like those described above to benefit from the use of ODPs and the eXtreme Design methodology, the author has developed the following project characterisation questions, and the accompanying method adaptation recommendations.

Customer Involvement

Questions to ask: *To what degree will the customer/requirements source be actively participating in the project?, How often during the project runtime will the results be evaluated or audited by the requirements source and/or their customers?*

If the customer participation can be characterised as “low” or “infrequent”, that is, if traditional agile methods are unsuitable:

- Schedule one or more extra expanded scoping and requirement meeting(s), at which key stakeholders (development team lead, customer lead, senior developers) decide on:
 - Overarching technical or social requirements for project—such as system architecture, information security/privacy concerns, legislation or compliance considerations, etc.
 - Customer and project team responsibilities and communication interfaces.
 - Critical toll points or delivery deadlines for the project.
- Ensure that sufficient amount and quality of documentation of the target system and/or domain exist and are made available such that the development team can elicit requirements from that documentation using established methods (e.g., by transforming phrases from that documentation into instance-free sentences, and reformulating those sentences as competency questions, as suggested in [22, p. 33-34])

Team Distribution

Questions to ask: *How will the development team be located geographically?, How often will project members be able to attend meetings face to face?, Will participants be located so that they are able to work in pairs at the same site?*

If the project and/or the development team can be characterised as being distributed (whether geographically or organisationally) rather than placed in close proximity:

- Ensure that a joint ontology development workspace (e.g., a shared triple store, WebProtégé, or similar) be made available and used for ontology development.
- Ensure that a requirements management system is implemented and that it is well understood by all participants prior to the ontology development starting.
- Ensure that other support and communications systems (email lists, issue trackers, shared document folders, etc.) are implemented prior to development starting. Given established user preferences for white-board drawings when doing conceptual modelling, also consider deploying support systems for such prototypes/doodling.
- Ensure the availability of customers or other requirement sources in the project to visit and collaborate with developers at each development site or unit.

- Prior to project initiation, structure the QA and release process, both in terms of internal releases of modules to the project, and in terms of releases to external stakeholders. This includes setting conditions on what is considered an acceptable release (e.g., with regard to test protocols) and assigning the required authority (both organisationally and in any required IT systems) to the Ontology Release Engineer.

Team Proficiency

Questions to ask: *What is the initial degree of ontology engineering maturity and capability among the team and the customer/requirements source?*

If the initial project capability can be characterised as being low:

- Schedule ontology engineering training seminars to begin as early as possible.
 - Ensure that the ontology IDE and other tooling to be used in the real project have been selected and made available to the trainees at the outset of training seminars. While the OWL language used may be identical, there are significant differences between using different ontology IDE:s, such as Protégé or TopBraid Composer.
 - As far as possible, run tutor-led sessions where the tutor is physically in the same room as the trainees—this is far superior to self-study or tutoring via video link.
 - However, to bootstrap the learning process and make optimal use of the tutor’s and trainee’s time, some initial self-study may be useful. Resources such as [123] and [92] are aimed at novice developers and can be very useful in this case.
 - For further efficiency and increased pedagogical effect, ensure that the scenarios employed in the training seminars relate to the domain and problem that is to be modelled in the project—ideally, output from the training sessions can then be reused as the project progresses.
- Plan training and resource allocation with the goal that at least one person filling the *Ontology Expert* role (see Section 6.1.2) should be available to lead the training and work available at each development site or partner.

6.4 Summary: eXtreme Design 1.1

The three previous sections have presented improvements that support eXtreme Design users and project managers, and allow the eXtreme Design project methodology to work in non-optimal project situations. These improvements are instantiated as concrete and usable artefacts, such as listings, recommendations, questionnaires, etc., as summarised in Table 6.3. If

the XD methodology were itself a version-managed artefact, the improvements presented here (which add features in a non-invasive and backward-compatible manner), would certainly justify incrementing the version number of said artefact to XD 1.1. The following section summarises and illustrates these improvements.

Table 6.3: XD 1.1 support artefacts.

Artefact	Section introduced
XD Roles and Responsibilities Listing	6.1.2
Ontology Reuse Methods Listing	6.2.1-6.2.2
Ontology Reuse Methods Attributes Listing	6.2.2
Ontology Reuse Method Selection Questionnaire	6.2.2
XD Project Adaptation Questionnaire	6.3.2

Section 6.1.2 recommends a number of roles to be assigned in an eXtreme Design project, suggests criteria by which these roles should be assigned, and enumerates the responsibilities that they entail. The roles are: *Project Manager*, *Requirements Source*, *Domain Expert*, *Software Developer*, *Ontology Developer*, *Ontology Expert*, and *Ontology Release Engineer* (several of which may, and are in fact likely to, overlap). These roles should initially be assigned at the beginning of an XD project, where the role allocation itself might act as a sanity check to ensure that all the required skills and knowledge are in fact available to the project. Thereafter, they may change as required by the project development; particularly, as developer skills increase, it is likely that the number of *Ontology Experts* who can act as *Ontology Release Engineers* will increase.

Table 6.4: Ontology reuse approaches and characteristics.

Approach	Complexity	Dependencies	Overhead	Modifiability
<i>Importing</i>	Very low	Yes	Varies	No
<i>Remote refs</i>	Low	Yes	None	No
<i>Slicing</i>	Medium	No	Low	Yes
<i>Partial cloning</i>	High	No	Low	Yes
<i>Patternising</i>	Very high	No	Low	Yes

Sections 6.2.1 and 6.2.2 introduce and describe five different approaches to ontology reuse in an eXtreme Design project. Section 6.2.2 in particular characterises these approaches based on several criteria that are important for reuse, as shown in Table 6.4. This section also provides a checklist of questions the developer may ask themselves when deciding which reuse approach to employ. Figure 6.1 simplifies this checklist by restructuring it into a decision tree structure. By answering the questions and following the designated arrows, this tree allows the developer to quickly and easily select an appropriate ontology reuse approach. Note that the questions in this

figure have been reformulated somewhat compared with those presented in Section 6.2.2, in order to allow only yes/no answers. Note also that this decision tree is, as stated, a simplification, that is, it includes only a subset of all possible paths between the included questions.

Section 6.3.2 provides guidance on how to adapt eXtreme Design to non-optimal project contexts, in the form of a number of questions project managers may ask themselves, and associated recommendations on actions to take to increase the chances of project success. Figure 6.2 illustrates how these actions, together with the actions proposed in Sections 6.1.2 and 6.2.2, modify the original eXtreme Design workflow. The new tasks proposed in this chapter are marked with double outlines in this figure. Note that this figure assumes that ontology reuse issues can be dealt with early on in the project, prior to the start of development loop iterations; in some cases this might not be possible, and in such cases ontology reuse would be decided on and implemented within one or more iterations of the development loop below the step labelled *Select story*.

The improvements to eXtreme Design presented in this chapter are derived from researcher observations of methodology shortcomings and the practical solutions to those shortcomings as implemented in the VALCRI, SSyncAHD, and OSTAG projects. However, it should be clear, that neither the individual improvement suggestions, nor XD 1.1 as a whole, have been formally evaluated within the scope of this PhD project.

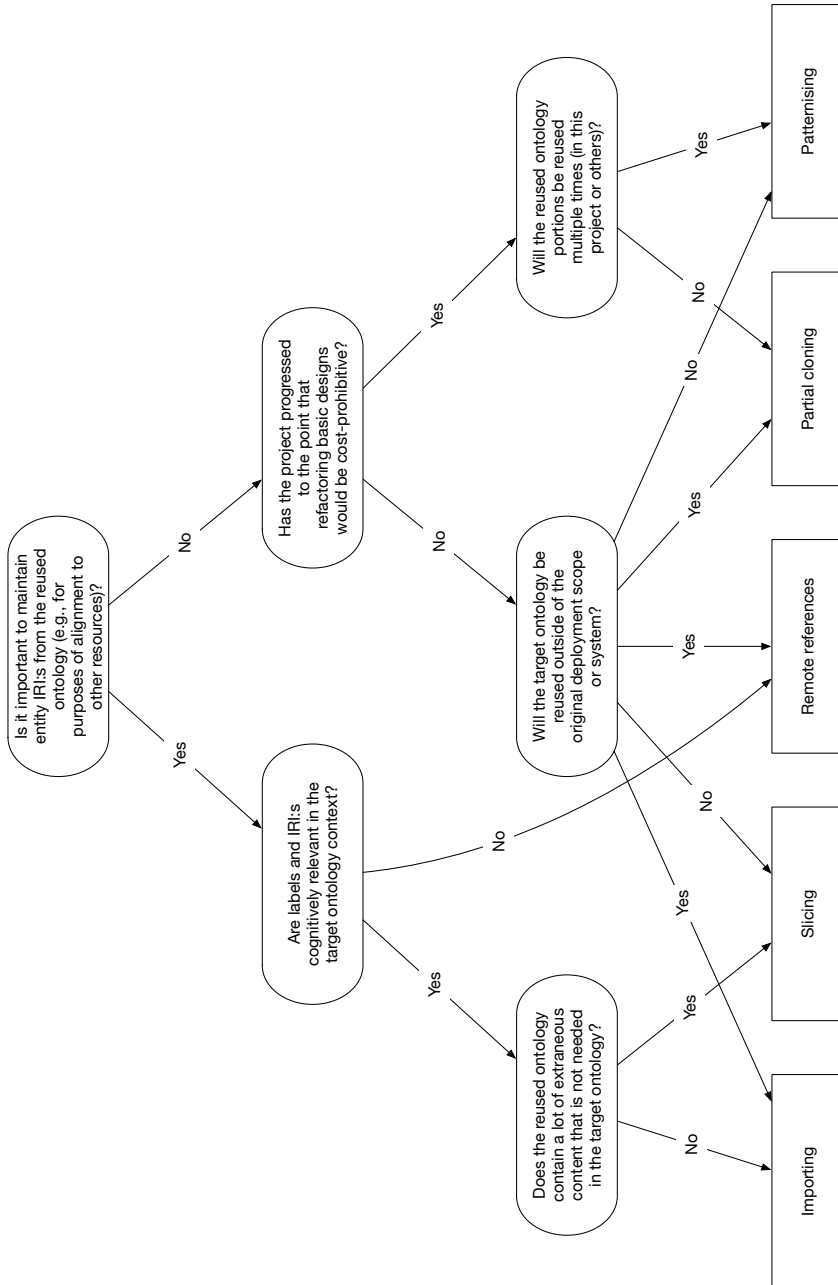


Figure 6.1: Ontology reuse approaches decision tree.

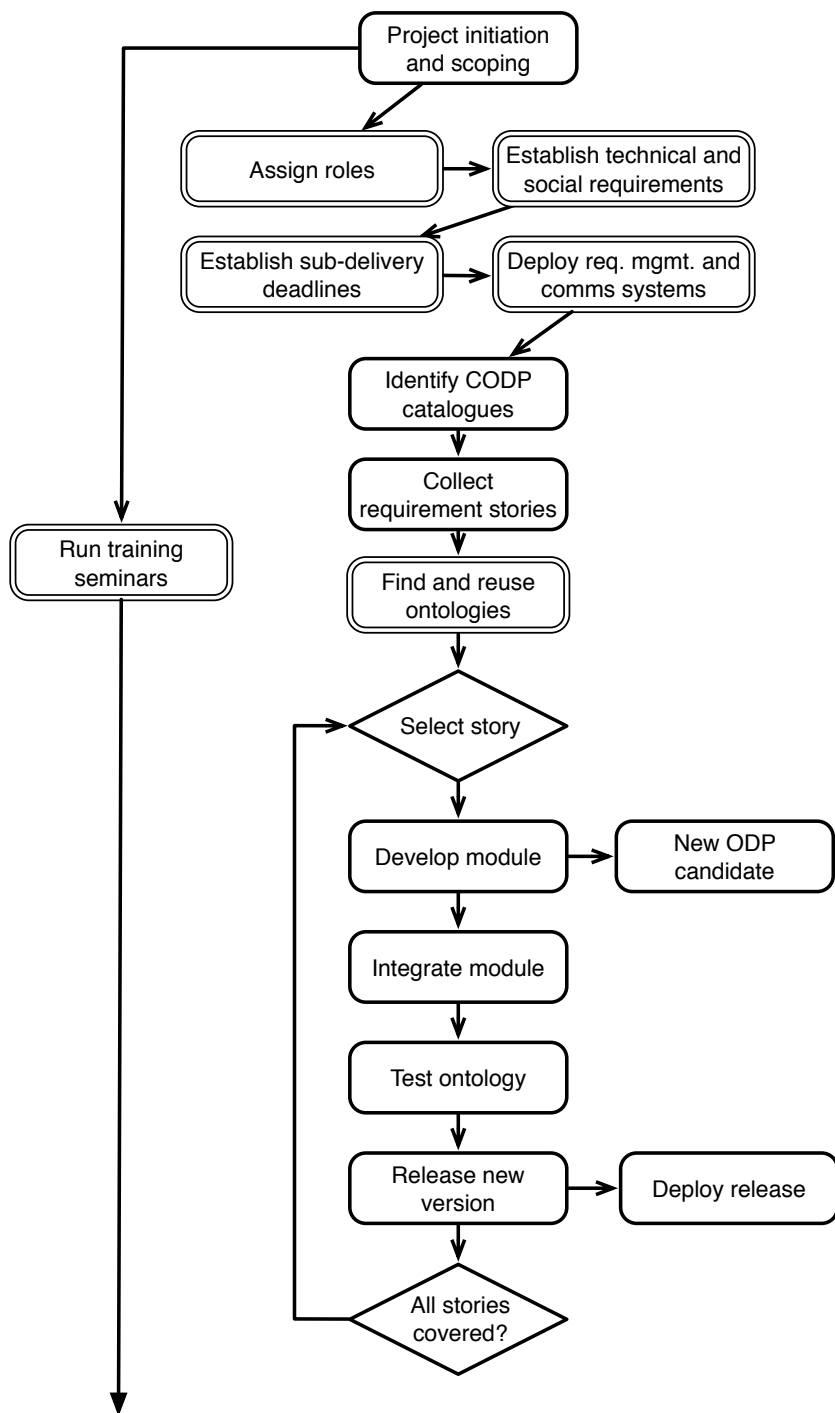


Figure 6.2: eXtreme Design 1.1 workflow.

Chapter 7

Discussion

This chapter discusses the degree to which the research questions have been answered, and the consequences of the findings in terms of impact on the academic community, impact on practice, and issues that remain open for future studies.

7.1 Research Questions Revisited

The research questions posed in Chapter 1 have been treated and answered as follows:

- *Which ODP features or qualities are important in supporting pattern understanding and use?*

To answer this question, an ODP quality model was developed and evaluated, as discussed in Chapter 4. The quality model consists of several components: an ODP quality metamodel, a set of quality characteristics, and a set of quality indicators (all of which are presented in Section 4.4). The model and its components provide an answer to the research question, by indicating how different attributes or features of an ODP (measured by the quality indicators) contribute to ODP quality characteristics (such as *Functional Suitability*, *Usability*, *Maintainability*, etc.).

It should be noted that this is a partial answer. The ODP quality model does not exhaustively cover the full spectrum of all possible aspects of ODP quality. Given the breadth of all measurable attributes of ontologies and ODPs, achieving such complete coverage is neither feasible nor useful. Instead, this work primarily contributes knowledge about the types of features that matter to inexperienced ontologists, in line with the intended objective of the dissertation to support ODP

use by this class of users. Further, the model does not cover the effects that different ontology engineering or usage contexts have on the prioritisation of quality characteristics. Until more work is performed in this area, inexperienced ontology engineers will have to decide for themselves which quality characteristics to prioritise.

- *How can the features and functionality of ODP usage tools be improved to support inexperienced ontologists?*

Answering this question requires an understanding of the tasks that inexperienced ontologists perform when using ODPs, and specifically, which of those tasks would benefit from improved tooling support. The three tooling improvements developed and discussed in Chapter 5 concern what are arguably the two core ODP usage tasks—namely, finding ODPs that are suitable for modelling some specific issue, and the instantiation of those ODPs into a target ontology. Unlike some “one-off” ODP usage tasks, the time overhead associated with manually (i.e., without appropriate tool support) performing recurring tasks such as finding and instantiating ODPs is likely to be quite significant. Also, the location and application of ODPs are likely to be among the most difficult tasks for inexperienced ontologists to perform without guidance. Finding the right ODPs to reuse would require the developer to possess a degree of familiarity with available ODPs and ODP portals, which inexperienced ontologists are quite unlikely to have. Having found an appropriate ODP, instantiating it into an ontology would require the developer to perform several repetitive and time-consuming steps, each carrying a significant risk of making mistakes. The tooling improvements presented in sections 5.2–5.1 help simplify these tasks for inexperienced and experienced ontologists alike, and in so doing, provide answers to the research question.

- *How can ODP usage methodology be improved to support inexperienced ontologists?*

For the purpose of methodology development, the author has focused on testing and improving the well-established eXtreme Design (XD) methodology. This testing and improvement was carried out as discussed in Chapter 6, and several improvement suggestions (see Section 6.4) were developed specifically to aid XD deployment in projects involving less experienced ontologists. While the efficacy of these improvement suggestions has not yet been formally evaluated, their grounding in experiences from several real-world projects is a good reason to consider them to be suitable answers to the research question.

7.2 Research Consequences and Future Challenges

Some of the findings presented in this dissertation may be of particular interest to the research community, as they imply exciting research opportunities. These opportunities concern both the development and evaluation of ontology engineering methodology and practice, and possible improvements to ontology languages and standards. The following section summarises these findings and briefly introduces such research opportunities.

7.2.1 ODP Quality Model

In developing the ODP Quality Model, the author observed several shortcomings in existing practice concerning how ODPs and their features are documented, organised, and visualised. Some examples are discussed below, together with suggestions on how researchers might work to resolve them (or, in the more complex cases, research them).

Documenting Known ODP Uses

The findings presented in Section 4.3 indicate that it is important to make any known uses of a pattern accessible through that pattern’s documentation, to ensure that users have access to usage examples when reusing the pattern. In the current format, the community ODP portal¹ has a field named ‘Known Uses’ that is intended to list other ontologies that use the same pattern, but for most patterns this field remains empty. This is puzzling, given that ODPs are typically either abstracted from existing older ontologies, or developed within some project for the explicit purpose of constructing new ontologies. Either way, some usage examples ought to exist—but when ODPs get uploaded to the community portal, examples of their instantiation do not seem to get propagated.

Examples of ODP instantiation could also be gathered when ODPs get reused. Currently the ODP portal does not include support for linking to such information. This omission is doubly unfortunate, as it not only reduces the usability of the ODPs in question, but also gives visitors to the community portal the impression that ODPs are not used very much.

Competency Question Heterogeneity

A core documentation field of any ODP is the *Competency Questions* listing. This field encodes the essential functionality of the ODP in human-readable form, and it is an important aid for ontologists when determining the suitability of an ODP for different modelling issues, as shown in Section 4.3.2.

¹<http://ontologydesignpatterns.org>

Competency questions are likely to be particularly important for less experienced ontologists, who are typically not comfortable with interpreting ODPs as just sets of description logic axioms.

In the ODP community portal, roughly 75 % of all ODPs have any competency questions defined at all. This figure could and should certainly be improved. Additionally, the competency questions that are defined vary widely in punctuation syntax and language. This lack of standardisation makes locating the right ODP needlessly difficult, particularly for a user who is new to the task. It also makes it difficult to parse the competency questions by software (e.g., for search engine indexing purposes). Enforcing proper usage of the Content Pattern Annotation Schema² for submitted ODPs could help alleviate these problems.

Free-text Documentation Quality

Further standardisation and data quality challenges become apparent when we consider other aspects of documentation, particularly free-text documentation fields. For instance, we know from Section 4.3.2 that an ODP’s stated *intent* is important in appropriateness recognisability—but in the ODP portal, the majority of ODPs have the intent field filled with a simple sentence that typically does not give guidance beyond the ODP name (e.g., for the *TimeInterval* pattern, the intent is defined as “To represent time intervals”).

The well known Gang of Four book on OOP design patterns [54] provides a template covering not only the fields needed to document object-oriented design patterns, but also examples of questions that these fields should answer, and in several cases, minimum requirements on content (e.g., that at least two known use cases should be documented for each pattern, or that a certain syntax should be used for graphical representations). Applying and enforcing similar requirements on ODP documentation quality would benefit ODP quality and the quality of the ODP community portal in a very tangible and visible way, which in turn could drive increased adoption of ODP usage.

ODP Interdependencies

As shown and discussed in Sections 4.4.4 and 5.3, a large transitive import closure can be harmful to ODP usability and reusability. Importing foundational concepts that are outside of the scope of the immediate problem that the ODP models could potentially be confusing for users, particularly inexperienced ones. Additionally, doing so may reduce an ODP’s reusability, if those foundational concepts are not compatible with all possible usage scenarios. Therefore, for optimum usability and reusability, an ODP should make as few ontological commitments outside of its intended usage scope as possible.

²<http://www.ontologydesignpatterns.org/schemas/cpannotationschema.owl>

At the same time, it could also be beneficial to be able to express that an ODP can be easily used together with a particular set of other ODPs. For instance, we may wish to express that a *Trajectory* ODP is compatible with, or possibly even includes in it, concepts that are provided by a *Place*, *Location*, or *SpatialPoint* ODP. There is no currently established practice for how to express such compatibility or inclusion requirements without making commitments regarding the design or structure of those ODPs. It would benefit the ODP community if such practices (and the associated meta-data vocabularies or tooling) were developed. For this purpose, it could also be beneficial to borrow a concept from the object-oriented programming paradigm and think in terms of *ontology interfaces* that hide (possibly through partial or intransitive OWL import features, if such things were developed) implementation-specific details.

Recent work co-authored by the author and a large section of the ODP research community, [82], touches upon these issues and discusses the need for pattern standards and representation languages, both in terms of visualisation of this type of ODP relations, and in terms of formal machine-parsable annotations or languages supporting new innovative tooling for pattern composition.

ODP Visualisation

Several promising ontology visualisation notations have been developed (for instance using concept diagrams to represent ontologies [151], or the VOWL syntax [112]), yet none have reached sufficient adoption to be considered the de facto standard. As such, ontology engineers must keep track of multiple competing visualisation languages, at significant cognitive overhead, and arguably negating at least some of the usability effects of having visualisation in the first place. This situation also has the effect of putting industry practitioners (who value the availability of good standards) off of the idea of RDF/OWL modelling.

Additionally, even if a standard ontology visualisation language were to be developed, such a language might not be suitable for ODPs, as ODPs have several attributes and uses that set them apart from ontologies in the general sense. As discussed above, one would likely want ODPs or ODP-based modules to be composable into larger ontologies in some sort of modular fashion, which would require a visualisation language to support some concept of modules or components. Moreover, it is not unusual for ODPs to extend or specialise one another, so the visualisation language would need syntax to represent such relations between ODPs. Such a language would likely need to be developed in tandem with a formal representation as discussed above, to guarantee consistency and coherency of representation.

Utilising the ODP Quality Model

The quality-related findings and recommendations of this dissertation might be implemented in practice by way of documentation and tooling that could aid users in the selection and instantiation of ODPs. Initial work in this direction is included with the XDP tooling as discussed in Chapter 5, but there is room for further research and development in this area. For instance, it might be beneficial to provide hints (or warnings) in ODP documentation, search engines, and instantiation wizards, which could notify the user if an ODP were to exhibit characteristics that are known to affect important qualities detrimentally or positively.

Other future research opportunities include further studies on which qualities are important to users, in which types of modelling cases, and for which purposes. For instance, if the end goal of an ontology-using software project is to provide data exploration facilities, it is likely that the requirements for the ontologies that get used will be different from a project where the end goal is to support data integration and querying. If such requirements could be captured and formalised, we might be able to develop scenario- or context-based ODP usage profiles, which could then be implemented in either method or tooling to aid users in selecting and utilising ODPs better given their specific use-case.

7.2.2 Tooling Support

The work on ODP tooling presented in Chapter 5 covers improved support for finding ODPs, and for instantiating or specialising ODPs. The findings that relate to the latter two tasks carry implications that warrant further discussion, as presented below.

Additional Specialisation Strategy Support

As shown and discussed in Section 5.2, there are different strategies for ODP object property specialisation, with the choice of strategy having a potentially significant effect on the results. The XDP tooling presented in this dissertation supports instantiation of ODPs taking these effects into account. However, there are additional ontology engineering tasks where strategy-aware tooling could be beneficial.

Firstly, it would be beneficial if tooling had the ability to visualise the uses of these specialisation strategies, such that an ontology engineer could quickly ascertain the suitability of an ODP-based ontology for different purposes; possibly some existing ontology metrics views could be extended with new metrics indicating the specialisation strategy. To simplify the use of ontologies built using a class-oriented strategy, displaying the “emulated” domains and ranges of specialised properties in proximity to those properties’ definitions would also be helpful for the less experienced user.

Secondly, it would likely be helpful if tooling could aid in refactoring an ontology from using one strategy to another, and in harmonising strategy use in cases where different strategies are employed in different parts of an ontology. The latter would be of special importance in ontology alignment scenarios. This problem appears to be a good use case for the OPPL (Ontology Pre-Processing Language) macro language for ontology transformation discussed in Section 2.4.3, or the more recent PatOMat framework that builds on OPPL [159, 158].

In addition to improved ODP tools, the discovery of these specialisation strategies might also warrant updates to ODP repositories; these repositories would need to provide examples of ODPs that have been specialised using the different strategies. This may necessitate new visual representations for representing universal and existential restrictions in an accessible and user-friendly manner; the work of Stapleton et al. [151] in this area appears very promising.

Implications of Template-Based ODP Instantiation

Section 5.3 introduces and discusses a template-based approach to ODP instantiation that, while not new to the research community, has previously seen relatively little study and exploitation. As shown in that same section, this approach is considered particularly desirable by the class of inexperienced ontologists that this dissertation aims to support. The approach, the motivation behind employing it, and its merits and consequences are discussed in some detail in that section. However, some of the results of this work merit further discussion, and indeed, further research.

While there may be formal definitions of what constitutes an ODP (see Section 2.4), traditionally a practical approach has been employed whereby the reusable OWL building block has been treated as the core of an ODP, if not the ODP itself (see e.g., [131]). The textual and graphical illustration of that ODP (whether displayed in a software tool or in an online repository) has been regarded as documentation or metadata. This is likely the result of the building block being the canonical representation which is, using the traditional specialisation-based instantiation approach, imported wholesale into every developed ontology module that reuses this ODP. However, with the template-based instantiation approach one could argue that there is nothing special about the building block itself; it is merely a convenient formal representation of the ODP, which can be stamped out to instantiate the ODP into a target ontology module. Per this perspective, one could further argue that the ODP documentation, graphical illustrations, and other metadata *are in fact* the canonical representation of the ODP. This perspective is more in line with how design patterns are treated in other related fields, and how they are presented in works such as [54] and [53]. Employing this perspective, the importance of improving the quality of the documentation provided in the community ODP portal becomes particularly clear. This perspective also supports the publishing of patterns in print-oriented

formats, such as through research papers or books, provided of course that the challenges regarding standardised documentation fields discussed in the previous sections are resolved.

Another consequence of the template-oriented approach is that in eschewing the usage of `owl:imports`, it removes the provenance information that can be helpful in tracking where and how an ODP has been implemented within an ontology (which may be relevant in terms of maintainability). One approach to resolving this issue would be to develop a formal language that supports representing ODP relations such as *instantiation of*, *composition of*, *extension of*, etc., as discussed in Section 7.2.1. While the aforementioned Content Pattern Annotation Schema covers some parts of this problem space, that schema has several drawbacks: it does not distinguish between annotating entities within an ODP and the ODP as a whole, it mixes in other orthogonal concerns such as requirements and testing, it does not cover all the possible relations one might wish to express, and it has no associated visual representation.

A related challenge and issue to consider as future work is whether it is feasible to develop a formal language for describing how an ODP is intended to be instantiated, and once instantiated, how it might then be utilised or queried. To answer the former question, we would need to be able to formalise which constituent entities (if not all of them) should be cloned or specialised, and in the case that cardinality or value restrictions should be applied to the specialised or cloned entities, we would need to formalise how these should in fact be constructed in keeping with the ODP author's intent. For the latter question, we would need to be able to pair the ODP (and its instantiations) with query archetypes or templates in SPARQL or other suitable languages, indicating which questions the ODP can answer. Ideally, with the aid of such a language (and with the aid of *ontology modules* or *components*, and the associated *ontology interfaces* as suggested in Section 7.2.1), we will be able to develop tooling in which users can create ontologies simply by dragging and dropping the relevant ODPs into their workspace, and those ODPs would compose themselves into usable and queryable ontologies or ontology modules semi-automatically.

7.2.3 Methodology Development

The results presented in Chapter 6 include several improvements to the XD methodology, but there is ample room for further development and research in this area. One such development would be to construct guidance intended to support users when no suitable ODP exists for a given modelling challenge. In such situations, the developer might choose to modify or extend an existing ODP, to develop a new ODP from scratch, or to forego the ODP paradigm entirely and simply develop the ontology model. These choices are associated with trade-offs, and it is likely that inexperienced developers would require further guidance in this area.

Another issue is how well aligned XD is with the workflows and practices inherent to software development. The benefits of understanding (and possibly improving) this alignment are twofold. Firstly, since software development best practices are encoded in a variety of project support systems (issue trackers, versioning systems, etc.), aligning XD with such practices may enable reuse of existing support systems from software engineering for XD-based ontology engineering projects. The availability of good tool support is likely to be particularly attractive to inexperienced developers. Secondly, it is rather rare that ontologies are developed in isolation—typically, as discussed in Chapter 2, ontologies are instead used within some software system, and consequently, the interplay between ontology and software is critical. Challenges here include versioning (of ontologies, query libraries, and code libraries), testing (both at the unit and system levels), and packaging/distribution of ontologies and/or dependent RDF data.

Yet another methodology (and possibly tooling) issue that would be worth exploring relates to how to handle cross-cutting concerns such as time-indexing, spatial indexing, metaphysical grounding, etc., in ODP-based ontology engineering. Overlapping ODPs can represent partially similar modelling challenges, but vary with regard to these concerns. For instance, we have *Part Of*³ and *Time Indexed Part Of*⁴, *Participation*⁵ and *Time Indexed Participation*⁶, *Place*⁷ and *Spatio-Temporal Extent*⁸. This situation is hardly surprising given the perspective in XD that there is a conceptual mapping between the problem space and the solution space, whereby a specific problem is mapped against one or more generic use cases that each have an ODP-based solution. If we consider instead that a generic use case might be associated with multiple ODP-based solution representations, depending on how such cross-cutting concerns are handled, we might end up with a rather different workflow and toolchain. It would benefit the ODP community to initiate a discussion on such matters before the proliferation of partially overlapping ODPs becomes too confusing for casual users—as indicated by the listing above, this point may be approaching.

7.3 Summary of Future Work

To summarise the discussion above, the future research and implementation challenges for the ODP research community concern two categories of work: firstly, improving the quality of existing patterns and pattern repositories, and the features of available tooling and methodology; and secondly, developing new standards, tools, and techniques which supports next-generation,

³<http://ontologydesignpatterns.org/wiki/Submissions:PartOf>

⁴<http://ontologydesignpatterns.org/wiki/Submissions:TimeIndexedPartOf>

⁵<http://ontologydesignpatterns.org/wiki/Submissions:Participation>

⁶http://ontologydesignpatterns.org/wiki/Submissions:Time_indexed_participation

⁷<http://ontologydesignpatterns.org/wiki/Submissions:Place>

⁸<http://ontologydesignpatterns.org/wiki/Submissions:SpatioTemporalExtent>

innovative yet user-friendly, ODP-based ontology engineering. The work in the first category will be evolutionary and consist of tasks such as improving the documentation coverage in the community ODP portal, standardising the documentation fields and documentation syntaxes used (in text, illustrations, and metadata annotations), and, indeed, evaluating the methodology improvements suggested in Chapter 6 of this dissertation.

The work in the second category is more forward-looking and possibly more interesting for researchers. Key contributions might include formal definitions and tooling implementations of the notions of ontology modules or components⁹, ontology interfaces, and partial or intransitive ontology imports. Other important contributions include methods of allowing ODPs to be annotated such that they can carry embedded documentation about how they should be instantiated, and how, once instantiated, they can be utilised (e.g., which types of queries they support, or which reasoning tasks they enable).

⁹Including any relations that may need to be expressed between such modules/components, or between modules/components and other semantic resources.

Chapter 8

Conclusions

Ontology Design Patterns (ODPs) are intended to simplify ontology engineering by packaging known good solutions to recurring problems in such a way that they can be easily reused, even by less experienced ontologists. Initial evaluations of the effects of ODP usage show promising results [24, 21], but there are still many unknowns concerning ODP quality, the need for ODP tool support, and the suitability of existing ODP methodologies.

The objective of this work was: *To develop an understanding of important ODP quality issues, and to develop ODP tooling and usage methodologies as required to support the use of ODP-based ontology engineering, particularly by inexperienced ontologists.* To achieve this objective, the author defined and studied three research questions:

1. Which ODP features or qualities are important in supporting pattern understanding and use?
2. How can the features and functionality of ODP usage tools be improved to support inexperienced ontologists?
3. How can ODP usage methodology be improved to support inexperienced ontologists?

Concerning the first research question we conclude that ODPs can be discussed in terms of quality characteristics, conceptually not unlike the quality characteristics employed in the evaluation of other sorts of conceptual models or software systems (Chapter 4). Quality characteristics can be measured via several quality indicators that cover aspects of ODP documentation, of formal representation or axiomatisation, and of usage by ontologists. Some of the effects of indicators on quality characteristics could be regarded as counter-intuitive. For instance, even though property restriction axioms are not displayed graphically in most tools (but rather as potentially confusing logic syntax), a higher *Property Restrictions Count* contributes positively

to ODP *Learnability*¹. Occasionally the features underlying two quality indicators clash, giving rise to trade-offs between quality characteristics that an ontologist (particularly one with limited experience) needs to be aware of. For instance, defining domains and ranges for object properties can help illustrate the classes with which those properties are intended to be used (again, increasing *Learnability*), but this also reduces the set of classes that the properties could be used with (lowering *Reusability*). Section 4.4 presents a listing of the developed quality characteristics, quality indicators, and quality trade-offs.

Concerning the second research question we describe the development of three improvements to existing methods and tooling, which are particularly important to the inexperienced ontologist (Chapter 5). These include improvements to ODP search, taking into consideration ODP-specific characteristics; documentation of three previously unpublished property specialisation strategies, and tool support for instantiating ODPs using these strategies; and heuristics and tool support for template-based, as opposed to specialisation-based, ODP instantiation. A high-quality ODP search engine is important for all ODP users, but inexperienced users in particular are likely to benefit from improvements, as they are less familiar with the growing set of available ODPs than more experienced users. The choice of which property specialisation strategy to employ carries with it trade-offs that are not immediately obvious or intuitive; supporting less experienced users in making this choice reduces the risk of unwanted consequences in the resulting ontology. Finally, Template-based instantiation has been shown to be more intuitive to some users than the established practice of instantiation via specialisation, in terms of both process and result. All three of these improvements have been integrated into an open source tool, as described in Section 5.4.

Concerning the third research question, we conclude that the established ODP usage methodology (eXtreme Design) would benefit from recommendations concerning developer roles and responsibilities, and guidance on the reuse of non-ODP resources (Chapter 6). Furthermore, as many ontology engineering projects do not display all of the characteristics required for an optimal eXtreme Design process, the methodology would also benefit from additional adaptability to such project contexts. Section 6.4 presents a set of guidelines and recommendations intended to help overcome these issues. These include: a listing of project roles (*Project Manager*, *Requirements Source*, *Domain Expert*, *Software Developer*, *Ontology Developer*, *Ontology Expert*, and *Ontology Release Engineer*), with recommendations on how to assign those roles; descriptions of five approaches to ontology reuse (*Importing*, *Partial cloning*, *Patternising*, *Remote references*, and *Slicing*), with a decision tree to help select the most appropriate approach; and a set of

¹This is because property restriction axioms, even though they are syntactically difficult to parse, can clarify which properties and classes that are intended to be used together.

six project characterisation questions for the project managers to ask themselves, with associated actions to take depending on the answers.

To summarise, this work achieves its defined objective by providing concrete and practically applicable results that contribute to improvements in three aspects of ODP usage that are important for ODP-based ontology engineering. In so doing, this work contributes to a research field that has developed over the past twelve years from a niche idea held by a few influential researchers into a topic that is researched by an entire community of scientists. There is no indication that this development will stall. Rather, the ODP research and practitioner community appears at this time to be particularly vital, as indicated by the launch of the Ontology Design and Patterns Association², and the recent release of the anthology *Ontology Engineering with Ontology Design Patterns* [90]. The collection of open research questions that is presented in that work indicates that there is a need for further research on how ODPs and ODP-based tooling can be used to support ontology engineering work. Thus the work presented in this dissertation, combined with other efforts on ODP-based ontology engineering, will enable broader uptake of Semantic Web ontologies and ontology-based technologies over the coming years.

²<http://ontologydesignpatterns.org/wiki/ODPA>

Bibliography

- [1] R. L. Ackoff. From Data to Wisdom. *Applied Systems Analysis*, 16:3–9, 1989.
- [2] C. Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.
- [3] D. Alur, J. Crupi, and D. Malks. *Core J2EE Patterns: Best Practices and Design Strategies*. Prentice Hall, 2 edition, 2003.
- [4] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic. Stream reasoning and complex event processing in ETALIS. *Semantic Web – Interoperability, Usability, Applicability*, 3(4):397–407, 2012.
- [5] G. Antoniou and F. van Harmelen. Web Ontology Language: OWL. In S. Staab and R. Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 91–110. Springer, 2 edition, 2009.
- [6] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge: Cambridge University Press, 2003.
- [7] A. Bangor, P. T. Kortum, and J. T. Miller. An Empirical Evaluation of the System Usability Scale. *International Journal of Human-Computer Interaction*, 24(6):574–594, 2008.
- [8] D. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. Incremental Reasoning on Streams and Rich Background Knowledge. In L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, and T. Tudorache, editors, *The Semantic Web: Research and Applications. ESWC 2010*, number 6088 in Lecture Notes in Computer Science, pages 1–15. Springer, 2010.
- [9] 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. *IEEE Intelligent Systems*, 25(6):32–41, 2010.

- [10] 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.
- [11] 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.
- [12] D. Batory, C. Johnson, B. MacDonald, and D. Von Heeder. Achieving Extensibility Through Product-Lines and Domain-Specific Languages: A Case Study. In W. B. Frakes, editor, *Software Reuse: Advances in Software Reusability. ICSR 2000*, number 1844 in Lecture Notes in Computer Science, pages 83–153, Berlin, Heidelberg, 2000. Springer.
- [13] 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, 1996.
- [14] G. Bellinger, D. Castro, and A. Mills. Data, Information, Knowledge, and Wisdom. <http://www.systems-thinking.org/dikw/dikw.htm>, accessed: 2017-04-18, 2004.
- [15] 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.
- [16] T. Berners-Lee. Linked Data. <http://www.w3.org/DesignIssues/LinkedData.html>, accessed: 2017-04-18, June 2009.
- [17] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):28–37, May 2001.
- [18] E. Blomqvist. *Semi-automatic Ontology Construction based on Patterns*. PhD thesis, Linköping University, 2009.
- [19] E. Blomqvist, V. K. Chaudhri, O. Corcho, V. Presutti, and K. Sandkuhl, editors. *Proceedings of the 2nd International Workshop on Ontology Patterns – WOP2010*. CEUR Workshop Proceedings, 2010.
- [20] E. Blomqvist, A. Gangemi, K. Hammar, and M. C. Suárez-Figueroa, editors. *Proceedings of the 3rd Workshop on Ontology Patterns*. CEUR Workshop Proceedings, 2012.
- [21] E. Blomqvist, A. Gangemi, and V. Presutti. Experiments on Pattern-based Ontology Design. In *K-CAP '09: Proceedings of the Fifth International Conference on Knowledge Capture*, pages 41–48. ACM, 2009.

- [22] E. Blomqvist, K. Hammar, and V. Presutti. Engineering Ontologies with Patterns – The extreme Design Methodology. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns - Foundations and Applications*, volume 25 of *Studies on the Semantic Web*, pages 23–50. IOS Press, 2016.
- [23] E. Blomqvist, P. Hitzler, A. Krisnadhi, T. Narock, and M. Solanki, editors. *Proceedings of the 6th Workshop on Ontology and Semantic Web Patterns (WOP 2015) co-located with the 14th International Semantic Web Conference (ISWC 2015)*. CEUR Workshop Proceedings, 2015.
- [24] E. Blomqvist, V. Presutti, E. Daga, and A. Gangemi. Experimenting with eXtreme Design. In P. Cimiano and H. S. Pinto, editors, *EKAU 2010: Knowledge Engineering and Management by the Masses*, pages 120–134. Springer, 2010.
- [25] E. Blomqvist and K. Sandkuhl. Patterns in Ontology Engineering: Classification of Ontology Patterns. In C.-S. Chen, J. Filipe, I. Seruca, and J. Cordeiro, editors, *ICEIS 2005: Proceedings of the Seventh International Conference on Enterprise Information Systems*, volume 3, pages 413–416, 2005.
- [26] E. Blomqvist, K. Sandkuhl, F. Scharffe, and V. Svatek, editors. *Proceedings of the Workshop on Ontology Patterns (WOP 2009)*. CEUR Workshop Proceedings, 2009.
- [27] D. Brickley and R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, W3C, February 2004.
- [28] D. Brickley and R. Guha. RDF Schema 1.1. W3C Recommendation, W3C, February 2014.
- [29] J. Brooke. SUS: A ‘quick and dirty’ usability scale. In P. W. Jordan, B. Thomas, I. L. McClelland, and B. Weerdmeester, editors, *Usability Evaluation In Industry*, pages 189–194. Taylor & Francis, London, 1996.
- [30] A. Budanitsky and G. Hirst. Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures. In *Workshop on WordNet and other lexical resources*. NAACL, 2001.
- [31] P. Burnard. A method of analysing interview transcripts in qualitative research. *Nurse Education Today*, 11(6):461–466, 1991.
- [32] 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.

- [33] H. Chen, T. Finin, and A. Joshi. An Ontology for Context-Aware Pervasive Computing Environments. *The Knowledge Engineering Review*, 18(3):197–207, September 2003.
- [34] 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.
- [35] O. Corcho, C. Roussey, L. M. V. Blázquez, and I. Pérez. Pattern-based OWL Ontology Debugging Guidelines. In E. Blomqvist, K. Sandkuhl, F. Scharffe, and V. Svatek, editors, *Proceedings of the Workshop on Ontology Patterns (WOP 2009)*, volume 516 of *CEUR Workshop Proceedings*, pages 68–82, 2009.
- [36] M. Courtot, F. Gibson, A. L. Lister, J. Malone, D. Schober, R. R. Brinkman, and A. Ruttenberg. Mireot: The minimum information to reference an external ontology term. *Applied Ontology*, 6(1):23–33, 2011.
- [37] R. Cyganiak, D. Wood, and M. Lanthaler. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, W3C, 2014.
- [38] 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.
- [39] R. L. Daft. Learning the Craft of Organizational Research. *Academy of Management Review*, 8(4):539–546, 1983.
- [40] 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 Consortium, 2007.
- [41] M. d’Aquin. Modularizing Ontologies. In M. C. Suárez-Figueroa, A. Gómez-Pérez, E. Motta, and A. Gangemi, editors, *Ontology Engineering in a Networked World*, pages 213–233. Springer, Berlin, Heidelberg, 2012.
- [42] V. de Boer, A. Gangemi, K. Janowicz, and A. Lawrynowicz, editors. *Proceedings of the 5th Workshop on Ontology and Semantic Web Patterns*. CEUR Workshop Proceedings, 2014.
- [43] M. Doerr, J. Hunter, and C. Lagoze. Towards a Core Ontology for Information Integration. *Journal of Digital Information*, 4(1), 2006.
- [44] 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.
- [45] M. Egaña, E. Antezana, and R. Stevens. Transforming the Axiomisation of Ontologies: The Ontology Pre-Processor Language. In K. Clark and P. F. Patel-Schneider, editors, *Proceedings of the Fourth OWLED Workshop on OWL: Experiences and Directions*. CEUR Workshop Proceedings, 2008.
 - [46] M. Eriksson, J. Börstler, and K. Borg. Software product line modeling made practical. *Communications of the ACM*, 49(12):49–54, 2006.
 - [47] R. Falbo, G. Guizzardi, A. Gangemi, and V. Presutti. Ontology Patterns: Clarifying Concepts and Terminology. In A. Gangemi, M. Gruninger, K. Hammar, L. Lefort, V. Presutti, and A. Scherp, editors, *Proceedings of the 4th Workshop on Ontology and Semantic Web Patterns*. CEUR Workshop Proceedings, 2013.
 - [48] M. Fernández-López and A. Gómez-Pérez. The integration of OntoClean in WebODE. In J. Angele and Y. Sure, editors, *EON2002: Proceedings of the OntoWeb-SIG3 Workshop at the 13th International Conference on Knowledge Engineering and Knowledge Management EKAW 2002*, pages 38–52. CEUR Workshop Proceedings, 2002.
 - [49] M. Fernández-López, A. Gómez-Pérez, and N. Juristo. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. Technical Report SS-97-06, American Association for Artificial Intelligence, March 1997.
 - [50] M. Fernández-López, M. C. Suárez-Figueroa, and A. Gómez-Pérez. Ontology Development by Reuse. In M. C. Suárez-Figueroa, A. Gómez-Pérez, E. Motta, and A. Gangemi, editors, *Ontology Engineering in a Networked World*, pages 147–170. Springer, Berlin, Heidelberg, 2012.
 - [51] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, N. Schlaefter, and C. Welty. Building Watson: An Overview of the DeepQA project. *AI Magazine*, 31(3):59–79, 2010.
 - [52] W. Foddy. *Constructing Questions for Interviews and Questionnaires: Theory and Practice in Social Research*. Cambridge University Press, 1994.
 - [53] M. Fowler. *Analysis Patterns: Reusable Object Models*. Addison-Wesley, 1997.
 - [54] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

- [55] A. Gangemi. Ontology Design Patterns for Semantic Web Content. In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors, *The Semantic Web – ISWC 2005*, number 3729 in Lecture Notes in Computer Science, pages 262–276. Springer, 2005.
- [56] A. Gangemi, C. Catenacci, M. Ciaramita, and J. Lehmann. Modelling Ontology Evaluation and Validation. In Y. Sure and J. Domingue, editors, *The Semantic Web: Research and Applications. ESWC 2006*, number 4011 in Lecture Notes in Computer Science, pages 140–154, Berlin, Heidelberg, 2006. Springer.
- [57] 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.
- [58] A. Gangemi, M. Gruninger, K. Hammar, L. Lefort, V. Presutti, and A. Scherp, editors. *Proceedings of the 4th Workshop on Ontology and Semantic Web Patterns*. CEUR Workshop Proceedings, 2014.
- [59] A. Gangemi, N. Guarino, C. Masolo, and A. Oltramari. Sweetening WordNet with DOLCE. *AI magazine*, 24(3):13, 2003.
- [60] A. Gangemi and V. Presutti. Ontology Design Patterns. In S. Staab and R. Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 221–243. Springer, 2 edition, 2009.
- [61] M. Genero, G. Poels, and M. Piattini. Defining and Validating Measures for Conceptual Data Model Quality. In A. B. Pidduck, M. T. Ozsu, J. Mylopoulos, and C. C. Woo, editors, *Advanced Information Systems Engineering. CAiSE 2002*, number 2348 in Lecture Notes in Computer Science, pages 724–727, Berlin, Heidelberg, 2002. Springer.
- [62] R. L. Glass, V. Ramesh, and I. Vessey. An Analysis of Research in Computing Disciplines. *Communications of the ACM*, 47(6):89–94, 2004.
- [63] A. Gómez-Pérez, M. Fernandez-Lopez, and O. Corcho. *Ontological Engineering*. Advanced Information and Knowledge Processing. Springer, London, 2004.
- [64] B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler. OWL 2: The Next Step for OWL. *Web Semantics*, 6(4):309–322, 2008.
- [65] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

-
- [66] M. Gruninger and M. S. Fox. The Role of Competency Questions in Enterprise Engineering. In A. Rolstadås, editor, *Benchmarking — Theory and Practice*, IFIP Advances in Information and Communication Technology, pages 22–31, Boston, MA, 1995. Springer.
- [67] N. Guarino, editor. *Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy*. IOS Press, January 1998.
- [68] N. Guarino and C. Welty. Evaluating Ontological Decisions with OntoClean. *Communications of the ACM*, 45(2):61–65, 2002.
- [69] N. Guarino and C. A. Welty. An Overview of OntoClean. In S. Staab and R. Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 201–220. Springer, Berlin, Heidelberg, 2 edition, 2009.
- [70] 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.
- [71] K. Hammar. DC Proposal: Towards an ODP Quality Model. In L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, and N. Noy, editors, *The Semantic Web – ISWC 2011*, volume 2 of *Lecture Notes in Computer Science*, pages 277–284. Springer, 2011.
- [72] K. Hammar. The State of Ontology Pattern Research. In L. Niedrite, R. Strazdina, and B. Wangler, editors, *Perspectives in Business Informatics Research: Associated Workshops and Doctoral Consortium*, pages 29–37. Riga Technical University, 2011.
- [73] K. Hammar. Modular Semantic CEP for Threat Detection. In L. Villavargas, L. Sheremetov, and H.-D. Haasis, editors, *ORADM 2012: Operations Research and Data Mining Workshop Proceedings*, Cancun, Mexico, 2012. National Polytechnic Institute.
- [74] K. Hammar. Ontology Design Patterns in Use: Lessons Learnt from an Ontology Engineering Case. In E. Blomqvist, A. Gangemi, K. Hammar, and M. C. Suárez-Figueroa, editors, *Proceedings of the 3rd Workshop on Ontology Patterns*, number 929 in CEUR Workshop Proceedings, 2012.
- [75] K. Hammar. Reasoning Performance Indicators for Ontology Design Patterns. In A. Gangemi, M. Gruninger, K. Hammar, L. Lefort, V. Presutti, and A. Scherp, editors, *Proceedings of the 4th Workshop on Ontology and Semantic Web Patterns*, number 1188 in CEUR Workshop Proceedings, 2013.

- [76] K. Hammar. *Towards an Ontology Design Pattern Quality Model*. Licentiate thesis, Department of Computer and Information Science, Linköping University, 2013.
- [77] K. Hammar. Ontology Design Pattern Property Specialisation Strategies. In K. Janowicz, S. Schlobach, P. Lambrix, and E. Hyvönen, editors, *EKAU 2014: Knowledge Engineering and Knowledge Management*, number 8876 in Lecture Notes in Computer Science, pages 165–180. Springer, 2014.
- [78] K. Hammar. Ontology Design Patterns: Improving Findability and Composition. In V. Presutti, E. Blomqvist, R. Troncy, H. Sack, I. Papadakis, and A. Tordai, editors, *The Semantic Web: ESWC 2014 Satellite Events*, number 8798 in Lecture Notes in Computer Science, pages 3–13. Springer, 2014.
- [79] K. Hammar. Ontology Design Patterns in WebProtégé. In S. Villata, J. Z. Pan, and M. Dragoni, editors, *Proceedings of the ISWC 2015 Posters & Demonstrations Track*, number 1486 in CEUR Workshop Proceedings, 2015.
- [80] K. Hammar. Quality of Content Ontology Design Patterns. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns - Foundations and Applications*, volume 25 of *Studies on the Semantic Web*, pages 51–71. IOS Press, 2016.
- [81] K. Hammar. Template-Based Content ODP Instantiation. In K. Hammar, P. Hitzler, A. Krisnadhi, A. Lawrynowicz, A. G. Nuzzolese, and M. Solanki, editors, *Advances in Ontology Design and Patterns*, Studies on the Semantic Web. IOS Press, Forthcoming 2017.
- [82] K. Hammar, E. Blomqvist, D. Carral, M. Van Erp, A. Fokkens, A. Gangemi, W. R. Van Hage, P. Hitzler, K. Janowicz, N. Karima, et al. Collected Research Questions Concerning Ontology Design Patterns. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns - Foundations and Applications*, volume 25 of *Studies on the Semantic Web*, pages 189–198. IOS Press, 2016.
- [83] K. Hammar, F. Lin, and V. Tarasov. Information Reuse and Interoperability with Ontology Patterns and Linked Data. In W. Abramowicz, R. Tolksdorf, and K. Wecel, editors, *BIS 2010: Business Information Systems Workshops*, number 57 in Lecture Notes in Business Information Processing, pages 168–179. Springer, 2010.
- [84] K. Hammar and K. Sandkuhl. The State of Ontology Pattern Research: A Systematic Review of ISWC, ESWC and ASWC 2005–2009. In E. Blomqvist, V. K. Chaudhri, O. Corcho, V. Presutti, and

- K. Sandkuhl, editors, *Proceedings of the 2nd International Workshop on Ontology Patterns – WOP2010*, number 671 in CEUR Workshop Proceedings, pages 5–17, 2010.
- [85] 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.
 - [86] M. Hepp. GoodRelations: An Ontology for Describing Products and Services Offers on the Web. In A. Gangemi and J. Euzenat, editors, *Knowledge Engineering: Practice and Patterns. EKAW 2008*, number 5268 in Lecture Notes in Computer Science, pages 329–346, Berlin, Heidelberg, 2008. Springer.
 - [87] G. Hermosillo, L. Seinturier, and L. Duchien. Using Complex Event Processing for Dynamic Business Process Adaptation. In *Services Computing (SCC), 2010 IEEE International Conference on*, pages 466–473. IEEE, 2010.
 - [88] A. Hevner and S. Chatterjee. *Design Research in Information Systems: Theory and Practice*, volume 22 of *Integrated Series in Information Systems*. Springer US, 2010.
 - [89] A. R. Hevner, S. T. March, J. Park, and S. Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.
 - [90] P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, and V. Presutti, editors. *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*. Number 25 in Studies on the Semantic Web. IOS Press, 2016.
 - [91] M. Horridge, M. E. Aranguren, J. Mortensen, M. Musen, and N. F. Noy. Ontology Design Pattern Language Expressivity Requirements. In E. Blomqvist, A. Gangemi, K. Hammar, and M. C. Suárez-Figueroa, editors, *Proceedings of the 3rd Workshop on Ontology Patterns*. CEUR Workshop Proceedings, 2012.
 - [92] M. Horridge, H. Knublauch, A. Rector, R. Stevens, C. Wroe, S. Jupp, G. Moulton, N. Drummond, and S. Brandt. A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools Edition 1.3. Technical report, The University Of Manchester, 2011.
 - [93] L. Iannone, A. Rector, and R. Stevens. Embedding Knowledge Patterns into OWL. In L. Aroyo, P. Traverso, F. Ciravegna, P. Cimiano, T. Heath, E. Hyvönen, R. Mizoguchi, E. Oren, M. Sabou, and E. Simperl, editors, *The Semantic Web: Research and Applications. ESWC 2009*, number 5554 in Lecture Notes in Computer Science, pages 218–232. Springer, Berlin, Heidelberg, 2009.

- [94] ISO. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. International Standard ISO/IEC 25010:2011, International Organization for Standardization, 2011.
- [95] 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.
- [96] R. B. Johnson and A. J. Onwuegbuzie. Mixed Methods Research: A Research Paradigm Whose Time Has Come. *Educational Researcher*, 33(7):14–26, 2004.
- [97] R. B. Johnson, A. J. Onwuegbuzie, and L. A. Turner. Toward a definition of mixed methods research. *Journal of mixed methods research*, 1(2):112–133, 2007.
- [98] Y.-B. Kang, Y.-F. Li, and S. Krishnaswamy. Predicting Reasoning Performance Using Ontology Metrics. In P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, and E. Blomqvist, editors, *The Semantic Web – ISWC 2012*, number 7649 in Lecture Notes in Computer Science, pages 198–214, Berlin, Heidelberg, 2012. Springer.
- [99] N. Karima, K. Hammar, and P. Hitzler. How to Document Ontology Design Patterns. In K. Hammar, P. Hitzler, A. Krisnadhi, A. Lawrynowicz, A. G. Nuzzolese, and M. Solanki, editors, *Advances in Ontology Design and Patterns*, Studies on the Semantic Web. IOS Press, Forthcoming 2017.
- [100] C. M. Keet. Detecting and Revising Flaws in OWL Object Property Expressions. In A. ten Teije, J. Völker, S. Handschuh, H. Stuckenschmidt, M. d’Acquin, A. Nikolov, N. Aussenac-Gilles, and N. Hernandez, editors, *Knowledge Engineering and Knowledge Management. EKAW 2012*, number 7603 in Lecture Notes in Computer Science, pages 252–266. Springer, Berlin, Heidelberg, 2012.
- [101] 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, December 2004.
- [102] B. Kitchenham. Procedures for Performing Systematic Reviews. Technical report, Keele University, 2004.
- [103] G. Klyne and J. J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, W3C, 2004.

-
- [104] L. W. Lacy. *OWL: Representing Information Using the Web Ontology Language*. Trafford Publishing, January 2005.
- [105] 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.
- [106] Y. Lei, V. Uren, and E. Motta. SemSearch: A Search Engine for the Semantic Web. In S. Staab and V. Svátek, editors, *Managing Knowledge in a World of Networks: 15th International Conference, EKAW 2006, Poděbrady, Czech Republic, October 2-6, 2006. Proceedings*, volume 4248 of *Lecture Notes in Computer Science*, pages 238–245. Springer, 2006.
- [107] P. LePendu, 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 P. F. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Z. Pan, I. Horrocks, and B. Glimm, editors, *The Semantic Web – ISWC 2010*, number 6496 in *Lecture Notes in Computer Science*, pages 486–501, Berlin, Heidelberg, 2010. Springer.
- [108] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.
- [109] R. Likert. A technique for the measurement of attitudes. *Archives of psychology*, 22(140):5–55, 1932.
- [110] O. I. Lindland, G. Sindre, and A. Solvberg. Understanding Quality in Conceptual Modeling. *IEEE Software*, 11(2):42–49, 1994.
- [111] S. Lodhi and Z. Ahmed. Content Ontology Design Pattern Presentation. Master's thesis, Jönköping University, 2011.
- [112] S. Lohmann, S. Negru, F. Haag, and T. Ertl. Visualizing Ontologies with VOWL. *Semantic Web – Interoperability, Usability, Applicability*, 7(4):399–419, 2016.
- [113] A. Lozano-Tello and A. Gómez-Pérez. ONTOMETRIC: A Method to Choose the Appropriate Ontology. *Journal of Database Management*, 2(15):1–18, 2004.
- [114] D. C. Luckham and B. Frasca. Complex Event Processing in Distributed Systems. Stanford University Technical Report CSL-TR-98-754, Stanford University, August 1998.

- [115] A. Martin, R. Biddle, and J. Noble. When XP Met Outsourcing. In J. Eckstein and H. Baumeister, editors, *Extreme Programming and Agile Processes in Software Engineering. XP 2004*, number 3092 in Lecture Notes in Computer Science, pages 51–59, Berlin, Heidelberg, 2004. Springer.
- [116] 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.
- [117] J. A. McCall, P. K. Richards, and G. F. Walters. Factors in Software Quality. Volume I. Concepts and Definitions of Software Quality. Technical report, General Electric Company, Sunnyvale, CA, 1977.
- [118] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation, W3C, February 2004.
- [119] G. A. Miller. WordNet: a Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.
- [120] D. L. Moody and G. G. Shanks. What Makes a Good Data Model? Evaluating the Quality of Entity Relationship Models. In P. Loucopoulos, editor, *Entity-Relationship Approach — ER '94 Business Modelling and Re-Engineering. ER 1994*, number 881 in Lecture Notes in Computer Science, pages 94–111, Berlin, Heidelberg, 1994. Springer.
- [121] G. Morgan and L. Smircich. The Case for Qualitative Research. *Academy of Management Review*, 5(4):491–500, 1980.
- [122] A. Newell. The Knowledge Level: Presidential Address. *AI Magazine*, 2(2):1–20, 1981.
- [123] N. F. Noy and D. L. McGuinness. Ontology Development 101. Technical report, Knowledge Systems Laboratory, Stanford University, 2001.
- [124] P. Palvia, D. Leary, E. Mao, V. Midha, P. Pinjani, and A. Salam. Research Methodologies in MIS: An Update. *Communications of the Association for Information Systems*, 14:526–542, 2004.
- [125] J. Z. Pan. Resource Description Framework. In S. Staab and R. Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 71–90. Springer, 2 edition, 2009.
- [126] S. Peroni. A Simplified Agile Methodology for Ontology Development. In M. Dragoni, M. Poveda-Villalón, and E. Jimenez-Ruiz, editors, *OWL: Experiences and Directions – Reasoner Evaluation*, volume 10161 of *Lecture Notes in Computer Science*, pages 55–69. Springer, 2017.

-
- [127] A. J. Pickard. *Research Methods in Information*. Facet Publishing, London, 2 edition, 2013.
- [128] H. S. Pinto, S. Staab, C. Tempich, and Y. Sure. Distributed Engineering of Ontologies (DILIGENT). In S. Staab and H. Stuckenschmidt, editors, *Semantic Web and Peer-to-Peer: Decentralized Management and Exchange of Knowledge and Information*, pages 303–322. Springer, 2006.
- [129] S. Pinto, S. Staab, Y. Sure, and C. Tempich. OntoEdit Empowering SWAP: a Case Study in Supporting DIstributed, Loosely-Controlled and evolInG Engineering of oNTologies (DILIGENT). In C. J. Busler, J. Davies, D. Fensel, and R. Studer, editors, *The Semantic Web: Research and Applications. ESWS 2004*, number 3053 in Lecture Notes in Computer Science, pages 16–30. Springer, 2004.
- [130] L. Prechelt, B. Unger-Lamprecht, M. Philippsen, and W. F. Tichy. Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance. *IEEE Transactions on Software Engineering*, 28(6):595–606, 2002.
- [131] 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.
- [132] V. Presutti, E. Daga, A. Gangemi, and E. Blomqvist. eXtreme Design with Content Ontology Design Patterns. In E. Blomqvist, K. Sandkuhl, F. Scharffe, and V. Svatek, editors, *Proceedings of the Workshop on Ontology Patterns (WOP 2009)*, pages 83–97. CEUR Workshop Proceedings, 2009.
- [133] V. Presutti, A. Gangemi, S. David, G. A. 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 Consortium, 2007.
- [134] E. Prud’hommeaux, G. Carothers, D. Beckett, and T. Berners-Lee. RDF 1.1 Turtle – Terse RDF Triple Language. W3C Recommendation, W3C, February 2014.
- [135] A. Ranganathan and R. H. Campbell. A Middleware for Context-Aware Agents in Ubiquitous Computing Environments. In M. Endler, editor, *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, pages 143–161. Springer-Verlag New York, Inc., 2003.
- [136] L. P. Ruddin. You Can Generalize Stupid! Social Scientists, Bent Flyvbjerg, and Case Study Methodology. *Qualitative Inquiry*, 12(4):797–812, 2006.

- [137] F. B. Ruy, C. C. Reginato, V. A. Santos, R. A. Falbo, and G. Guizzardi. Ontology Engineering by Combining Ontology Patterns. In P. Johannesson, M. L. Lee, S. W. Liddle, A. L. Opdahl, and Ó. P. López, editors, *Conceptual Modeling*, number 9381 in Lecture Notes in Computer Science, pages 173–186. Springer, Cham, 2015.
- [138] T. L. Saaty. A Scaling Method for Priorities in Hierarchical Structures. *Journal of Mathematical Psychology*, 15(3):234–281, 1977.
- [139] 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.
- [140] L. Schatzman and A. L. Strauss. *Field research: Strategies for a natural sociology*. Prentice Hall, 1973.
- [141] S. Schneider, R. Torkar, and T. Gorschek. Solutions in global software engineering: A systematic literature review. *International Journal of Information Management*, 33(1):119–132, 2012.
- [142] C. B. Seaman. Qualitative Methods. In *Guide to Advanced Empirical Software Engineering*, pages 35–62. Springer, 2008.
- [143] F. Shull and R. L. Feldmann. Building Theories from Multiple Evidence Sources. In F. Shull, J. Singer, and D. I. K. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 337–364. Springer, London, 2008.
- [144] H. A. Simon. *The Sciences of the Artificial*. MIT Press, 3 edition, 1996.
- [145] 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.
- [146] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch. COVAMOF: A Framework for Modeling Variability in Software Product Families. In R. L. Nord, editor, *Software Product Lines. SPLC 2004*, number 3154 in Lecture Notes in Computer Science, pages 197–213, Berlin, Heidelberg, 2004. Springer.
- [147] 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.
- [148] B. Smith, M. Ashburner, C. Rosse, J. Bard, W. Bug, W. Ceusters, L. J. Goldberg, K. Eilbeck, A. Ireland, C. J. Mungall, T. O. Consortium, N. Leontis, P. Rocca-Serra, A. Ruttenberg, S.-A. Sansone, R. H. Scheuermann, N. Shah, P. L. Whetzel, and S. Lewis. The OBO

- Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature Biotechnology*, 25(11):1251–1255, 2007.
- [149] P. Spyns, R. Meersman, and M. Jarrar. Data modelling versus Ontology engineering. *ACM SIGMOD Record*, 31(4):12–17, 2002.
 - [150] R. D. Stacey. *Complex Responsive Processes in Organizations: Learning and knowledge creation*. Routledge, 2001.
 - [151] G. Stapleton, J. Howse, K. Taylor, A. Delaney, J. Burton, and P. Chapman. Towards Diagrammatic Ontology Patterns. In A. Gangemi, M. Gruninger, K. Hammar, L. Lefort, V. Presutti, and A. Scherp, editors, *Proceedings of the 4th Workshop on Ontology and Semantic Web Patterns*, volume 1188 of *CEUR Workshop Proceedings*, 2014.
 - [152] S. S. Stevens. On the Theory of Scales of Measurement. *Science*, 103(2684):677–680, 1946.
 - [153] R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, 25(1–2):161–197, 1998.
 - [154] M. C. Suárez-Figueroa, A. Gómez-Pérez, and M. Fernández-López. The NeOn Methodology for Ontology Engineering. In M. C. Suárez-Figueroa, A. Gómez-Pérez, E. Motta, and A. Gangemi, editors, *Ontology Engineering in a Networked World*, pages 9–34. Springer, Berlin, Heidelberg, 2012.
 - [155] M. C. Suárez-Figueroa, A. Gómez-Pérez, E. Motta, and A. Gangemi, editors. *Ontology Engineering in a Networked World*. Springer, Berlin, Heidelberg, 2012.
 - [156] J. Sun, H. Zhang, Y. F. Li, and 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.
 - [157] Y. Sure, S. Staab, and R. Studer. On-To-Knowledge Methodology (OTKM). In S. Staab and R. Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 117–132. Springer Berlin Heidelberg, 2004.
 - [158] O. Šváb-Zamazal, M. Dudáš, and V. Svátek. User-Friendly Pattern-Based Transformation of OWL Ontologies. In A. ten Teije, J. Völker, S. Handschuh, H. Stuckenschmidt, M. d’Acquin, A. Nikolov, N. Aussenac-Gilles, and N. Hernandez, editors, *Knowledge Engineering and Knowledge Management. EKAW 2012*, number 7603 in Lecture Notes in Computer Science, pages 426–429, Berlin, Heidelberg, 2012. Springer.

- [159] O. Šváb-Zamazal, V. Svátek, and L. Iannone. Pattern-Based Ontology Transformation Service Exploiting OPPL and OWL-API. In P. Cimini-ano and H. S. Pinto, editors, *Knowledge Engineering and Management by the Masses. EKAW 2010*, number 6317 in Lecture Notes in Computer Science, pages 105–119, Berlin, Heidelberg, 2010. Springer.
- [160] H. Svensson and M. Host. Introducing an Agile Process in a Software Maintenance and Evolution Organization. In *Ninth European Conference on Software Maintenance and Reengineering. CSMR 2005*, pages 256–264. IEEE, 2005.
- [161] C. Thörn. *On the Quality of Feature Models*. PhD thesis, Department of Computer and Information Science, Linköping University, 2010.
- [162] H. Tsoukas and E. Vladimirov. What is Organizational Knowledge? *Journal of Management Studies*, 38(7):973–993, 2001.
- [163] T. Tudorache, C. Nyulas, N. F. Noy, and M. A. Musen. Webprotégé: A collaborative ontology editor and knowledge acquisition tool for the web. *Semantic Web – Interoperability, Usability, Applicability*, 4(1):89–99, 2013.
- [164] J. Urbani, S. Kotoulas, J. Maassen, F. Van Harmelen, and H. Bal. OWL Reasoning with WebPIE: Calculating the Closure of 100 Billion Triples. In L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, and T. Tudorache, editors, *The Semantic Web: Research and Applications. ESWC 2010*, number 6088 in Lecture Notes in Computer Science, pages 213–227, Berlin, Heidelberg, 2010. Springer.
- [165] J. Urbani, S. Kotoulas, E. Oren, and F. Van Harmelen. Scalable Distributed Reasoning using MapReduce. In A. Bernstein, D. R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan, editors, *The Semantic Web – ISWC 2009*, number 5823 in Lecture Notes in Computer Science, pages 634–649, Berlin, Heidelberg, 2009. Springer.
- [166] 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.
- [167] 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. ACM, 2009.
- [168] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview (Second Edition). W3C Recommendation, W3C, December 2012.

- [169] R. A. Watts. *Measuring software quality*. NCC Publications, 1987.
- [170] C. Welty and N. Guarino. Supporting ontological analysis of taxonomic relationships. *Data & Knowledge Engineering*, 39(1):51–74, 2001.
- [171] D. Widdows and K. Ferraro. Semantic Vectors: a Scalable Open Source Package and Online Technology Management Application. In N. Calzolari, K. Choukri, B. Maegaard, J. Mariani, J. Odijk, S. Piperidis, and D. Tapias, editors, *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, 2008. European Language Resources Association (ELRA).
- [172] W. Yaoa, C.-H. Chub, and Z. Lia. Leveraging complex event processing for smart hospitals using RFID. *Journal of Network and Computer Applications*, 34(3):799–810, May 2011.
- [173] R. K. Yin. *Case Study Research: Design and Methods*. SAGE Publications, Inc., 4 edition, 2009.
- [174] 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.

Appendix A

ODP Quality Model Indicators

This appendix lists the quality indicators for ODPs developed in Chapter 4, grouped by category based on the aspects of the ODP that they concern (see Section 4.4.3 for details). Each indicator is listed with a suggested measurement method, a scale of measurement (per Steven’s typology [152]), and a listing of the quality characteristics that are improved or diminished by increases to the measured indicator (see Section 4.4.2 for the full listing of quality characteristics). In some cases, recommendations for indicator values and/or suggestions for ontology engineers are also included.

It should be noted that the measurement methods listed here have not been evaluated empirically, nor are they assumed to be the only suitable methods for measuring each indicator. Several indicators might in fact be measurable using many different methods (e.g., *Size* can be measured by number of axioms, entities, RDF triples, bytes when serialised into some representation, etc.). The methods listed here are thus merely an initial set of suggestions.

A.1 Documentation Indicators

DI1 Accompanying Text Description

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

Scale: Nominal (boolean)

Recommendation: Prioritise documentation fields in the following order: ODP intent, Solution description, Implementation/design consequences. In the Solution description, ensure that any uses of OWL 2 property chains are documented.

Affects characteristics: Appropriateness Recognisability (positively)

DI2 Common Pitfalls Description

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

Scale: Nominal (boolean)

Affects characteristics: User error protection (positively)

DI3 Competency Question Count

Method: Divide the number of competency questions expressed in the pattern documentation by the size (MI19) of the ODP.

Scale: Ratio (integer count)

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

DI4 Documentation Completeness

Method: Assert that each class or property within the ODP reusable OWL building block is mentioned in the ODP's accompanying textual description.

Scale: Nominal (boolean)

Affects characteristic: Usability (positively)

DI5 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)

DI6 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: Appropriateness Recognisability (positively), Learnability (positively)

DI7 Usage Example Count

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

Scale: Ratio (integer count)

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

DI8 Usage Example Illustrations

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

Scale: Ratio (integer count)

Affects characteristics: Learnability (positively)

DI9 Quality Approval Stamp

Method: Assert that documentation exists indicating that the ODP has been evaluated through some structured quality assurance process and received passing marks in that process.

Scale: Nominal (boolean)

Affects characteristics: Appropriateness recognisability (positively)

A.2 Model Indicators

MI1 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 (fraction)

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

MI2 Average Class In-Degree

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

Scale: Ratio (fraction)

Affects characteristics: Resulting performance efficiency (negatively)

MI3 Average Class Out-Degree

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

Scale: Ratio (fraction)

Affects characteristics: Resulting performance efficiency (negatively)

MI4 Axiom/Class Ratio

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

Scale: Ratio (fraction)

Affects characteristics: Analysability (positively)

MI5 Class/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 (fraction)

Affects characteristics: Usability (positively)

MI6 Class Disjointness Ratio

Method: Take the ratio of the number of inferred disjointness axioms in the ODP to the possible number of disjointness axioms given the number of classes (if C is the number of classes in the ODP, the latter is given by $\frac{C(C-1)}{2}$). If `DisjointUnion` axioms are used instead, take the ratio of all classes involved in a `DisjointUnion` axiom to the total number of classes in the ODP.

Scale: Ratio (fraction)

Affects characteristics: Resulting performance efficiency (positively)

MI7 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 [98].

Scale: Ratio (integer count)

Affects characteristics: Resulting performance efficiency (negatively)

MI8 Entity Naming Structure

Method: Assert that ODP classes and properties are named and/or labelled using a naming structure that is internally consistent and consistent with the subsumption hierarchies, and that this structure is documented.

Scale: Nominal (boolean)

Recommendation: For optimum usability, use properties named using a noun-verb-noun or noun-adjective-noun structure. Attempt to structure sub-entity names such that they extend on super-entity names, which clarifies the subsumption hierarchy. Avoid the use of homonyms.

Affects characteristics: Modifiability (negatively), Usability (positively)

MI9 Existential Quantification Count

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

Scale: Ratio (integer count)

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: Resulting performance efficiency (negatively)

MI10 Human-Readable Entity Names

Method: Assert that local IRI fragments (i.e., those parts of of ODP entity

IRIs that extend the ODP namespace) consist of terms that exist in human language(s).

Scale: Nominal (boolean)

Affects characteristics: Modifiability (negatively), Testability (positively)

MI11 Minimalism

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

Scale: Nominal (boolean)

Affects characteristics: Usability (positively)

MI12 OWL 2 EL Adherence

Method: Assert that the ODP uses only axioms that 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: Resulting performance efficiency (positively)

MI13 OWL 2 QL Adherence

Method: Assert that the ODP uses only axioms that 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: Resulting performance efficiency (positively)

MI14 OWL 2 RL Adherence

Method: Assert that the ODP uses only axioms that 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: Resulting performance efficiency (positively)

MI15 OWL Horst Adherence

Method: Assert that the ODP uses only axioms that 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: Resulting performance efficiency (positively)

MI16 Property Domain Restrictions Ratio

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

Scale: Ratio (fraction)

Recommendation: To avoid unexpected inferences, do not use multiple independently declared property domain declarations: instead, assert that a property has as its domain exactly one anonymous class, constructed as the union and/or intersection of the intended classes. Additionally, be aware that the use of domain restrictions increases *Average Class In-Degree*, an indicator associated with reduced performance.

Affects characteristics: Learnability (positively), Reusability (negatively), Resulting Performance Efficiency (negatively)

MI17 Property Range Restrictions Ratio

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

Scale: Ratio (fraction)

Recommendation: To avoid unexpected inferences, do not use multiple independently declared property range declarations: instead, assert that a property has as its range exactly one anonymous class, constructed as the union and/or intersection of the intended classes. Additionally, be aware that the use of range restrictions increases *Average Class In-Degree*, an indicator associated with reduced performance.

Affects characteristics: Learnability (positively), Reusability (negatively), Resulting Performance Efficiency (negatively)

MI18 Property Restriction Count

Method: Count the cardinality of the set of property restrictions in the associated OWL file.

Scale: Ratio (integer count)

Recommendation: The use of property restrictions (i.e., value or cardinality constraints such as `owl:someValuesFrom` or `owl:maxCardinality`) on classes can promote understanding of the intended usage of the restricted property with the class in question. However, such property restrictions also contribute to increased *Average Class Out-Degree*, which is known to affect performance efficiency negatively. Developers should thus take care to prioritise either usability or performance efficiency.

Affects characteristics: Usability (positively), Resulting Performance Efficiency (negatively)

MI19 Size

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

Scale: Ratio (integer count)

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

MI20 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 (fraction)

Affects characteristics: Usability (negatively)

MI21 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 length, that is, the number of nodes, 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 (fraction)

Affects characteristics: Usability (negatively), Resulting Performance Efficiency (negatively)

MI22 Tangledness

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

Scale: Ratio (fraction)

Affects characteristics: Usability (negatively), Compatibility (negatively), Resulting Performance Efficiency (negatively)

MI23 Terminological Cycle Count

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

Scale: Ratio (integer count)

Affects characteristics: Resulting performance efficiency (negatively)

MI24 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 (integer count)

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

A.3 In-Use Indicators

Several of the indicators listed below need to be measured by way of users performing some task or answering some survey. Consequently, the values gathered by applying the proposed methods can only be related if they are either gathered in the same context (i.e., using the same group of users), or if a sufficient number of evaluations for each ODP are carried out such that generalisability is assured through sample set size (see also Section 3.1.4). In the latter case, it would benefit the research community if the results of such evaluations were published as reusable standard references.

If the purpose of the measurement is explicitly to compare or rank a limited set of ODPs per the measured indicator(s), pairwise comparison approaches yielding ordinal scale results may be considered instead.

IUI1 Functionality Questionnaire Time

Method: Apply a questionnaire on ODP functionality and usage, and have a set of representative users answer this questionnaire, measuring the average 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 (fraction)

Affects characteristics: Learnability (negatively)

IUI2 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 average time required to perform the 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 (fraction)

Affects characteristics: Modifiability (negatively)

IUI3 Name Appropriateness

Method: Provide a representative set of users with a set of illustrations of ODP structure, and measure the average time users take to accurately select which of these illustrations correctly represents the ODP with the

name being evaluated.

Scale: Ratio (fraction)

Affects characteristics: Appropriateness Recognisability (positively)

IUI3 OntoClean Adherence

Method: Employ the OntoClean method to tag 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: Accuracy (positively)

IUI4 Semantic Distance Consistency

Method: Using representative users of target ontology functionality (typically software developers), perform pairwise ranking of the similarity of all sub-/superconcept pairs present in the ODP, such that the users respond whether they consider two sub-/superconcept pairs to display equivalent semantic distance, or whether one pair displays a higher degree of semantic distance than the other. Map responses to a suitable scale of measurement (e.g., 0 for equivalent distance, 1 for non-equivalent distance). Aggregate and average the responses across several users to account for outliers and/or demand characteristics-based biases.

Scale: Ratio (fraction)

Affects characteristics: Functional Suitability (positively)

IUI5 User-reported 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 using suitable categories (e.g., “very concrete” through “very abstract”). 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 the variation between patterns, which makes it easier for them to answer confidently (especially if they lack prior experience of ODPs). Map response categories to a suitable numeric scale of measurement and average the results.

Scale: Ratio (fraction)

Affects characteristics: Usability (negatively)

List of Figures

2.1	Ackoff's Knowledge Hierarchy.	14
2.2	Course ontology example.	17
2.3	Course data example	17
2.4	The Semantic Web layer cake	19
2.5	Context Dependant Information ODP	35
2.6	NeOn ODP typology	36
2.7	Blomqvist's ODP typology	38
2.8	XD pattern selection approach	40
2.9	XD workflow	41
2.10	MAPPER validation framework metamodel	44
3.1	Information Systems research framework by Hevner et al.	62
3.2	Research question evolution and interrelations	74
3.3	Overview of research process for answering RQ 1	76
3.4	Overview of research process for answering RQ 2	79
3.5	Overview of research process for answering RQ 3	81
3.6	Qualitative analysis coding structure used in answering RQ1	87
3.7	Qualitative analysis coding structure used in answering RQ3	87
4.1	Class in-degree and Class to property ratio distributions	115
4.2	Class out-degree and Anonymous class count distributions	117
4.3	Subsumption depth indicator variance	118
4.4	Ontology visualisation formats	124
4.5	Quality Metamodel	129
5.1	Property-oriented ODP specialisation strategy	146
5.2	Class-oriented ODP specialisation strategy	147
5.3	XDP architecture diagram	162
5.4	XDP design patterns tab	163
5.5	XDP visualisation tab	164
5.6	System Usability Scale assessment questions	164
6.1	Ontology reuse approaches decision tree	191
6.2	eXtreme Design 1.1 workflow	192

List of Tables

2.1	ISO 25010 quality in use model	49
2.2	ISO 25010 product quality model	50
3.1	IMSK project qualitative analysis: fragments and codes . . .	86
4.1	ILOG course study: ODPs used	107
4.2	ILOG course study: CQ recognition ratio	108
4.3	ILOG course study: class recognition ratio	108
4.4	ILOG course study: time to answer questions	108
4.5	ILOG course study: concreteness and usage difficulty	109
4.6	ILOG course study: time required for modelling tasks	109
4.7	ILOG course study: usability and learnability effects	109
4.8	OE survey: documentation impact on recognisability	120
4.9	OE survey: impact of illustrations on ontology learnability . .	121
4.10	OE survey: preferences on graphical illustration style	121
4.11	ODP survey: documentation impact on recognisability	122
4.12	ODP survey: documentation minimalism preferences	123
4.13	ODP survey: visualisation notation preferences	125
4.14	ODP Quality Model: quality characteristics	130
4.15	ODP Quality Model: observed indicator effects	133
4.16	ODP Quality Model: hypothesised indicator effects	134
5.1	Recall improvement for ODP search	142
5.2	ODP specialisation mapping axioms summary	144
5.3	ODP specialisation strategy use	145
5.4	Ontology specialisation strategy use	150
5.5	Specialisation strategy reasoning performance effects	152
5.6	Instantiation approaches ease-of-use comparison	158
5.7	User preferences on communications methods	160
5.8	Reported use of communications methods	161
5.9	SUS Evaluations of XDP 1.0	166
5.10	SUS Evaluations of XDP 1.1	166
5.11	XDP improvement suggestions from users	167

6.1	SSyncAHD workshops: Statements per developer	172
6.2	VALCRI project: degree to which imported entities were used	178
6.3	XD 1.1 support artefacts	189
6.4	Ontology reuse approaches and characteristics	189

Dissertations

Linköping Studies in Science and Technology

Linköping Studies in Arts and Science

Linköping Studies in Statistics

Linköping Studies in Information Science

Linköping Studies in Science and Technology

- No 14 **Anders Haraldsson:** A Program Manipulation System Based on Partial Evaluation, 1977, ISBN 91-7372-144-1.
- No 17 **Bengt Magnhagen:** Probability Based Verification of Time Margins in Digital Designs, 1977, ISBN 91-7372-157-3.
- No 18 **Mats Cedwall:** Semantisk analys av process-beskrivningar i naturligt språk, 1977, ISBN 91-7372-168-9.
- No 22 **Jaak Urmi:** A Machine Independent LISP Compiler and its Implications for Ideal Hardware, 1978, ISBN 91-7372-188-3.
- No 33 **Tore Risch:** Compilation of Multiple File Queries in a Meta-Database System 1978, ISBN 91-7372-232-4.
- No 51 **Erland Jungert:** Synthesizing Database Structures from a User Oriented Data Model, 1980, ISBN 91-7372-387-8.
- No 54 **Sture Hägglund:** Contributions to the Development of Methods and Tools for Interactive Design of Applications Software, 1980, ISBN 91-7372-404-1.
- No 55 **Pär Emanuelson:** Performance Enhancement in a Well-Structured Pattern Matcher through Partial Evaluation, 1980, ISBN 91-7372-403-3.
- No 58 **Bengt Johnsson, Bertil Andersson:** The Human-Computer Interface in Commercial Systems, 1981, ISBN 91-7372-414-9.
- No 69 **H. Jan Komorowski:** A Specification of an Abstract Prolog Machine and its Application to Partial Evaluation, 1981, ISBN 91-7372-479-3.
- No 71 **René Reboh:** Knowledge Engineering Techniques and Tools for Expert Systems, 1981, ISBN 91-7372-489-0.
- No 77 **Östen Oskarsson:** Mechanisms of Modifiability in large Software Systems, 1982, ISBN 91-7372-527-7.
- No 94 **Hans Lunell:** Code Generator Writing Systems, 1983, ISBN 91-7372-652-4.
- No 97 **Andrzej Lingas:** Advances in Minimum Weight Triangulation, 1983, ISBN 91-7372-660-5.
- No 109 **Peter Fritzson:** Towards a Distributed Programming Environment based on Incremental Compilation, 1984, ISBN 91-7372-801-2.
- No 111 **Erik Tengvald:** The Design of Expert Planning Systems. An Experimental Operations Planning System for Turning, 1984, ISBN 91-7372-805-5.
- No 155 **Christos Levcopoulos:** Heuristics for Minimum Decompositions of Polygons, 1987, ISBN 91-7870-133-3.
- No 165 **James W. Goodwin:** A Theory and System for Non-Monotonic Reasoning, 1987, ISBN 91-7870-183-X.
- No 170 **Zebo Peng:** A Formal Methodology for Automated Synthesis of VLSI Systems, 1987, ISBN 91-7870-225-9.
- No 174 **Johan Fagerström:** A Paradigm and System for Design of Distributed Systems, 1988, ISBN 91-7870-301-8.
- No 192 **Dimiter Driankov:** Towards a Many Valued Logic of Quantified Belief, 1988, ISBN 91-7870-374-3.
- No 213 **Lin Padgham:** Non-Monotonic Inheritance for an Object Oriented Knowledge Base, 1989, ISBN 91-7870-485-5.
- No 214 **Tony Larsson:** A Formal Hardware Description and Verification Method, 1989, ISBN 91-7870-517-7.
- No 221 **Michael Reiffrank:** Fundamentals and Logical Foundations of Truth Maintenance, 1989, ISBN 91-7870-546-0.
- No 239 **Jonas Löwgren:** Knowledge-Based Design Support and Discourse Management in User Interface Management Systems, 1991, ISBN 91-7870-720-X.
- No 244 **Henrik Eriksson:** Meta-Tool Support for Knowledge Acquisition, 1991, ISBN 91-7870-746-3.
- No 252 **Peter Eklund:** An Epistemic Approach to Interactive Design in Multiple Inheritance Hierarchies, 1991, ISBN 91-7870-784-6.
- No 258 **Patrick Doherty:** NML3 - A Non-Monotonic Formalism with Explicit Defaults, 1991, ISBN 91-7870-816-8.
- No 260 **Nahid Shadmehri:** Generalized Algorithmic Debugging, 1991, ISBN 91-7870-828-1.
- No 264 **Nils Dahlbäck:** Representation of Discourse-Cognitive and Computational Aspects, 1992, ISBN 91-7870-850-8.
- No 265 **Ulf Nilsson:** Abstract Interpretations and Abstract Machines: Contributions to a Methodology for the Implementation of Logic Programs, 1992, ISBN 91-7870-858-3.
- No 270 **Ralph Rönquist:** Theory and Practice of Tense-bound Object References, 1992, ISBN 91-7870-873-7.
- No 273 **Björn Fjellborg:** Pipeline Extraction for VLSI Data Path Synthesis, 1992, ISBN 91-7870-880-X.
- No 276 **Staffan Bonnier:** A Formal Basis for Horn Clause Logic with External Polymorphic Functions, 1992, ISBN 91-7870-896-6.
- No 277 **Kristian Sandahl:** Developing Knowledge Management Systems with an Active Expert Methodology, 1992, ISBN 91-7870-897-4.
- No 281 **Christer Bäckström:** Computational Complexity of Reasoning about Plans, 1992, ISBN 91-7870-979-2.
- No 292 **Mats Wirén:** Studies in Incremental Natural Language Analysis, 1992, ISBN 91-7871-027-8.
- No 297 **Mariam Kamkar:** Interprocedural Dynamic Slicing with Applications to Debugging and Testing, 1993, ISBN 91-7871-065-0.
- No 302 **Tingting Zhang:** A Study in Diagnosis Using Classification and Defaults, 1993, ISBN 91-7871-078-2.
- No 312 **Arne Jönsson:** Dialogue Management for Natural Language Interfaces - An Empirical Approach, 1993, ISBN 91-7871-110-X.
- No 338 **Simin Nadjm-Tehrani:** Reactive Systems in Physical Environments: Compositional Modelling and Framework for Verification, 1994, ISBN 91-7871-237-8.
- No 371 **Bengt Savén:** Business Models for Decision Support and Learning. A Study of Discrete-Event Manufacturing Simulation at Asea/ABB 1968-1993, 1995, ISBN 91-7871-494-X.

- No 375 **Ulf Söderman**: Conceptual Modelling of Mode Switching Physical Systems, 1995, ISBN 91-7871-516-4.
- No 383 **Andreas Kågedal**: Exploiting Groundness in Logic Programs, 1995, ISBN 91-7871-538-5.
- No 396 **George Fodor**: Ontological Control, Description, Identification and Recovery from Problematic Control Situations, 1995, ISBN 91-7871-603-9.
- No 413 **Mikael Pettersson**: Compiling Natural Semantics, 1995, ISBN 91-7871-641-1.
- No 414 **Xinli Gu**: RT Level Testability Improvement by Testability Analysis and Transformations, 1996, ISBN 91-7871-654-3.
- No 416 **Hua Shu**: Distributed Default Reasoning, 1996, ISBN 91-7871-665-9.
- No 429 **Jaime Villegas**: Simulation Supported Industrial Training from an Organisational Learning Perspective - Development and Evaluation of the SSIT Method, 1996, ISBN 91-7871-700-0.
- No 431 **Peter Jonsson**: Studies in Action Planning: Algorithms and Complexity, 1996, ISBN 91-7871-704-3.
- No 437 **Johan Boye**: Directional Types in Logic Programming, 1996, ISBN 91-7871-725-6.
- No 439 **Cecilia Sjöberg**: Activities, Voices and Arenas: Participatory Design in Practice, 1996, ISBN 91-7871-728-0.
- No 448 **Patrick Lambrix**: Part-Whole Reasoning in Description Logics, 1996, ISBN 91-7871-820-1.
- No 452 **Kjell Orsborn**: On Extensible and Object-Relational Database Technology for Finite Element Analysis Applications, 1996, ISBN 91-7871-827-9.
- No 459 **Olof Johansson**: Development Environments for Complex Product Models, 1996, ISBN 91-7871-855-4.
- No 461 **Lena Strömbäck**: User-Defined Constructions in Unification-Based Formalisms, 1997, ISBN 91-7871-857-0.
- No 462 **Lars Degerstedt**: Tabulation-based Logic Programming: A Multi-Level View of Query Answering, 1996, ISBN 91-7871-858-9.
- No 475 **Fredrik Nilsson**: Strategi och ekonomisk styrning - En studie av hur ekonomiska styrsystem utformas och används efter företagsförvärv, 1997, ISBN 91-7871-914-3.
- No 480 **Mikael Lindvall**: An Empirical Study of Requirements-Driven Impact Analysis in Object-Oriented Software Evolution, 1997, ISBN 91-7871-927-5.
- No 485 **Göran Forslund**: Opinion-Based Systems: The Cooperative Perspective on Knowledge-Based Decision Support, 1997, ISBN 91-7871-938-0.
- No 494 **Martin Sköld**: Active Database Management Systems for Monitoring and Control, 1997, ISBN 91-7219-002-7.
- No 495 **Hans Olsén**: Automatic Verification of Petri Nets in a CLP framework, 1997, ISBN 91-7219-011-6.
- No 498 **Thomas Drakengren**: Algorithms and Complexity for Temporal and Spatial Formalisms, 1997, ISBN 91-7219-019-1.
- No 502 **Jakob Axelsson**: Analysis and Synthesis of Heterogeneous Real-Time Systems, 1997, ISBN 91-7219-035-3.
- No 503 **Johan Ringström**: Compiler Generation for Data-Parallel Programming Languages from Two-Level Semantics Specifications, 1997, ISBN 91-7219-045-0.
- No 512 **Anna Moberg**: Närhet och distans - Studier av kommunikationsmönster i satellitkontor och flexibla kontor, 1997, ISBN 91-7219-119-8.
- No 520 **Mikael Ronström**: Design and Modelling of a Parallel Data Server for Telecom Applications, 1998, ISBN 91-7219-169-4.
- No 522 **Niclas Ohlsson**: Towards Effective Fault Prevention - An Empirical Study in Software Engineering, 1998, ISBN 91-7219-176-7.
- No 526 **Joachim Karlsson**: A Systematic Approach for Prioritizing Software Requirements, 1998, ISBN 91-7219-184-8.
- No 530 **Henrik Nilsson**: Declarative Debugging for Lazy Functional Languages, 1998, ISBN 91-7219-197-x.
- No 555 **Jonas Hallberg**: Timing Issues in High-Level Synthesis, 1998, ISBN 91-7219-369-7.
- No 561 **Ling Lin**: Management of 1-D Sequence Data - From Discrete to Continuous, 1999, ISBN 91-7219-402-2.
- No 563 **Eva L Ragnemalm**: Student Modelling based on Collaborative Dialogue with a Learning Companion, 1999, ISBN 91-7219-412-X.
- No 567 **Jörgen Lindström**: Does Distance matter? On geographical dispersion in organisations, 1999, ISBN 91-7219-439-1.
- No 582 **Vanja Josifovski**: Design, Implementation and Evaluation of a Distributed Mediator System for Data Integration, 1999, ISBN 91-7219-482-0.
- No 589 **Rita Kovordányi**: Modeling and Simulating Inhibitory Mechanisms in Mental Image Reinterpretation - Towards Cooperative Human-Computer Creativity, 1999, ISBN 91-7219-506-1.
- No 592 **Mikael Ericsson**: Supporting the Use of Design Knowledge - An Assessment of Commenting Agents, 1999, ISBN 91-7219-532-0.
- No 593 **Lars Karlsson**: Actions, Interactions and Narratives, 1999, ISBN 91-7219-534-7.
- No 594 **C. G. Mikael Johansson**: Social and Organizational Aspects of Requirements Engineering Methods - A practice-oriented approach, 1999, ISBN 91-7219-541-X.
- No 595 **Jörgen Hansson**: Value-Driven Multi-Class Overload Management in Real-Time Database Systems, 1999, ISBN 91-7219-542-8.
- No 596 **Niklas Hallberg**: Incorporating User Values in the Design of Information Systems and Services in the Public Sector: A Methods Approach, 1999, ISBN 91-7219-543-6.
- No 597 **Vivian Vimarlund**: An Economic Perspective on the Analysis of Impacts of Information Technology: From Case Studies in Health-Care towards General Models and Theories, 1999, ISBN 91-7219-544-4.
- No 598 **Johan Jenvald**: Methods and Tools in Computer-Supported Taskforce Training, 1999, ISBN 91-7219-547-9.
- No 607 **Magnus Merkel**: Understanding and enhancing translation by parallel text processing, 1999, ISBN 91-7219-614-9.
- No 611 **Silvia Coradeschi**: Anchoring symbols to sensory data, 1999, ISBN 91-7219-623-8.
- No 613 **Man Lin**: Analysis and Synthesis of Reactive Systems: A Generic Layered Architecture Perspective, 1999, ISBN 91-7219-630-0.
- No 618 **Jimmy Tjäder**: Systemimplementering i praktiken - En studie av logiker i fyra projekt, 1999, ISBN 91-7219-657-2.
- No 627 **Vadim Engelson**: Tools for Design, Interactive Simulation, and Visualization of Object-Oriented Models in Scientific Computing, 2000, ISBN 91-7219-709-9.

- No 637 **Esa Falkenroth:** Database Technology for Control and Simulation, 2000, ISBN 91-7219-766-8.
- No 639 **Per-Arne Persson:** Bringing Power and Knowledge Together: Information Systems Design for Autonomy and Control in Command Work, 2000, ISBN 91-7219-796-X.
- No 660 **Erik Larsson:** An Integrated System-Level Design for Testability Methodology, 2000, ISBN 91-7219-890-7.
- No 688 **Marcus Bjärelund:** Model-based Execution Monitoring, 2001, ISBN 91-7373-016-5.
- No 689 **Joakim Gustafsson:** Extending Temporal Action Logic, 2001, ISBN 91-7373-017-3.
- No 720 **Carl-Johan Petri:** Organizational Information Provision - Managing Mandatory and Discretionary Use of Information Technology, 2001, ISBN-91-7373-126-9.
- No 724 **Paul Scerri:** Designing Agents for Systems with Adjustable Autonomy, 2001, ISBN 91 7373 207 9.
- No 725 **Tim Heyer:** Semantic Inspection of Software Artifacts: From Theory to Practice, 2001, ISBN 91 7373 208 7.
- No 726 **Pär Carlshamre:** A Usability Perspective on Requirements Engineering - From Methodology to Product Development, 2001, ISBN 91 7373 212 5.
- No 732 **Juha Takkinen:** From Information Management to Task Management in Electronic Mail, 2002, ISBN 91 7373 258 3.
- No 745 **Johan Åberg:** Live Help Systems: An Approach to Intelligent Help for Web Information Systems, 2002, ISBN 91-7373-311-3.
- No 746 **Rego Granlund:** Monitoring Distributed Teamwork Training, 2002, ISBN 91-7373-312-1.
- No 757 **Henrik André-Jönsson:** Indexing Strategies for Time Series Data, 2002, ISBN 917373-346-6.
- No 747 **Anneli Hagdahl:** Development of IT-supported Interorganisational Collaboration - A Case Study in the Swedish Public Sector, 2002, ISBN 91-7373-314-8.
- No 749 **Sofie Pilemalm:** Information Technology for Non-Profit Organisations - Extended Participatory Design of an Information System for Trade Union Shop Stewards, 2002, ISBN 91-7373-318-0.
- No 765 **Stefan Holmlid:** Adapting users: Towards a theory of use quality, 2002, ISBN 91-7373-397-0.
- No 771 **Magnus Morin:** Multimedia Representations of Distributed Tactical Operations, 2002, ISBN 91-7373-421-7.
- No 772 **Pawel Pietrzak:** A Type-Based Framework for Locating Errors in Constraint Logic Programs, 2002, ISBN 91-7373-422-5.
- No 758 **Erik Berglund:** Library Communication Among Programmers Worldwide, 2002, ISBN 91-7373-349-0.
- No 774 **Choong-ho Yi:** Modelling Object-Oriented Dynamic Systems Using a Logic-Based Framework, 2002, ISBN 91-7373-424-1.
- No 779 **Mathias Broxvall:** A Study in the Computational Complexity of Temporal Reasoning, 2002, ISBN 91-7373-440-3.
- No 793 **Asmus Pandikow:** A Generic Principle for Enabling Interoperability of Structured and Object-Oriented Analysis and Design Tools, 2002, ISBN 91-7373-479-9.
- No 785 **Lars Hult:** Publika Informationstjänster. En studie av den Internetbaserade encyklopedins bruksegenskaper, 2003, ISBN 91-7373-461-6.
- No 800 **Lars Taxén:** A Framework for the Coordination of Complex Systems' Development, 2003, ISBN 91-7373-604-X.
- No 808 **Klas Gäre:** Tre perspektiv på förväntningar och förändringar i samband med införande av informationssystem, 2003, ISBN 91-7373-618-X.
- No 821 **Mikael Kindborg:** Concurrent Comics - programming of social agents by children, 2003, ISBN 91-7373-651-1.
- No 823 **Christina Ölvingson:** On Development of Information Systems with GIS Functionality in Public Health Informatics: A Requirements Engineering Approach, 2003, ISBN 91-7373-656-2.
- No 828 **Tobias Ritzau:** Memory Efficient Hard Real-Time Garbage Collection, 2003, ISBN 91-7373-666-X.
- No 833 **Paul Pop:** Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems, 2003, ISBN 91-7373-683-X.
- No 852 **Johan Moe:** Observing the Dynamic Behaviour of Large Distributed Systems to Improve Development and Testing - An Empirical Study in Software Engineering, 2003, ISBN 91-7373-779-8.
- No 867 **Erik Herzog:** An Approach to Systems Engineering Tool Data Representation and Exchange, 2004, ISBN 91-7373-929-4.
- No 872 **Aseel Berglund:** Augmenting the Remote Control: Studies in Complex Information Navigation for Digital TV, 2004, ISBN 91-7373-940-5.
- No 869 **Jo Skåmedal:** Telecommuting's Implications on Travel and Travel Patterns, 2004, ISBN 91-7373-935-9.
- No 870 **Linda Askenäs:** The Roles of IT - Studies of Organising when Implementing and Using Enterprise Systems, 2004, ISBN 91-7373-936-7.
- No 874 **Annika Flycht-Eriksson:** Design and Use of Ontologies in Information-Providing Dialogue Systems, 2004, ISBN 91-7373-947-2.
- No 873 **Peter Bunus:** Debugging Techniques for Equation-Based Languages, 2004, ISBN 91-7373-941-3.
- No 876 **Jonas Mellin:** Resource-Predictable and Efficient Monitoring of Events, 2004, ISBN 91-7373-956-1.
- No 883 **Magnus Bång:** Computing at the Speed of Paper: Ubiquitous Computing Environments for Healthcare Professionals, 2004, ISBN 91-7373-971-5.
- No 882 **Robert Eklund:** Disfluency in Swedish human-human and human-machine travel booking dialogues, 2004, ISBN 91-7373-966-9.
- No 887 **Anders Lindström:** English and other Foreign Linguistic Elements in Spoken Swedish. Studies of Productive Processes and their Modelling using Finite-State Tools, 2004, ISBN 91-7373-981-2.
- No 889 **Zhiping Wang:** Capacity-Constrained Production-inventory systems - Modelling and Analysis in both a traditional and an e-business context, 2004, ISBN 91-85295-08-6.
- No 893 **Pernilla Qvarfordt:** Eyes on Multimodal Interaction, 2004, ISBN 91-85295-30-2.
- No 910 **Magnus Kald:** In the Borderland between Strategy and Management Control - Theoretical Framework and Empirical Evidence, 2004, ISBN 91-85295-82-5.
- No 918 **Jonas Lundberg:** Shaping Electronic News: Genre Perspectives on Interaction Design, 2004, ISBN 91-85297-14-3.
- No 900 **Mattias Arvola:** Shades of use: The dynamics of interaction design for sociable use, 2004, ISBN 91-85295-42-6.
- No 920 **Luis Alejandro Cortés:** Verification and Scheduling Techniques for Real-Time Embedded Systems, 2004, ISBN 91-85297-21-6.
- No 929 **Diana Szentivanyi:** Performance Studies of Fault-Tolerant Middleware, 2005, ISBN 91-85297-58-5.

- No 933 **Mikael Cäker:** Management Accounting as Constructing and Opposing Customer Focus: Three Case Studies on Management Accounting and Customer Relations, 2005, ISBN 91-85297-64-X.
- No 937 **Jonas Kvarnström:** TALplanner and Other Extensions to Temporal Action Logic, 2005, ISBN 91-85297-75-5.
- No 938 **Bourhane Kadmiry:** Fuzzy Gain-Scheduled Visual Servoing for Unmanned Helicopter, 2005, ISBN 91-85297-76-3.
- No 945 **Gert Jervan:** Hybrid Built-In Self-Test and Test Generation Techniques for Digital Systems, 2005, ISBN: 91-85297-97-6.
- No 946 **Anders Arpteg:** Intelligent Semi-Structured Information Extraction, 2005, ISBN 91-85297-98-4.
- No 947 **Ola Angelsmark:** Constructing Algorithms for Constraint Satisfaction and Related Problems - Methods and Applications, 2005, ISBN 91-85297-99-2.
- No 963 **Calin Curescu:** Utility-based Optimisation of Resource Allocation for Wireless Networks, 2005, ISBN 91-85457-07-8.
- No 972 **Björn Johansson:** Joint Control in Dynamic Situations, 2005, ISBN 91-85457-31-0.
- No 974 **Dan Lawesson:** An Approach to Diagnosability Analysis for Interacting Finite State Systems, 2005, ISBN 91-85457-39-6.
- No 979 **Claudiu Duma:** Security and Trust Mechanisms for Groups in Distributed Services, 2005, ISBN 91-85457-54-X.
- No 983 **Sorin Manolache:** Analysis and Optimisation of Real-Time Systems with Stochastic Behaviour, 2005, ISBN 91-85457-60-4.
- No 986 **Yuxiao Zhao:** Standards-Based Application Integration for Business-to-Business Communications, 2005, ISBN 91-85457-66-3.
- No 1004 **Patrik Haslum:** Admissible Heuristics for Automated Planning, 2006, ISBN 91-85497-28-2.
- No 1005 **Aleksandra Tešanovic:** Developing Reusable and Reconfigurable Real-Time Software using Aspects and Components, 2006, ISBN 91-85497-29-0.
- No 1008 **David Dinka:** Role, Identity and Work: Extending the design and development agenda, 2006, ISBN 91-85497-42-8.
- No 1009 **Iakov Nakhimovski:** Contributions to the Modeling and Simulation of Mechanical Systems with Detailed Contact Analysis, 2006, ISBN 91-85497-43-X.
- No 1013 **Wilhelm Dahllöf:** Exact Algorithms for Exact Satisfiability Problems, 2006, ISBN 91-85523-97-6.
- No 1016 **Levon Saldamli:** PDEModelica - A High-Level Language for Modeling with Partial Differential Equations, 2006, ISBN 91-85523-84-4.
- No 1017 **Daniel Karlsson:** Verification of Component-based Embedded System Designs, 2006, ISBN 91-85523-79-8.
- No 1018 **Ioan Chisalita:** Communication and Networking Techniques for Traffic Safety Systems, 2006, ISBN 91-85523-77-1.
- No 1019 **Tarja Susi:** The Puzzle of Social Activity - The Significance of Tools in Cognition and Cooperation, 2006, ISBN 91-85523-71-2.
- No 1021 **Andrzej Bednarski:** Integrated Optimal Code Generation for Digital Signal Processors, 2006, ISBN 91-85523-69-0.
- No 1022 **Peter Aronsson:** Automatic Parallelization of Equation-Based Simulation Programs, 2006, ISBN 91-85523-68-2.
- No 1030 **Robert Nilsson:** A Mutation-based Framework for Automated Testing of Timeliness, 2006, ISBN 91-85523-35-6.
- No 1034 **Jon Edvardsson:** Techniques for Automatic Generation of Tests from Programs and Specifications, 2006, ISBN 91-85523-31-3.
- No 1035 **Vaida Jakoniene:** Integration of Biological Data, 2006, ISBN 91-85523-28-3.
- No 1045 **Genevieve Gorrell:** Generalized Hebbian Algorithms for Dimensionality Reduction in Natural Language Processing, 2006, ISBN 91-85643-88-2.
- No 1051 **Yu-Hsing Huang:** Having a New Pair of Glasses - Applying Systemic Accident Models on Road Safety, 2006, ISBN 91-85643-64-5.
- No 1054 **Åsa Hedenskog:** Perceive those things which cannot be seen - A Cognitive Systems Engineering perspective on requirements management, 2006, ISBN 91-85643-57-2.
- No 1061 **Cécile Åberg:** An Evaluation Platform for Semantic Web Technology, 2007, ISBN 91-85643-31-9.
- No 1073 **Mats Grindal:** Handling Combinatorial Explosion in Software Testing, 2007, ISBN 978-91-85715-74-9.
- No 1075 **Almut Herzog:** Usable Security Policies for Runtime Environments, 2007, ISBN 91-85715-65-7.
- No 1079 **Magnus Wahlström:** Algorithms, measures, and upper bounds for Satisfiability and related problems, 2007, ISBN 978-91-85715-55-8.
- No 1083 **Jesper Andersson:** Dynamic Software Architectures, 2007, ISBN 978-91-85715-46-6.
- No 1086 **Ulf Johansson:** Obtaining Accurate and Comprehensible Data Mining Models - An Evolutionary Approach, 2007, ISBN 978-91-85715-34-3.
- No 1089 **Traian Pop:** Analysis and Optimisation of Distributed Embedded Systems with Heterogeneous Scheduling Policies, 2007, ISBN 978-91-85715-27-5.
- No 1091 **Gustav Nordh:** Complexity Dichotomies for CSP-related Problems, 2007, ISBN 978-91-85715-20-6.
- No 1106 **Per Ola Kristensson:** Discrete and Continuous Shape Writing for Text Entry and Control, 2007, ISBN 978-91-85831-77-7.
- No 1110 **He Tan:** Aligning Biomedical Ontologies, 2007, ISBN 978-91-85831-56-2.
- No 1112 **Jessica Lindblom:** Minding the body - Interacting socially through embodied action, 2007, ISBN 978-91-85831-48-7.
- No 1113 **Pontus Wärnestål:** Dialogue Behavior Management in Conversational Recommender Systems, 2007, ISBN 978-91-85831-47-0.
- No 1120 **Thomas Gustafsson:** Management of Real-Time Data Consistency and Transient Overloads in Embedded Systems, 2007, ISBN 978-91-85831-33-3.
- No 1127 **Alexandru Andrei:** Energy Efficient and Predictable Design of Real-time Embedded Systems, 2007, ISBN 978-91-85831-06-7.
- No 1139 **Per Wikberg:** Eliciting Knowledge from Experts in Modeling of Complex Systems: Managing Variation and Interactions, 2007, ISBN 978-91-85895-66-3.
- No 1143 **Mehdi Amirijoo:** QoS Control of Real-Time Data Services under Uncertain Workload, 2007, ISBN 978-91-85895-49-6.
- No 1150 **Sanny Syberfeldt:** Optimistic Replication with Forward Conflict Resolution in Distributed Real-Time Databases, 2007, ISBN 978-91-85895-27-4.
- No 1155 **Beatrice Alenljung:** Envisioning a Future Decision Support System for Requirements Engineering - A Holistic and Human-centred Perspective, 2008, ISBN 978-91-85895-11-3.

- No 1156 **Artur Wilk**: Types for XML with Application to Xcerpt, 2008, ISBN 978-91-85895-08-3.
- No 1183 **Adrian Pop**: Integrated Model-Driven Development Environments for Equation-Based Object-Oriented Languages, 2008, ISBN 978-91-7393-895-2.
- No 1185 **Jörgen Skågeby**: Gifting Technologies - Ethnographic Studies of End-users and Social Media Sharing, 2008, ISBN 978-91-7393-892-1.
- No 1187 **Imad-Eldin Ali Abugessaisa**: Analytical tools and information-sharing methods supporting road safety organizations, 2008, ISBN 978-91-7393-887-7.
- No 1204 **H. Joe Steinhauer**: A Representation Scheme for Description and Reconstruction of Object Configurations Based on Qualitative Relations, 2008, ISBN 978-91-7393-823-5.
- No 1222 **Anders Larsson**: Test Optimization for Core-based System-on-Chip, 2008, ISBN 978-91-7393-768-9.
- No 1238 **Andreas Borg**: Processes and Models for Capacity Requirements in Telecommunication Systems, 2009, ISBN 978-91-7393-700-9.
- No 1240 **Fredrik Heintz**: DyKnow: A Stream-Based Knowledge Processing Middleware Framework, 2009, ISBN 978-91-7393-696-5.
- No 1241 **Birgitta Lindström**: Testability of Dynamic Real-Time Systems, 2009, ISBN 978-91-7393-695-8.
- No 1244 **Eva Blomqvist**: Semi-automatic Ontology Construction based on Patterns, 2009, ISBN 978-91-7393-683-5.
- No 1249 **Rogier Woltjer**: Functional Modeling of Constraint Management in Aviation Safety and Command and Control, 2009, ISBN 978-91-7393-659-0.
- No 1260 **Gianpaolo Conte**: Vision-Based Localization and Guidance for Unmanned Aerial Vehicles, 2009, ISBN 978-91-7393-603-3.
- No 1262 **AnnMarie Ericsson**: Enabling Tool Support for Formal Analysis of ECA Rules, 2009, ISBN 978-91-7393-598-2.
- No 1266 **Jiri Trnka**: Exploring Tactical Command and Control: A Role-Playing Simulation Approach, 2009, ISBN 978-91-7393-571-5.
- No 1268 **Bahlol Rahimi**: Supporting Collaborative Work through ICT - How End-users Think of and Adopt Integrated Health Information Systems, 2009, ISBN 978-91-7393-550-0.
- No 1274 **Fredrik Kuivinen**: Algorithms and Hardness Results for Some Valued CSPs, 2009, ISBN 978-91-7393-525-8.
- No 1281 **Gunnar Mathiason**: Virtual Full Replication for Scalable Distributed Real-Time Databases, 2009, ISBN 978-91-7393-503-6.
- No 1290 **Viacheslav Izosimov**: Scheduling and Optimization of Fault-Tolerant Distributed Embedded Systems, 2009, ISBN 978-91-7393-482-4.
- No 1294 **Johan Thapper**: Aspects of a Constraint Optimisation Problem, 2010, ISBN 978-91-7393-464-0.
- No 1306 **Susanna Nilsson**: Augmentation in the Wild: User Centered Development and Evaluation of Augmented Reality Applications, 2010, ISBN 978-91-7393-416-9.
- No 1313 **Christer Thörn**: On the Quality of Feature Models, 2010, ISBN 978-91-7393-394-0.
- No 1321 **Zhiyuan He**: Temperature Aware and Defect-Probability Driven Test Scheduling for System-on-Chip, 2010, ISBN 978-91-7393-378-0.
- No 1333 **David Broman**: Meta-Languages and Semantics for Equation-Based Modeling and Simulation, 2010, ISBN 978-91-7393-335-3.
- No 1337 **Alexander Siemers**: Contributions to Modelling and Visualisation of Multibody Systems Simulations with Detailed Contact Analysis, 2010, ISBN 978-91-7393-317-9.
- No 1354 **Mikael Asplund**: Disconnected Discoveries: Availability Studies in Partitioned Networks, 2010, ISBN 978-91-7393-278-3.
- No 1359 **Jana Rambusch**: Mind Games Extended: Understanding Gameplay as Situated Activity, 2010, ISBN 978-91-7393-252-3.
- No 1373 **Sonia Sangari**: Head Movement Correlates to Focus Assignment in Swedish, 2011, ISBN 978-91-7393-154-0.
- No 1374 **Jan-Erik Källhammer**: Using False Alarms when Developing Automotive Active Safety Systems, 2011, ISBN 978-91-7393-153-3.
- No 1375 **Mattias Eriksson**: Integrated Code Generation, 2011, ISBN 978-91-7393-147-2.
- No 1381 **Ola Leifler**: Affordances and Constraints of Intelligent Decision Support for Military Command and Control - Three Case Studies of Support Systems, 2011, ISBN 978-91-7393-133-5.
- No 1386 **Soheil Samii**: Quality-Driven Synthesis and Optimization of Embedded Control Systems, 2011, ISBN 978-91-7393-102-1.
- No 1419 **Erik Kuiper**: Geographic Routing in Intermittently-connected Mobile Ad Hoc Networks: Algorithms and Performance Models, 2012, ISBN 978-91-7519-981-8.
- No 1451 **Sara Stymne**: Text Harmonization Strategies for Phrase-Based Statistical Machine Translation, 2012, ISBN 978-91-7519-887-3.
- No 1455 **Alberto Montebelli**: Modeling the Role of Energy Management in Embodied Cognition, 2012, ISBN 978-91-7519-882-8.
- No 1465 **Mohammad Saifullah**: Biologically-Based Interactive Neural Network Models for Visual Attention and Object Recognition, 2012, ISBN 978-91-7519-838-5.
- No 1490 **Tomas Bengtsson**: Testing and Logic Optimization Techniques for Systems on Chip, 2012, ISBN 978-91-7519-742-5.
- No 1481 **David Byers**: Improving Software Security by Preventing Known Vulnerabilities, 2012, ISBN 978-91-7519-784-5.
- No 1496 **Tommy Färnqvist**: Exploiting Structure in CSP-related Problems, 2013, ISBN 978-91-7519-711-1.
- No 1503 **John Wilander**: Contributions to Specification, Implementation, and Execution of Secure Software, 2013, ISBN 978-91-7519-681-7.
- No 1506 **Magnus Ingmarsson**: Creating and Enabling the Useful Service Discovery Experience, 2013, ISBN 978-91-7519-662-6.
- No 1547 **Wladimir Schamai**: Model-Based Verification of Dynamic System Behavior against Requirements: Method, Language, and Tool, 2013, ISBN 978-91-7519-505-6.
- No 1551 **Henrik Svensson**: Simulations, 2013, ISBN 978-91-7519-491-2.
- No 1559 **Sergiu Rafiliiu**: Stability of Adaptive Distributed Real-Time Systems with Dynamic Resource Management, 2013, ISBN 978-91-7519-471-4.
- No 1581 **Usman Dastgeer**: Performance-aware Component Composition for GPU-based Systems, 2014, ISBN 978-91-7519-383-0.
- No 1602 **Cai Li**: Reinforcement Learning of Locomotion based on Central Pattern Generators, 2014, ISBN 978-91-7519-313-7.
- No 1652 **Roland Samlaus**: An Integrated Development Environment with Enhanced Domain-Specific

- Interactive Model Validation, 2015, ISBN 978-91-7519-090-7.
- No 1663 **Hannes Uppman:** On Some Combinatorial Optimization Problems: Algorithms and Complexity, 2015, ISBN 978-91-7519-072-3.
- No 1664 **Martin Sjölund:** Tools and Methods for Analysis, Debugging, and Performance Improvement of Equation-Based Models, 2015, ISBN 978-91-7519-071-6.
- No 1666 **Kristian Stavåker:** Contributions to Simulation of Modelica Models on Data-Parallel Multi-Core Architectures, 2015, ISBN 978-91-7519-068-6.
- No 1680 **Adrian Lifa:** Hardware/ Software Codesign of Embedded Systems with Reconfigurable and Heterogeneous Platforms, 2015, ISBN 978-91-7519-040-2.
- No 1685 **Bogdan Tanasa:** Timing Analysis of Distributed Embedded Systems with Stochastic Workload and Reliability Constraints, 2015, ISBN 978-91-7519-022-8.
- No 1691 **Håkan Warnquist:** Troubleshooting Trucks – Automated Planning and Diagnosis, 2015, ISBN 978-91-7685-993-3.
- No 1702 **Nima Aghaee:** Thermal Issues in Testing of Advanced Systems on Chip, 2015, ISBN 978-91-7685-949-0.
- No 1715 **Maria Vasilevskaya:** Security in Embedded Systems: A Model-Based Approach with Risk Metrics, 2015, ISBN 978-91-7685-917-9.
- No 1729 **Ke Jiang:** Security-Driven Design of Real-Time Embedded System, 2016, ISBN 978-91-7685-884-4.
- No 1733 **Victor Lagerkvist:** Strong Partial Clones and the Complexity of Constraint Satisfaction Problems: Limitations and Applications, 2016, ISBN 978-91-7685-856-1.
- No 1734 **Chandan Roy:** An Informed System Development Approach to Tropical Cyclone Track and Intensity Forecasting, 2016, ISBN 978-91-7685-854-7.
- No 1746 **Amir Aminifar:** Analysis, Design, and Optimization of Embedded Control Systems, 2016, ISBN 978-91-7685-826-4.
- No 1747 **Ekhioz Vergara:** Energy Modelling and Fairness for Efficient Mobile Communication, 2016, ISBN 978-91-7685-822-6.
- No 1748 **Dag Sonntag:** Chain Graphs – Interpretations, Expressiveness and Learning Algorithms, 2016, ISBN 978-91-7685-818-9.
- No 1768 **Anna Vapen:** Web Authentication using Third-Parties in Untrusted Environments, 2016, ISBN 978-91-7685-753-3.
- No 1778 **Magnus Jandinger:** On a Need to Know Basis: A Conceptual and Methodological Framework for Modelling and Analysis of Information Demand in an Enterprise Context, 2016, ISBN 978-91-7685-713-7.
- No 1798 **Rahul Hiran:** Collaborative Network Security: Targeting Wide-area Routing and Edge-network Attacks, 2016, ISBN 978-91-7685-662-8.
- No 1813 **Nicolas Melot:** Algorithms and Framework for Energy Efficient Parallel Stream Computing on Many-Core Architectures, 2016, ISBN 978-91-7685-623-9.
- No 1823 **Amy Rankin:** Making Sense of Adaptations: Resilience in High-Risk Work, 2017, ISBN 978-91-7685-596-6.
- No 1831 **Lisa Malmberg:** Building Design Capability in the Public Sector: Expanding the Horizons of Development, 2017, ISBN 978-91-7685-585-0.
- No 1851 **Marcus Bendtsen:** Gated Bayesian Networks, 2017, ISBN 978-91-7685-525-6.
- No 1852 **Zlatan Dragisic:** Completion of Ontologies and Ontology Networks, 2017, ISBN 978-91-7685-522-5.
- No 1854 **Meysam Aghighi:** Computational Complexity of some Optimization Problems in Planning, 2017, ISBN 978-91-7685-519-5.
- No 1863 **Simon Ståhlberg:** Methods for Detecting Unsolvable Planning Instances using Variable Projection, 2017, ISBN 978-91-7685-498-3.
- No 1879 **Karl Hammar:** Content Ontology Design Patterns: Qualities, Methods, and Tools, 2017, ISBN 978-91-7685-454-9.
- Linköping Studies in Arts and Science**
- No 504 **Ing-Marie Jonsson:** Social and Emotional Characteristics of Speech-based In-Vehicle Information Systems: Impact on Attitude and Driving Behaviour, 2009, ISBN 978-91-7393-478-7.
- No 586 **Fabian Segelström:** Stakeholder Engagement for Service Design: How service designers identify and communicate insights, 2013, ISBN 978-91-7519-554-4.
- No 618 **Johan Blomkvist:** Representing Future Situations of Service: Prototyping in Service Design, 2014, ISBN 978-91-7519-343-4.
- No 620 **Marcus Mast:** Human-Robot Interaction for Semi-Autonomous Assistive Robots, 2014, ISBN 978-91-7519-319-9.
- No 677 **Peter Berggren:** Assessing Shared Strategic Understanding, 2016, ISBN 978-91-7685-786-1.
- No 695 **Mattias Forsblad:** Distributed cognition in home environments: The prospective memory and cognitive practices of older adults, 2016, ISBN 978-91-7685-686-4.
- Linköping Studies in Statistics**
- No 9 **Davood Shahsavani:** Computer Experiments Designed to Explore and Approximate Complex Deterministic Models, 2008, ISBN 978-91-7393-976-8.
- No 10 **Karl Wahlén:** Roadmap for Trend Detection and Assessment of Data Quality, 2008, ISBN 978-91-7393-792-4.
- No 11 **Oleg Sysoev:** Monotonic regression for large multivariate datasets, 2010, ISBN 978-91-7393-412-1.
- No 13 **Agné Burauskaite-Harju:** Characterizing Temporal Change and Inter-Site Correlations in Daily and Sub-daily Precipitation Extremes, 2011, ISBN 978-91-7393-110-6.
- Linköping Studies in Information Science**
- No 1 **Karin Axelsson:** Metodisk systemstrukturerings- att skapa samstämmighet mellan informationssystem- arkitektur och verksamhet, 1998. ISBN-9172-19-296-8.
- No 2 **Stefan Cronholm:** Metodverktyg och användbarhet - en studie av datorstödd metodbaserad systemutveckling, 1998, ISBN-9172-19-299-2.
- No 3 **Anders Avdic:** Användare och utvecklare - om anveckling med kalkylprogram, 1999. ISBN-91-7219-606-8.
- No 4 **Owen Eriksson:** Kommunikationskvalitet hos informationssystem och affärsprocesser, 2000, ISBN 91-7219-811-7.

- No 5 **Mikael Lind:** Från system till process - kriterier för processbestämning vid verksamhetsanalys, 2001, ISBN 91-7373-067-X.
- No 6 **Ulf Melin:** Koordination och informationssystem i företag och nätverk, 2002, ISBN 91-7373-278-8.
- No 7 **Pär J. Ågerfalk:** Information Systems Actability - Understanding Information Technology as a Tool for Business Action and Communication, 2003, ISBN 91-7373-628-7.
- No 8 **Ulf Seigerroth:** Att förstå och förändra systemutvecklingsverksamheter - en taxonomi för metautveckling, 2003, ISBN 91-7373-736-4.
- No 9 **Karin Hedström:** Spår av datoriseringens värden – Effekter av IT i äldreomsorg, 2004, ISBN 91-7373-963-4.
- No 10 **Ewa Braf:** Knowledge Demanded for Action - Studies on Knowledge Mediation in Organisations, 2004, ISBN 91-85295-47-7.
- No 11 **Fredrik Karlsson:** Method Configuration method and computerized tool support, 2005, ISBN 91-85297-48-8.
- No 12 **Malin Nordström:** Styrbar systemförvaltning - Att organisera systemförvaltningsverksamhet med hjälp av effektiva förvaltningsobjekt, 2005, ISBN 91-85297-60-7.
- No 13 **Stefan Holgersson:** Yrke: POLIS - Yrkeskunskap, motivation, IT-system och andra förutsättningar för polisarbete, 2005, ISBN 91-85299-43-X.
- No 14 **Benneth Christiansson, Marie-Therese Christiansson:** Mötet mellan process och komponent - mot ett ramverk för en verksamhetsnära kravspecifikation vid anskaffning av komponentbaserade informationssystem, 2006, ISBN 91-85643-22-X.