

Linköping Studies in Science and Technology

Dissertation No. 1244

Semi-automatic Ontology Construction based on Patterns

by

Eva Blomqvist



Linköping University
INSTITUTE OF TECHNOLOGY

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

Linköping 2009

Copyright © 2009 Eva Blomqvist
Cover photograph by Stefan Nylander

ISBN 978-91-7393-683-5
ISSN 0345-7524
Printed by LiU-Tryck, Linköping 2009

*To Charles,
who always believed that anything is possible.
Hope you found all the answers.*

Abstract

This thesis aims to improve the ontology engineering process, by providing better semi-automatic support for constructing ontologies and introducing knowledge reuse through ontology patterns. The thesis introduces a typology of patterns, a general framework of pattern-based semi-automatic ontology construction called OntoCase, and provides a set of methods to solve some specific tasks within this framework. Experimental results indicate some benefits and drawbacks of both ontology patterns, in general, and semi-automatic ontology engineering using patterns, the OntoCase framework, in particular.

The general setting of this thesis is the field of information logistics, which focuses on how to provide the right information at the right moment in time to the right person or organisation, sent through the right medium. The thesis focuses on constructing enterprise ontologies to be used for structuring and retrieving information related to a certain enterprise. This means that the ontologies are quite 'light weight' in terms of logical complexity and expressiveness.

Applying ontology content design patterns within semi-automatic ontology construction, i.e. ontology learning, is a novel approach. The main contributions of this thesis are a typology of patterns together with a pattern catalogue, an overall framework for semi-automatic pattern-based ontology construction, specific methods for solving partial problems within this framework, and evaluation results showing the characteristics of ontologies constructed semi-automatically based on patterns. Results show that it is possible to improve the results of typical existing ontology learning methods by selecting and reusing patterns. OntoCase is able to introduce a general top-structure to the ontologies, and by exploiting background knowledge, the ontology is given a richer structure than when patterns are not applied.

Acknowledgements

This research was financed by, and conducted at, Jönköping University, through a cooperation agreement with the Department of Computer and Information Science at Linköping University. The initial stage of the research was conducted within the project Semantic Structuring of Components for Model-based Software Engineering of Dependable Systems (SEMCO) based on a grant from the Swedish KK-foundation (grant no. 2003/0241). Other parts of the project were conducted within the Media Information Logistics (MediaILog) project based on a grant from the foundation Carl-Olof och Jenz Hamrins Stiftelse. During the final stage some experiments were also conducted in the context of the project Lifecycle Support for Networked Ontologies (NeOn), funded by the European Commission's Sixth Framework Programme (through grant no. IST-2005-027595), due to an institutional cooperation project, Development and Evolution of Ontologies in Networked Organizations (DEON), financed by The Swedish Foundation for International Cooperation in Research and Higher Education (STINT).

My supervisors deserve my deepest thanks for providing ideas, feedback and encouragement. The main supervisor was Kurt Sandkuhl at the Information Engineering group of Jönköping University, and the secondary supervisor was Henrik Eriksson at the Human Centered Systems group of Linköping University. Very special thanks to my third supervisor Ralf-D. Kutsche, at the DIMA group of TU Berlin, for always wanting to discuss research issues and providing new perspectives. I also owe a great deal to all my co-workers at Jönköping University, especially at the Computer and Electrical Engineering department and the Centre for Evolving IT in Networked Organisations (CenIT). Special thanks to my 'office-mate' Anika Öhgren for sharing every-day concerns, and to Andreas Billig, now back at Fraunhofer ISST in Berlin, for interesting discussions and great (brain)storming.

During the past years a number of people have, in different ways, contributed to the success of this thesis. Thanks to Fabio Ciravegna at the University of Sheffield for getting me back on track when I was lost. Thanks to Aldo Gangemi and Valentina Presutti, at the STLab of CNR in Rome, for providing insight into ontology patterns. Also, thanks to Johanna Völker, at the AIFB of the University of Karlsruhe, and Claudio Baldassarre, at the Food and Agriculture Organisation (FAO) in Rome, for assisting with the final evaluations, and to all students and colleagues who contributed to this work. Particular thanks to Ludovic Jean-Louis, who provided an early implementation of the method.

Last but not least, thanks to all my family and friends who have put up with me during these five years when I spent more time in my office than anywhere else. Thanks to my mother for all support, and to my father who will always be with me in spirit. Stefan, thanks for not giving up on me!

Eva Blomqvist
Rome, February 2009

Contents

1	Introduction	1
1.1	Background	3
1.1.1	Ontologies	3
1.1.2	Ontology application case example - SEMCO	4
1.1.3	Ontology engineering	6
1.2	Motivation and problem	9
1.2.1	Manual methods for ontology engineering	9
1.2.2	Patterns	10
1.2.3	Semi-automatic ontology construction	11
1.2.4	Research questions	15
1.3	Contributions	16
1.4	Delimitations	18
1.5	Published papers	19
1.6	Organisation of the thesis	23
2	Knowledge representation through ontologies	25
2.1	Knowledge representation in ILOG	26
2.2	Basic concepts	28
2.2.1	Data, information and knowledge	28
2.2.2	The semiotic triangle	29
2.2.3	Linguistic terms, resources and methods	30
2.2.4	The concept of ontology	31
2.3	Ontology engineering	41
2.3.1	Manual ontology construction	43
2.3.2	Ontology learning	45
2.4	Chapter summary	60
2.4.1	Manual ontology construction summary	61

2.4.2	Ontology learning summary	61
3	Patterns and knowledge reuse	63
3.1	Reuse	63
3.1.1	Ontology reuse	66
3.2	Patterns	78
3.2.1	What is a pattern?	79
3.2.2	Patterns in different fields	83
3.2.3	Ontology patterns	94
3.3	Case-based reasoning	98
3.3.1	Benefits of CBR and when to use it	98
3.4	Chapter summary	99
3.4.1	Ontology reuse summary	99
3.4.2	Ontology pattern summary	101
4	Method and evaluation strategies	103
4.1	Research methods	104
4.1.1	Experimentation in computer science	105
4.1.2	A broader perspective on research method	107
4.2	Evaluation methods	111
4.2.1	Research evaluation	111
4.2.2	Result evaluation	114
4.3	Description of the research process	126
4.3.1	First iteration	127
4.3.2	Second iteration	133
5	Ontology patterns	139
5.1	Characteristics	140
5.1.1	Extraction and purpose	142
5.1.2	Structure and content	144
5.1.3	Abstraction and granularity	145
5.2	Typology of ontology patterns	146
5.2.1	Ontology application patterns	146
5.2.2	Ontology architecture patterns	147
5.2.3	Ontology design patterns	149
5.2.4	Syntactic ontology patterns	151
5.2.5	Summary of typology levels	152
5.3	Ontology content design patterns	154
5.3.1	Pattern representation	155

5.3.2	Constructing patterns	156
5.3.3	Pattern catalogue	159
5.3.4	Are patterns really useful?	161
6	Initial method, industry evaluation and experiences	173
6.1	Initial method	173
6.1.1	Pattern matching and selection	174
6.1.2	Ontology composition	176
6.2	Experiment - SEMCO	177
6.2.1	Semi-automatic ontology construction	178
6.2.2	Manual ontology construction	180
6.2.3	Evaluation	183
6.2.4	Analysis and practical consequences in the project	190
7	Semi-automatic pattern-based ontology construction	199
7.1	OntoCase overview	199
7.1.1	Semi-automatic ontology construction and CBR	200
7.1.2	The OntoCase framework	203
7.1.3	Pattern base	205
7.2	Retrieval	208
7.2.1	Text processing for ontology learning	208
7.2.2	Retrieval steps	210
7.3	Reuse	214
7.4	Future work - revise and retain	217
7.5	Pattern ranking	218
7.5.1	Concept coverage	219
7.5.2	Relation coverage	221
7.5.3	Utility measures	222
7.5.4	Rank calculation	224
7.5.5	Pattern selection	224
7.5.6	Ranking experiment	225
7.6	Notes on the OntoCase implementation	230
7.7	A small example	232
7.7.1	Ontology construction	232
7.7.2	Result and analysis	233
8	Evaluation of OntoCase	237
8.1	Extended pattern catalogue	238
8.2	SEMCO revisited	239

8.2.1	Ontology construction	239
8.2.2	Evaluation setup	241
8.2.3	Evaluation results and analysis	241
8.2.4	Summary and discussion	250
8.3	JIBSNet - the JIBS enterprise ontology	251
8.3.1	Ontology construction	252
8.3.2	Evaluation setup	252
8.3.3	Evaluation results and analysis	254
8.3.4	Summary and discussion	265
8.4	FAO - agricultural ontologies	265
8.4.1	Ontology construction	267
8.4.2	Evaluation setup	269
8.4.3	Evaluation results and analysis	270
8.4.4	Summary and discussion	276
9	Discussion and future work	277
9.1	Research evaluation	277
9.1.1	Significance	278
9.1.2	Internal validity	279
9.1.3	External validity	281
9.1.4	Objectivity and reliability	282
9.2	Ontology patterns	284
9.2.1	Benefits of ontology patterns	284
9.2.2	Pattern construction	287
9.2.3	Ontology content design patterns	288
9.2.4	Ontology architecture patterns	289
9.3	OntoCase	290
9.3.1	Pattern retrieval	290
9.3.2	Pattern reuse	292
9.3.3	Ontology revision	293
9.4	Summary of future work	294
10	Conclusions	295
	Bibliography	299
A	Ontology metamodel	323
B	Pattern catalogues	329

List of Figures	343
List of Tables	346

Chapter 1

Introduction

Ontologies are a means of formally expressing the semantics of some set of concepts. To be able to process the meaning of symbols and not only the syntactic structures of a language has been a goal of computer science research almost since the emergence of computers. Most researchers have surrendered to the complexity of the problem and have abandoned the idea of formally representing all collected and humanly understandable knowledge. Instead, focus is on specific tools, methods and approaches to solving restricted versions of this problem. An example of this is expressing the meaning of the terminology used within a certain domain or a certain enterprise, with the intention of solving some well-specified task within a community or a specific software system.

A severe problem that many companies experience today is information overload. Information is easily available in electronic formats, so instead of not being able to access information, the problem is more commonly finding the right information when you need it in a vast sea of enormous amounts of information, and combining pieces of information to form the answer to a specific question. This is true both for internal and external information, for example in a study presented by Öhgren and Sandkuhl [152] a set of companies were asked about their use and need for information. As many as 69% of the small and medium-sized companies replied that they generally received too much information, only 4% perceived a lack of information. As many as 16% of the respondents spent more than one hour each day searching for the *right* information to solve their work tasks, and 33% spent between 30 minutes and one hour on such a search. Finding solutions and support related to these problems is the general research aim

of the information logistics (ILOG) field, thus focusing on how to provide the right information at the right moment in time to the right person or organisation sent through the right medium. This problem has been defined and further described by, for example, Sandkuhl [170].

When trying to provide solutions to the ILOG problem, ontologies can assist in several ways. Ontologies may be used as a means of describing the content of the information as such. This thesis mainly focuses on ontologies for describing information content, specifically enterprise ontologies that describe the information related to a certain enterprise and needed for the enterprise's internal processes. A key issue for an enterprise that wishes to apply an ILOG system or method is then to construct an accurate and up-to-date ontology that describes their organisation and information content. Constructing ontologies has classically been a purely manual task, performed by knowledge engineers in close cooperation with domain experts.

This knowledge acquisition task is difficult and involves many problems, this is why it is sometimes denoted the 'knowledge acquisition bottleneck', as named by Feigenbaum [60]. This task is described as a bottleneck since it is very resource and time-consuming to elicit knowledge from experts, and experts often have a hard time expressing their knowledge explicitly, and some knowledge might not even be possible to express formally, so called tacit knowledge as defined by Polanyi [160] and later explained by Gourlay [86]. Thereby, constructing an enterprise ontology might seem to be an impossible task. However, the key is to restrict the problem and focus on a specific task for the ontology and tailor it to this task, not attempting to cover all aspects and views of the domain, in this case the enterprise.

Even within the limits of this problem, it is difficult to manually construct ontologies. It demands resources, is time consuming and error prone unless both domain experts and ontology engineers involved have long experience and hands-on knowledge of similar problems. Developments during the last decade attempt to change this, primarily the focus has been on more well-specified manual methods, better tools to support the process, and introducing reuse into ontology engineering. In connection with introducing new and better tools, the field of semi-automatic ontology construction has emerged, also called ontology learning (OL). The field attempts to provide a set of semi-automatic tools that will aid the ontology engineer in constructing an ontology by extracting as much information automatically as possible and then proposing this to the user, as a starting point for building the ontology manually, or as part of an iterative refinement.

The rest of this thesis focuses on methods to improve existing semi-automatic methods, specifically by processing the results of existing OL methods, algorithms and tools in order to further assist the ontology engineer in building better ontologies. The focus is on introducing knowledge reuse into semi-automatic ontology construction and using this on top of existing OL approaches. Some techniques used are ontology patterns, pattern matching and ranking, pattern specialisation and composition, combined with an overall method framework inspired by case-based reasoning as a reuse methodology.

1.1 Background

This section presents some history and background that set the stage for the discussion on research focus, open issues and research questions in section 1.2. First the ontology concept is introduced and subsequently other relevant notions are explained, together with some background on the ontology engineering process and existing methods. Our research is introduced through an application case example, in the form of an industry project scenario.

1.1.1 Ontologies

The notion of ontology is old and stems from philosophy, but ironically there is still a debate on what is actually meant by the term ontology. Sowa [182] describes the notion of ontology as follows: "*[...] it is the study of existence, of all the kinds of entities - abstract and concrete - that make up the world. It supplies the predicates of predicate calculus and the labels that fill the boxes and circles of conceptual graphs.*" In this sense different kinds of logics are the means, the tools, that are used to describe things but that which is actually described and how it is described is the actual ontology.

Another common definition of ontology comes from Gruber [87] where an ontology is described as an "*explicit specification of a conceptualization*". The definition by Gruber was later developed further by Studer et al. [191] who state that "*an ontology is a formal, explicit specification of a shared conceptualisation*". This means that an ontology should be formally represented, for example in a logical language, definitions should be explicitly stated and the conceptualisation it represents should be shared within some group of people or agents within a domain. The ontology provides a vocabulary for describing and reasoning about the concrete instances of a domain.

Ontologies can be used for many different purposes, as described by McGuinness [129]; to provide a controlled vocabulary, to customise and personalise search possibilities, to provide a structure from which to extend content, to perform word sense disambiguation, to provide interoperability support, and other similar tasks. Ontologies can also be used in a variety of applications, ranging from autonomous agents to web portals, corporate intranets or, as in our case, ILOG systems. This variety influences the features needed in an ontology, for example sometimes a simple taxonomy might be sufficient for providing a controlled vocabulary, but in other cases more advanced features, such as general axioms representing business rules are needed to enhance reasoning capabilities.

A term commonly used in ILOG is *enterprise ontology*. Uschold et al. [207] have developed and described a top-level enterprise ontology, including many basic concepts that concern enterprises and their activities. This ontology, however, cannot be used as such for our purposes, since we deal with describing the information present, or needed, for the enterprise in question. The enterprise ontology of Uschold et al. [207] does not contain information about the specific terminology of the domain of the enterprise, and only general concepts of products, services and processes. This thesis focuses on enterprise ontology construction for specific applications within enterprises. This means that enterprise ontologies should be tailored to their intended applications. An enterprise ontology would typically contain parts describing different aspects of the organisation, such as products and their features and functions, processes, organisational context, and other aspects relevant to the intended task.

1.1.2 Ontology application case example - SEMCO

The research project SEMCO (Semantic structuring of components for model-based software engineering of dependable systems) was run by the Information Engineering research group at Jönköping University between 2004 and 2007. SEMCO aimed at introducing semantic technologies into the development process of software-intensive electronic systems in order to improve efficiency in managing variants and versions of software artefacts. The project included two industrial partners active in the automotive industry domain and one research institute. One concrete application scenario was to structure and annotate all documents produced throughout the development process, to maintain connections between initial requirements and their respective influence on specifications, and to store parts of these

documents in a domain repository for future matching, retrieval and reuse. For this task it was concluded that ontologies could highly improve the structuring and retrieval of existing information, compared to existing systems and classic information retrieval (IR) technologies, as well as improve matching and analysis of new incoming information.

The scope of the experiment connected to this thesis was to develop a selected part of the enterprise ontology for one of the SEMCO industry project partners, a developer and manufacturer of complex products within the automotive supplier industry. The purpose of the ontology was more specifically to support capturing relations between development processes, organisation structures, product structures, and artefacts within the software development process. The ontology was initially limited to describing the requirements engineering process, requirements and specifications that pertain to products and parts, organisational concepts and project artefacts, thus not covering the complete development process within the first proof of concept study.

To apply the ontology in practice an application for artefact management had to be developed. Within the artefact manager tool the enterprise application ontology is used to define and store metadata and attributes of an artefact, as well as a storing a link to the artefact itself. The enterprise application ontology provides the attributes and the metadata structure, and artefacts can then be attached to it as instances, and they can be connected to instantiated attributes. When artefacts are stored in this way they can be searched, retrieved, and compared using their connection to the enterprise ontology. The ArtifactManager, described in a paper by Billig and Sandkuhl [19], was developed as a plug-in for the ontology development environment Protégé*.

A second scenario of the project was integrating feature models and enterprise ontologies by including a feature metamodel in the enterprise ontology, with the aim of using the ontology to identify similar requirements and product features in future projects. The features could be related to organisational elements in order to track responsibilities and expertise. The long term goal was to support generation of internal requirements directly from detected features in source documents, i.e. customer requirements, and the enterprise ontology and its feature model, based on semantic similarities between source documents and stored requirements of previous projects. This scenario would require some additions to the current version of the

*<http://protege.stanford.edu/>

enterprise ontology, as noted by Thörn et al. [202], and an application supporting this scenario has still not been developed but is proposed in future extensions of the SEMCO project.

1.1.3 Ontology engineering

Ontology engineering is a continuous process incorporating the complete life-cycle of an ontology; everything from the description of its intended application, requirements engineering, ontology construction, ontology reuse, to deploying the ontology in the application, maintaining, and evolving it. An important part of ontology engineering is the actual development of the ontology, the main focus of the thesis, but the development is strongly connected to the specification of the intended application, requirements engineering both for the application and the ontology, and other related tasks. When building an ontology-based application, the ontology development can be divided into a set of activities, for example as done by Fernández et al. [63] in the METHONTOLOGY methodology. They propose activities such as planning, specification, conceptualisation, formalisation, integration, implementation, evaluation, documentation, and maintenance of the ontology.

Whether applying a top-down, bottom-up or middle-out approach when developing an ontology there are several levels of abstraction that need to be considered. Just as a software system has an overall architecture, a detailed design, and a set of program code realising the design, an ontology needs to be structured and designed on several levels. A set of questions briefly summarise the problem areas on different abstraction levels that need to be addressed in order to build an ontology:

1. What is the purpose of the ontology? How is it to be applied in a software system?
2. What parts are to form the ontology? How should the architecture of the ontology be formed?
3. What should the ontology contain? What concepts, relations, and axioms?
4. How should the ontology be represented syntactically?

The questions range from considering the complete ontology on a high level, number 1, via what it should actually contain and how to arrange its parts,

to the very detailed level, number 4, of how to represent the individual concepts and relations syntactically. The first set of questions pertains to the requirements and the interface of the ontology, what operations it is to support and what information it is to provide. The second set of questions concerns the overall architecture of the ontology, what parts are to be included and how they should be arranged in order to solve the overall problem. The third set of questions addresses the detailed design and content of the ontology and finally, the fourth set of questions deals with implementation details.

Ontology development has classically been considered a manual process. When constructing ontologies for relatively static systems, where the environment and requirements do not change frequently, this might be a perfectly justified effort in order to achieve the best ontology possible for the case at hand. Such efforts were common in the 80's and 90's, when for example constructing knowledge bases for expert systems, and more recently for other kinds of applications in the field of artificial intelligence (AI), such as robotics or guidance systems for unmanned vehicles. On the other hand, the kind of ontologies discussed in this thesis are concerned with a highly uncertain and highly changeable context, i.e. an enterprise. Such a context would render the manual construction process a continuous and tedious effort, constantly requiring a high amount of resources to keep the ontology up-to-date.

When considering enterprises we can intuitively note that different people and groups may have different viewpoints and opinions of the reality to be modelled. There might, for example be a top-level management view of an organisational unit, but members of the unit might have a different, sometimes even conflicting, view of what the unit really does or consists of. Such a conflicting view of reality is an additional complexity when modelling reality. Enterprises also have to adapt to the fast changes of the market and in the environment, and to internal changes in the enterprise itself. An even more agile perspective must be taken if the ontologies are to be used on the web, where the change rate is even faster and the application environment is truly global. Due to these circumstances, recent development methodologies for constructing ontologies incorporate the complete life-cycle of the ontologies as an iterative evolution and maintenance process, also integrated with software maintenance and organisational evolution.

In addition to developments in manual ontology engineering, many semi-automatic approaches have emerged. The category of semi-automatic approaches can cover everything from simple ontology development tools, that

provide a graphical interface for ontology development instead of requiring the user to input the ontology elements expressed in the representation language, to complete tool-sets providing algorithms for extracting logical formulae from, for example text input. The semi-automatic approaches that exist may be considered as a starting point for manual ontology engineering or a refinement step integrated into a manual process, and not primarily as a separate process that 'compete' with manual methods.

In this thesis we let semi-automatic ontology construction denote the approaches that focus on automating as much as possible of the ontology construction process, thereby excluding ontology engineering tools that provide only basic support for the manual design of ontologies. The problem of supporting semi-automatic ontology construction can be divided into a set of activities and their sub-tasks as illustrated in Figure 1.1.

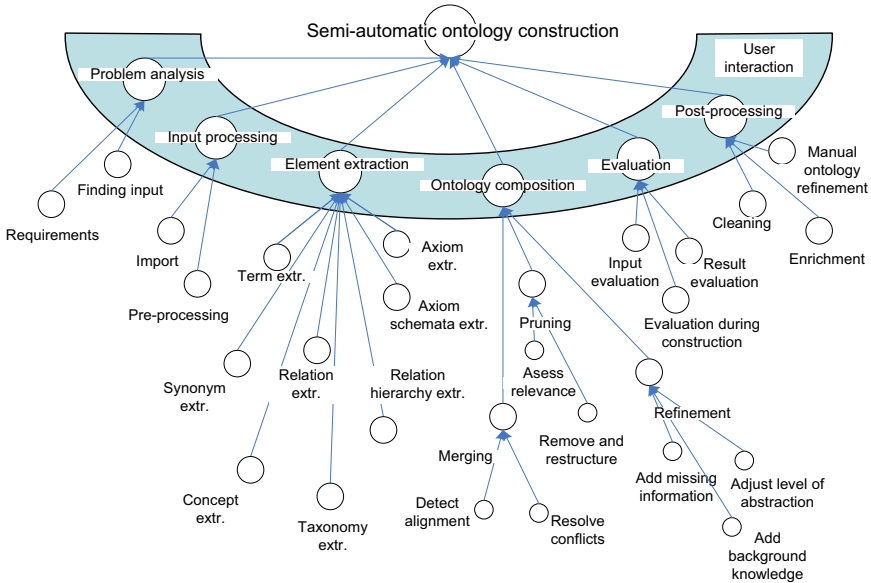


Figure 1.1: Semi-automatic ontology construction and its activities.

Sometimes semi-automatic ontology engineering is denoted ontology learning (OL). OL research aims to develop algorithms that extract ontological elements from different kinds of input, i.e. 'learning' since many approaches apply some kind of machine learning (ML), and semi-automatically compose an ontology from those elements. According to Maedche [123] and Cimiano [34] the field of OL is composed of a set of methods and algorithms through

which elements of differing expressiveness can be extracted from the input, often a text corpus, and included in the proposed ontology, or presented to the user for validation. This includes term extraction, synonym identification, concept formation, taxonomy induction, relation extraction, axiom extraction, and other methods. Most OL systems that exist today rely heavily on techniques from natural language processing (NLP) and computational linguistics for pre-processing a text corpus, and techniques from, for example text mining for extracting different elements from the text, including different kinds of relevance or confidence calculations, representing the confidence with which the elements suggested to the user have been correctly extracted. In this thesis OL will be used in a broad sense, as a synonym to semi-automatic ontology construction.

Another direction within the ontology engineering field that has received a lot of attention recently is knowledge reuse, and more specifically the use of ontology patterns. Reuse has been suggested for knowledge engineering for several decades, but it is not until recently the notion of ontology patterns has been adopted on a broader scale. Patterns have been applied in other areas, such as software engineering, where patterns are commonly carefully engineered templates that represent a consensus view of how to solve a specific problem. The templates have to be sufficiently general and abstract in order to be reusable in many cases, and they usually have to represent some notion of best practices. Patterns can then be used as templates, or even partial solutions, from which a designer can bootstrap a solution to the current problem. The underlying intuition is to be able to build better ontologies by basing new solutions on 'old' and well-proven solutions. Patterns are also commonly believed to give general guidance, to point out common problems, and improve communication between developers.

1.2 Motivation and problem

This section describes the current problems and open issues in the field of ontology engineering that motivate our research and lead to the formulation of our research questions.

1.2.1 Manual methods for ontology engineering

Whenever ontologies are constructed, the same *knowledge acquisition bottleneck* occurs as it has in all knowledge acquisition efforts since the emergence of knowledge-based systems and ontologies. It is resource and time-

consuming to construct ontologies, and both the ontology engineers and domain experts involved need to be highly skilled and preferably have knowledge about each others' perspectives on the problem. For some applications this might be worth the cost, i.e. investing a lot of time and effort in constructing a 'perfect' ontology, but in many cases this is not feasible. The benefits of having a knowledge-based, or ontology-based, application have to be balanced against the effort of constructing and maintaining the ontology in question.

Although ontologies have been built since the early days of AI, such ontologies were not common in 'standard' software systems. Currently more and more semantic applications emerge, and what started out as researchers building 'toy' ontologies has now become a movement that is about to realise the vision of the Semantic Web. The direction of much computer science research is to build better and better applications that solve new problems or solve problems better. Today however, normal web users and software developers want to build ontologies, at the same time the applications are more complex than just a few years ago. This is especially true in the Semantic Web context, where the entire web is the application area of ontology-based systems, and issues such as scalability and efficiency put higher requirements also on the ontologies. The size of the ontologies to be built can also be a problem. In response to this, distributed development methodologies, modularisation and decoupling of components, as well as methods for assisting the users in the most repetitive and time-consuming tasks of ontology engineering have emerged.

Even more crucial is the issue of ontology maintenance and evolution. In a majority of cases, the world around the application, i.e. the basis for and the context of the ontology, is not stable, it is constantly changing and evolving. To benefit from an ontology-based application, the ontology must change at the same rate as the environment. If the change rate is relatively slow then manual evolution and change tracking of the ontology might be a reasonable option, but in highly agile environments where changes occur often and are not always easy to detect and track, purely manual methods are probably not sufficient.

1.2.2 Patterns

Whether or not there are any actual benefits of using patterns, i.e. reuse, guidance, communication, and other benefits as mentioned earlier, is not clearly established. To what extent benefits actually exist when using pat-

terns has been an ongoing dispute in the software engineering field, as for example discussed by Menzies [132], Beck et al. [15], Dearden et al. [47] and Prechelt et al. [162]. The benefits of pattern usage have not yet been scientifically proven in ontology engineering, although such experiments have recently been proposed. Open issues are consequently to show that benefits of using patterns exist, and how the use of patterns affects the resulting ontologies and the process of constructing them.

Patterns are currently developed and used more or less ad-hoc in the ontology engineering field. In software engineering, patterns are common practice and are taught in most universities, consequently there are conventions for what is considered a pattern, how it is best described, and how it can be used. It remains for ontology engineering to develop methods to elicit, describe, use and maintain patterns. Some description templates have been proposed, but there is still no consensus on such templates. Neither is it clear what kinds of patterns actually exist and for what purposes they can be used. There is no accepted terminology on how to refer to different kinds of patterns and what the patterns describe.

1.2.3 Semi-automatic ontology construction

Semi-automatic ontology construction is still a relatively new field. The term ontology learning (OL) was coined at the beginning of this decade, although similar efforts can be traced back to the early days of AI, and some of the specific techniques used have been common practise in, for example NLP and information retrieval (IR) for several decades. So far semi-automatic ontology construction has largely dealt with element extraction, and approaches have mostly focused on adapting and utilising techniques from, for example NLP, computational linguistics, machine learning and text mining in order to assist users when constructing ontologies. This has resulted in quite a diverse set of methods and tools, many of which are collections of algorithms rather than actual ontology engineering tools, suitable for an end-user.

There are many approaches to term extraction for OL, based on previous research in NLP and term relevance measures in, for example IR. There are also different approaches for synonym detection, but there is much less research on actually forming concepts. Even the definition of what a concept is is not always clear. The concept formation, and especially the labelling, is commonly either left up to the ontology engineer or terms are treated as concepts and added to the ontology directly, based on term extraction from

text. Quite a few methods for relation extraction have been proposed, both taxonomical and other binary relations, but there is usually no assistance for structuring these relations, e.g. putting them on the appropriate level in the taxonomical hierarchy. Some approaches have been proposed in the past few years for extracting specific kinds of axioms and transforming restricted sets of natural language expressions into logical languages directly. Even though many of the approaches to OL originate in other areas and have been around for many years, they remain subject to research. The quality of the ontologies constructed by means of semi-automatic methods is far from perfect. Without the input and correction of an ontology engineer the ontologies are not directly usable, in many cases. One important issue is improving the quality of results, the ontology quality, of OL systems.

Another major open issue of current OL is using input other than a set of natural language texts, and combining 'clues' from many different sources when constructing an ontology semi-automatically. Many approaches have attempted to use different kinds of background knowledge in addition to the input text corpus, such as using the WordNet dictionary or the web as sources of knowledge. However, the intended scope, the level of detail and intended application of the ontology to be constructed is seldom explicitly taken into account in a semi-automatic construction process. Usually it is assumed that the user, the ontology engineer, will provide this background knowledge of the goal and intended usage of the ontology, but this might not be an easy task.

The question of what actually helps an ontology engineer is also a largely unexplored field, which in turn is somewhat related to human-computer interaction (HCI). Which methods really help? Does it help to get a list of 500 terms that are deemed important for the domain by the system, even if there is no additional explanation or assistance? Probably not, since the user in such a case still has to find the definitions of the terms in the domain in order to form concepts to put in the ontology. To develop useful user interfaces, also for existing methods, integrating the algorithms and then determining what actually helps an ontology engineer seems to be a very important issue for the future. One step in the right direction would be to strive for more transparent methods and algorithms, to let the user see what is going on. Today most OL systems do not provide any explanations to the user, for example explaining why a certain term is proposed as a concept label or why a relation was extracted and what in the text corpus this is based on.

Open issues in semi-automatic ontology construction

When combining the description of the general semi-automatic ontology construction problem and the open issues described above we can construct a map of the area, that visualises the subproblems and the unexplored areas of the field. A hierarchical division of the problem can be seen in Figure 1.2. The first problem concerns how to determine the requirements that can be used semi-automatically, and to find the appropriate input for the process. Whereas requirements have at least been treated in the case of manual ontology engineering, finding the appropriate input for the semi-automatic process is a largely untouched problem. Once the input has been found it needs to be imported and processed, in the case of text corpus input or existing ontologies, these problems have well-established solutions. The core OL problem of element extraction is not solved in general, but some of its constituents, such as term extraction, are well understood while others, such as general axiom extraction, remain largely untouched.

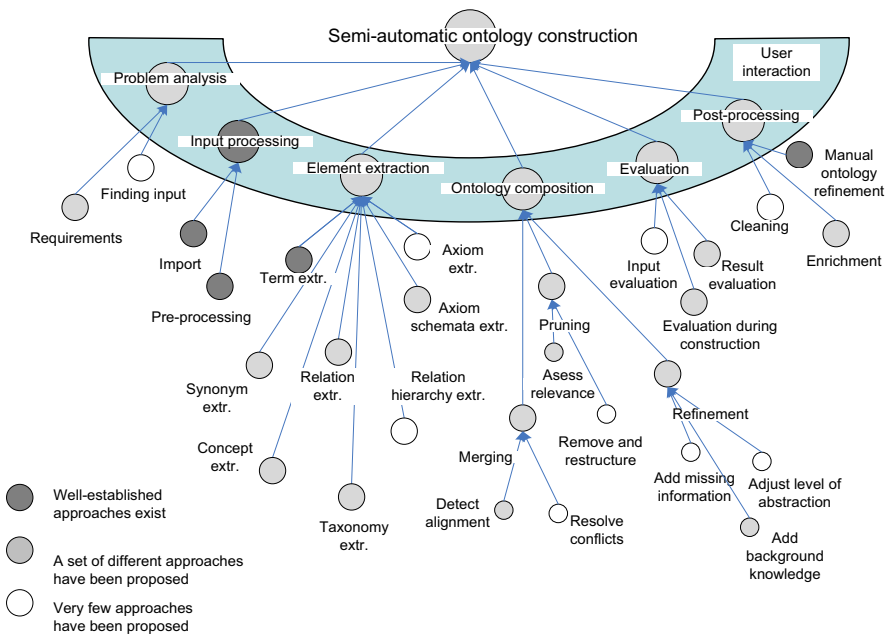


Figure 1.2: The unexplored areas of OL.

Ontology composition involves taking the results from the element extraction step and combining them into an ontology. During this process

pieces must be fitted together, and parts may have to be pruned or refined. Fitting the pieces together is similar to the problem of ontology matching and merging, and consequently there exist some approaches, even though conflict resolution is not very well researched. Pruning involves a relevance assessment of the parts to be included, and restructuring of the ontology if something is removed. Approaches have been proposed in this area for semi-automatic ontology construction, especially relevance measures of the individual 'elements' extracted are usually included in existing OL approaches. Refinement involves, for example detecting missing pieces that are essential to include, detecting missing background information, and adjusting the abstraction level of the hierarchy. There are approaches for refinement that attempt to use external knowledge sources, such as information extracted from the web, to enrich the ontology.

Evaluations can be performed during the complete process of semi-automatic ontology construction. Generally, evaluations can be divided into those that address the preliminaries of the process, the input and requirements for example, those that address intermediate results during the process, and those that evaluate the resulting ontology. There exist ontology evaluation approaches, but not many are tailored to semi-automatic ontology construction and very few can be performed automatically. Usually a step of post-processing is needed after the ontology has been constructed, either to correct detected problems, to refine the ontology further or to add missing information. Cleaning involves repairing problems discovered in evaluations, e.g. resolving inconsistencies, and enrichment involves adding missing parts. So far few approaches include semi-automatic ways to clean the ontology, while for enrichment many of the same techniques as for element extraction can be used.

A semi-automatically constructed ontology can be used as input to a manual ontology engineering process, and this can be seen as the post-processing of the ontology, or the complete set of semi-automatic methods can be part of an iterative manual methodology. Finally, the semi-automatic construction process needs to interact with the user throughout the process, since it is not completely automated. Interfaces have been proposed, but so far few have been evaluated or studied to determine the appropriateness of such interfaces for specific users and specific semi-automatic approaches.

1.2.4 Research questions

The area of ontology patterns is in need of some structure. One can imagine patterns for different purposes as mentioned in the discussion of ontology engineering in section 1.1.3, it is important to theoretically analyse what kinds of patterns exist, or should be developed, and how these can help in the development process. Apart from studying ontology patterns, in general, the overall long-term goal of this research is to reduce the effort and time required to construct enterprise application ontologies for ILOG-applications through further automation of the ontology construction process and better assistance for the ontology developer. This means solving the complete semi-automatic ontology construction problem, as described in the last section, for the specific case of enterprise ontologies for ILOG applications, which is of course not possible in one thesis.

As a starting point, we have chosen to focus on the more specific problem of ontology composition, how to use the results from element extraction algorithms to construct an ontology. Within that problem, our main focus is on merging the results and refining them, only indirectly do we address the pruning problem in the sense that it slightly overlaps with the refinement and merging problems. With this focus in mind we pose three research questions to be answered in this thesis. The questions address ontology patterns as such, using ontology patterns in combination with semi-automatic approaches, and set the focus on increased quality of the output ontology.

Research questions:

- What kinds of ontology patterns can be differentiated?
- How can ontology design patterns be used in semi-automatic ontology construction?
- How does ontology design pattern-usage in the ontology composition step affect the quality of the resulting ontologies?

The first question implies the theoretical study of the origin and background of the notion of patterns and what this means in the field of ontology engineering. There exist certain pattern approaches already, but the field lacks an overall structure and focus. Different categories of patterns need to be motivated, defined, characterised, and described. The second question entails the study of how certain types of ontology engineering design patterns can be exploited in semi-automatic ontology construction in order to improve current OL approaches. Many algorithms have been proposed that

address specific parts of the semi-automatic ontology construction problem, but how an overall framework incorporating patterns can be formed needs to be examined. Finally, the third question addresses how the ontologies constructed using a pattern-based approach to ontology composition are affected in terms of quality. The goal is to produce ontologies of higher quality, but it remains to be defined what we mean by quality and then to study how the patterns affect this notion of quality.

1.3 Contributions

A general contribution of this research is to connect two traditions in ontology engineering, the ontology learning tradition and the more manually focused ontology pattern community. This connection has not been made before, on the level of ontology design patterns, only on lower abstraction levels, such as syntactic patterns. The area of ontology patterns remains quite unstructured and only very recently have general definitions and descriptions of different kinds of patterns been proposed. One major contribution is the different categories of patterns presented in chapter 5, their structure and definitions, as well as a thorough 'state of the art' overview of patterns, as presented in chapter 3. The general theoretical comparison of semi-automatic ontology engineering and case-based reasoning (CBR), including the proposed general OntoCase framework, as illustrated in Figure 1.3, that is inspired by the CBR methodology, is a considerable theoretical contribution. The OntoCase framework consists of four phases; retrieve, reuse, revise and retain. In the retrieval phase the input is processed and used to select appropriate patterns. These patterns are then reused in the next phase to construct the ontology. In the revision phase the ontology is to be extended and revised, and finally new pattern candidates are to be discovered in the retain phase.

The more technical contributions of this thesis are the methods for matching, selecting and composing patterns based on the input from existing OL systems. These methods are part of the general OntoCase framework proposed as illustrated in Figure 1.3. The overall framework is an important contribution in itself, and the first two phases, retrieve and reuse, have been studied in detail and implemented. A set of matching criteria have been introduced and a ranking scheme has been implemented and tested for ranking patterns with respect to extracted terms and binary relations. The ranking scheme accounts for the rich structure of the extracted input

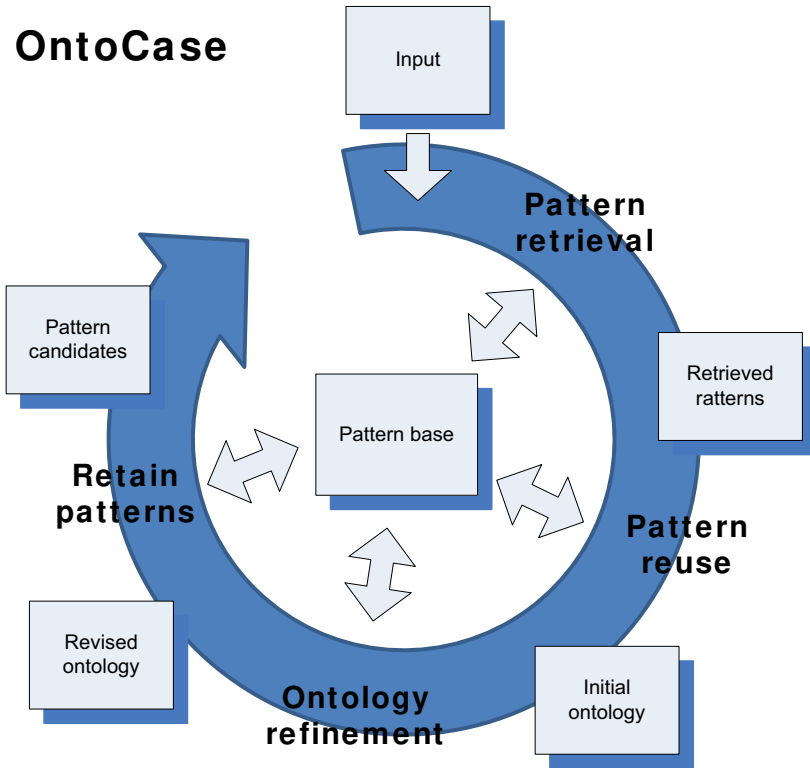


Figure 1.3: The OntoCase framework.

and the inherent differences in the abstraction level between patterns and extracted elements. The selection method is flexible and provides parameters that can be tuned by the user if desired. The pattern composition step utilises the matching results from the selection step, and the elements originally extracted from the input are also used to compose patterns through a set of heuristics.

A pattern catalogue has been developed during the process of testing and evaluating the OntoCase method. This is a catalogue of domain-specific patterns, suitable for product development companies, which contributes to the collected set of patterns within the ontology patterns community. Another contribution is the experience gleaned from the evaluation and comparison of a manual and a semi-automatic method using the first version of the OntoCase method. This has not been done extensively previously,

other than using a manually engineered ontology as a 'gold standard'. In our case the manually engineered ontology was not used as a 'gold standard' but was evaluated on its own and the benefits and shortcomings of both ontologies were then analysed together.

1.4 Delimitations

The thesis takes a specific perspective on ontologies, focusing specifically on enterprise ontologies intended to support ILOG systems within enterprises. This means that the ontologies considered are focussed mainly on describing the company in question and the information available, or to some extent the information demands of the company. This focus does not include ontologies that technically describe the information sources in the enterprise or externally, and neither does it include only top-level ontologies, such as the top level enterprise ontology mentioned in section 1.1.1. Ontologies for information structuring and retrieval purposes are generally quite 'light weight' with respect to logical complexity and expressiveness, thereby general axiom extraction and matching is not considered in this thesis.

The method presented for constructing such ontologies, called OntoCase, consists of four phases. In this thesis only the first two phases of the method, retrieve and reuse, have been implemented and tested in detail. The implementation is of a proof of concept nature whereby little consideration has been given to time and space complexity and the efficiency of the implementation. Assumptions, with respect to complexity, are explicitly mentioned in the descriptions where they are made. The first phase includes the use of existing OL systems, and consequently the OntoCase approach should not be considered as a new OL approach, but as building on top of existing OL methods. OntoCase assumes the presence of a pattern catalogue and the manual selection and input of a text corpus, or an initial seed ontology, to initiate the process. Only ontology design patterns are currently used in the method and included in the catalogue. Details of the final two phases of OntoCase are beyond the scope of this thesis.

The initial evaluation of the first attempted method was done in only one industry case and compared to the results of a specific manual method, although several different methods for evaluation were used. This evaluation was mainly used to develop new criteria for the second version of the OntoCase method, and not to evaluate the precise quality of the results of

the method. The second set of evaluations deal with the two first phases of the second version of OntoCase, and have been conducted more thoroughly using several different cases. One imitation was that only a few domains were used, and the method was not compared to all existing OL methods or manual methods. In line with the research questions this thesis focuses on output quality and not on process improvements, such as saving time and reducing user effort.

1.5 Published papers

The following peer reviewed conference and journal papers, as well as one technical report, were published previously and contain many of the results presented in this thesis. The publications are presented below and main contributions are noted, as well as the specific contribution by the author of this thesis if several authors wrote the paper.

- Blomqvist, E.: State of the Art: Patterns in Ontology Engineering. Technical Report 04:8, Jönköping University, November 2004.
 - **Contribution:** A thorough state of the art presenting existing research on both patterns in general, ontology patterns, and semi-automatic ontology construction. The report analyses the weak spots in existing research and notes a set of open research questions and issues. This report served as a basis for chapters 2 and 3.
- Blomqvist E., Sandkuhl K.: Patterns in Ontology Engineering Classification of Ontology Patterns. In: Proceedings of the 7th International Conference on Enterprise Information Systems, Miami, USA, May 2005.
 - **Contribution:** The paper proposes a framework for classifying ontology patterns, mainly based on experience and related research on software patterns. Five levels of ontology patterns are proposed based on abstraction and granularity of the patterns.
 - **Contribution of the author:** The analysis of different kinds of patterns and the development of the five levels were done by the author of this thesis. The results have been further developed, but the paper forms the basis of chapter 5.

- Blomqvist, E.: Fully Automatic Construction of Enterprise Ontologies Using Design Patterns: Initial Method and First Experiences. In: Proceedings of OTM 2005 Conferences, Ontologies, DataBases, and Applications of Semantics (ODBASE), Agia Napa, Cyprus, Oct 31-Nov 4, 2005.
 - **Contribution:** The paper presents a thorough analysis of open problems in semi-automatic ontology construction and proposes an initial method for pattern usage in semi-automatic ontology construction. The implementation tests some naive algorithms for pattern matching and selection. The description of the initial semi-automatic method in chapter 6 is based on this paper.
- Blomqvist E. and Öhgren A.: Constructing an Enterprise Ontology for an Automotive Supplier. In: Proceedings of the 12th IFAC Symposium on Information Control Problems in Manufacturing, Saint-Etienne, France, May 2006.
 - **Contribution:** The paper describes a specific industry case as part of a research project, where an enterprise ontology was constructed in two different ways in parallel. The paper presents the results of this experiment and discusses the methods used.
 - **Contribution of the author:** The semi-automatic ontology construction was conducted by the author of this thesis, who also contributed to the evaluation of the ontologies. This experiment is described in chapter 6.
- Blomqvist E., Öhgren A. and Sandkuhl K.: Ontology Construction in an Enterprise context: Comparing and Evaluating two Approaches. In: Proceedings of 8th International Conference on Enterprise Information Systems, Paphos, Cyprus, May 2006.
 - **Contribution:** This paper describes an industry case where an enterprise ontology was constructed and evaluated for a company in the automotive supplier domain. The ontology development was done in two different ways in parallel using one manual method and one semi-automatic method. The paper focuses on the evaluation and comparison of the two ontologies.
 - **Contribution of the author:** The semi-automatic ontology construction was conducted by the author of this thesis, together

with the main part of the evaluations. This paper forms the basis for the experimental results of the initial semi-automatic method described in chapter 6.

- Blomqvist E.: Semi-automatic Ontology Engineering using Patterns. In: Proceedings of the ISWC07 Doctoral Consortium, Busan, Korea, November 11-15, 2007.
 - **Contribution:** The paper proposes the overall OntoCase framework and ideas for realising the four phases. This paper has contributed to chapter 7.
- Blomqvist E.: OntoCase - A Pattern-based Ontology Construction Approach. In: Proceedings of OTM 2007: ODBASE - The 6th International Conference on Ontologies, DataBases, and Applications of Semantics, Vilamoura, Algarve, Portugal, November 25-30, 2007.
 - **Contribution:** The paper details the proposed OntoCase framework and proposes methods for solving problems in the retrieval and reuse phases, as well as outlines the future work potential of the final two phases. Some experiments on existing OL methods are presented. Chapter 7 is partly based on results presented in this paper.
- Blomqvist E., Öhgren A. and Sandkuhl K.: Comparing and Evaluating Ontology Construction in an Enterprise Context. In: Enterprise Information Systems - 8th International Conference, ICEIS 2006, Paphos, Cyprus, May 23-27, 2006, Revised Selected Papers. Lecture Notes in Business Information Processing, Manolopoulos, Y.; Filipe, J.; Constantopoulos, P.; Cordeiro, J. (Eds.), Vol. 3, 2008.
 - **Contribution:** This is an invited paper, an extended version of the publication at ICEIS2006. The paper extends the previous publication by discussing the final evaluation of the combined ontology resulting from the research project and some applications of the ontology. The main focus of the paper is on the evaluation of the ontologies and experience gleaned from those evaluations.
 - **Contribution of the author:** Apart from previous contributions to the original paper, the author of this thesis participated in the evaluations of the final ontology and contributed to the analysis of future applications of the ontology. This paper contributed to chapter 6.

- Blomqvist E. and Öhgren A.: Constructing an enterprise ontology for an automotive supplier. In: Engineering Applications of Artificial Intelligence, Vol 21, Issue 3, pp 386-397, Elsevier Ltd., 2008.
 - **Contribution:** This is an invited journal paper based on the publication at the 12th IFAC Symposium on Information Control Problems in Manufacturing 2006. The paper extends the previous publication by discussing the combination of the two ontologies, the final evaluation of the combined ontology, and some applications of the ontology within the research project.
 - **Contribution of the author:** Apart from the already published semi-automatic ontology development and evaluation, the combination of the ontologies was done jointly by the two authors of the paper and additionally the author of this thesis contributed to the descriptions of future applications of the ontology. This paper has contributed mainly to chapter 6.
- Blomqvist E.: Pattern Ranking for Semi-automatic Ontology Construction. In: Proceedings of SAC'08: Track on Semantic Web and Applications (SWA), Fortaleza, Ceará, Brazil, March 16-20, 2008.
 - **Contribution:** The paper presents the details of a ranking scheme for pattern ranking and selection within OntoCase. The method is tested on an example dataset and important features of the ranking scheme are noted, such as the possibility to match general pattern concepts to specific extracted terms. The paper has contributed to chapter 7.
- Blomqvist E.: Case-based Reasoning for Ontology Engineering. In: Proceedings of the 10th Scandinavian Conference on Artificial Intelligence, Stockholm, Sweden, May 26-28, 2008.
 - **Contribution:** This publication mainly consists of a theoretical analysis that compares semi-automatic ontology construction to Case-based Reasoning. Similarities to existing approaches are analysed and the suitability of CBR for ontology engineering is assessed. The paper has mainly contributed to chapters 3 and 7.

The following papers were coauthored by the author of this thesis during the research leading to the thesis. Some are related to ontology patterns

and the development of OntoCase, but are not presented in detail in this thesis, while others are only remotely related.

- Blomqvist E., Levashova T., Öhgren A., Sandkuhl K., Smirnov A.: Formation of Enterprise Networks for Collaborative Engineering. In: Post-conference proceedings of 3. Intl. Workshop on Collaborative Engineering, Sopron, Hungary, April 2005.
- Blomqvist E., Levashova T., Öhgren A., Sandkuhl K., Smirnov A., Tarassov V.: Configuration of Dynamic SME Supply Chains Based on Ontologies. Accepted at 2nd International Conference on Industrial Applications of Holonic and Multi-Agent Systems. Copenhagen, Denmark, August 2005.
- Thörn C., Eriksson Ö., Blomqvist E. and Sandkuhl K.: Potentials and Limits of Graph-Algorithms for Discovering Ontology Patterns. In: Proceedings of the International Conference on Intelligent Agents, Web Technology and Internet Commerce - IAWTIC'2005, Wien, Austria, November 2005.
- Albertsen T. and Blomqvist E.: Describing Ontology Applications. In: Proceedings of the 4th European Semantic Web Conference (ESWC07), Innsbruck, Austria, June 3-7 2007.
- Billig A., Blomqvist E. and Lin F.: Semantic Matching based on Enterprise Ontologies. In: Proceedings of OTM 2007: ODBASE - The 6th International Conference on Ontologies, DataBases, and Applications of Semantics, Vilamoura, Algarve, Portugal, November 25-30, 2007.
- Ricklefs M. and Blomqvist E.: Ontology-based relevance assessment - An evaluation of different semantic similarity measures. To appear in: Proceedings of OTM 2008: ODBASE - The 7th International Conference on Ontologies, DataBases, and Applications of Semantics, Monterrey, Mexico, November 9-14, 2008.

1.6 Organisation of the thesis

The following two chapters provide a more detailed background to ontologies and ontology engineering. Together with the methods and approaches that are closely related to our proposed approach, more general approaches

of knowledge reuse and patterns are presented as part of the background. Chapter 4 describes the research process and method applied in this thesis. Chapter 5 attempts to answer the first research question about the nature and existence of ontology patterns, both based on the theoretical background in chapter 3 and on our own research efforts. Chapter 6 presents the first version of the OntoCase method, and discusses the initial experiments and evaluation leading to the improved version of OntoCase. Chapter 7 addresses the second research question by describing the current version of OntoCase in detail, the proof of concept implementation, and the parts of the method that belong to future work. Next, chapter 8 discusses the evaluation of the approach and the results achieved, thereby addressing the third research question. Finally, the thesis ends with the discussion in chapter 9 and a set of conclusions in chapter 10.

Chapter 2

Knowledge representation through ontologies

Representing and formally reasoning about knowledge has been an active field of study, and of much dispute, ever since Aristotle made the first large scale attempt to represent and structure the world's knowledge. He both invented categories for structuring the knowledge and methods for reasoning about the knowledge, thus he invented what we nowadays know as logic. In a sense Aristotle thereby constructed the first formal ontology, by structuring and categorising the fields of knowledge.

The term ontology originates in ancient Greek, where *on*, genitive *ontos*, is a noun referring to the notion of 'being' and *logia*, originally derived from *logos* meaning 'word', is the term for 'science' or 'study' [1]. Ontology can thereby be interpreted as 'the study of being', and throughout history the term ontology has been used to denote the field of metaphysics devoted to the study of the nature of being. In more recent times the term ontology has additionally been adopted by the computer science field, and now commonly denotes a formal structure that explicitly specifies the concepts existing within some domain. In contrast to the original meaning, ontologies are today used in a less prescriptive way. Ontologies in computer science do not primarily deal with the *true* nature of reality, instead they are a means of describing a domain.

Ontologies are used for many purposes, and can solve a wide range of tasks. Recently the Semantic Web has emerged as a prime application field for ontologies, and the popularity of the Semantic Web initiative has

also resulted in an increased interest in ontologies. The Semantic Web was first proposed by Berners-Lee et al. [16] in their article in the *Scientific American* 2001. The Semantic Web is not a research field as such, but rather a vision for the future of the current World Wide Web where all resources are semantically described and can be accessed by artificial agents in addition to human beings. This thesis does not focus on the Semantic Web however, instead the main application field of interest is information logistics (ILOG). Nevertheless, the fields are closely related, i.e. since ILOG may also be realised in the form of web applications.

2.1 Knowledge representation in ILOG

The research presented in this thesis is part of the field of Information Logistics (ILOG), addressing the information overload problem as discussed in chapter 1. ILOG focuses on how to provide the right information at the right moment in time to the right person or organisation sent through the right medium, as described by Deiters et al. [48] and Sandkuhl [170]. ILOG research usually deals with three aspects; the content aspect, the demand aspect and the distribution aspect. The aspects are illustrated in Figure 2.1, as described by Sandkuhl [170]. The information demand can be the demand of an organisation, of a role within the organisation, or of an individual. This demand can tentatively be met by information present in different forms and in different locations. However, in order for a system to find and to process the meaning of the information content it needs to be formally represented. The correct content, possibly a combination of information content from different sources, then has to be distributed to the person or organisation presenting the demand, in an appropriate format and via an appropriate medium, to the correct place and at the correct time.

Knowledge representation, by the means of ontologies, can assist in several ways when providing ILOG solutions. Ontologies that describe services and distribution channel capabilities can assist when choosing an appropriate way of distributing a certain content, i.e. the distribution aspect. Ontologies can also be used when formally describing the information demand of a certain person, role, or organisation, which is related to the demand aspect. Demand can change in different settings and is thus context dependent. Ontologies can be used as a means for describing existing content and the meaning of information, for matching it to demands and distribution channels, or combining it with other pieces of information. In

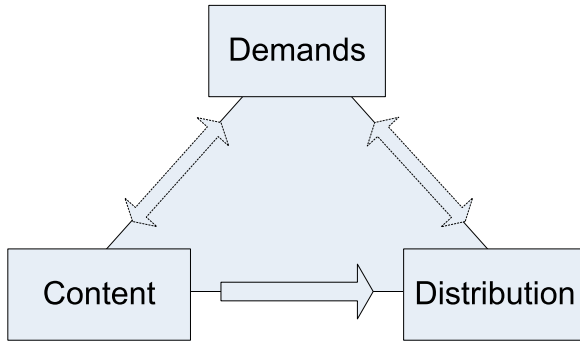


Figure 2.1: The information logistics triangle, according to Sandkuhl [170].

this thesis we mainly focus on ontologies for describing information content, specifically enterprise ontologies describing the information related to a certain enterprise, and needed for supporting its internal processes.

A key issue for an enterprise that wishes to apply an ILOG system or method is to construct an accurate and up to date ontology, describing the organisation and information content of the enterprise. Constructing ontologies has classically been a purely manual task, performed by knowledge engineers in close cooperation with domain experts. However, manually constructing ontologies is a difficult task. It is resource-demanding, time-consuming and error prone. The field of ontology learning (OL) has emerged, providing a set of semi-automatic tools that aid the ontology engineer when modelling an ontology. The goal is to automatically extract as much information as possible, and propose the extracted elements to the user as a starting point for building the ontology manually, or as a part of an iterative semi-automatic refinement process. In our research we additionally focus on introducing knowledge reuse into this process, through utilising ontology patterns in OL.

The ontology patterns and the method for semi-automatic ontology construction proposed later in this thesis is based on existing theories and previous research in the field. This chapter first includes definitions and descriptions of basic concepts, e.g. concepts related to information engineering and ILOG, different kinds of ontologies, and ontology languages. Based on these basic concepts, ontology engineering is presented in detail, specifically semi-automatic ontology construction, ontology learning, is discussed. When presenting the OL field related approaches and methods are treated, such as methods for ontology search, selection, matching, and integration.

2.2 Basic concepts

This section defines and explains basic terms in order to introduce the reader to our research viewpoint, and assist further reading.

2.2.1 Data, information and knowledge

The terms knowledge and information are used in numerous places in this thesis, in the introductory chapter they were used without further explanation. The distinction between knowledge and information is sometimes not explicitly defined, and in some literature it is even completely ignored. Commonly, the difference between data, information, and knowledge is defined as an increasing level of interpretation and internalisation within some agent.

The knowledge management (KM) community reserve the term knowledge for information that has been internalised by a person, as summarised by for example Tsoukas and Vladimirou [203] and Stacey [186]. This means that the information has been put into a context, and has some impact on that person's thoughts, values, reasoning processes and actions. Data in this sense is the raw information that can exist without any human involvement, it can easily be represented, for example in a computer. To transform data into information some human processing, i.e. reasoning, is required in order to put the data into a context and give it meaning.

However, most researchers in technical fields state that information can also be stored in some formal representation, e.g. in a computer, while knowledge on the other hand is something that can only exist within a human mind. A fourth level is sometimes proposed, denoted understanding or wisdom. This level is more abstract and used to denote the state when knowledge is truly internalised in a human mind and can be used for decision making and reasoning, almost without effort.

Another tradition, originating from the field of artificial intelligence (AI), has slightly different definitions of data, information and knowledge. The term knowledge representation (KR) is commonly used in this tradition, implying that knowledge can in fact be represented and stored outside a human mind. In a paper from 1981 Allen Newell defines knowledge as "whatever can be ascribed to an agent, such that its behaviour can be computed according to the principle of rationality" [142]. This is similar to the notion of knowledge as defined in KM, aside from the fact that in AI also an artificial agent, not only a human, may hold and use knowledge.

In our research we adopt the view from the AI field, stating that both data, information, and knowledge can be represented formally and processed by artificial agents or software systems. Data is a set of symbols with no inherent meaning, other than being symbols of some language. Information is the interpretation of those symbols in some context, the data after it has been processed. The number 30 is a data item, it consists of the two symbols 3 and 0. However, when that data item is found in the database field interpreted by a system as the age of the persons stored in that table, it has become information. The difference between information and knowledge is more subtle, knowledge is information that is used for some purpose. In our view the information represented in, or connected to, an ontology becomes knowledge when the information is reasoned with and applied in a system or application. The ontology in itself, without any surrounding context, is a representation of information.

2.2.2 The semiotic triangle

The study of semiotics is closely related to the notion of ontology, since semiotics is the study of signs and other carriers of meaning [2] and ontologies attempt to do exactly that, represent meaning. Semiotics is commonly explained through a triangle of meaning. There are many variations of this triangle in literature, but one of the most commonly used originate from a book by Ogden and Richards [149]. An illustration of the triangle can be seen in Figure 2.2. The triangle signifies the connection between an object or abstract entity in reality, the referent, and a symbol that represents this entity. The connection is indirect, via the thought, or reference, provided by some interpreter, usually a human being.

The concepts and relations expressed through symbols of an ontology representation language, usually a logical language, intend to represent some, abstract or concrete, entities in the world. In order to interpret the symbols, and 'make the connection' between the symbols and real-world entities and events, some reasoning mechanism is needed. In the case of formal ontologies this process is provided by reasoning engines, and other application software using the ontologies. The main challenge when engineering ontologies is finding the correct representation symbols to represent the meaning of the referred real-world entities. The semantic representation has to be accurate, in order to later 'recreate' the intended meaning. This can be viewed as the main objective of ontology learning, to automate the process of finding the correct symbols to represent a real-world entity.

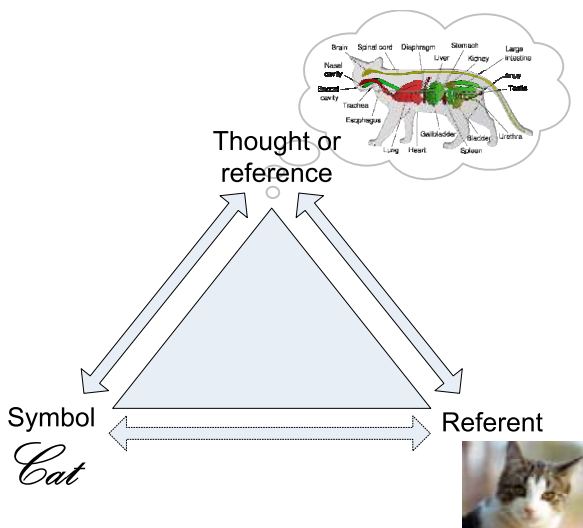


Figure 2.2: An illustration of the semiotic triangle.

2.2.3 Linguistic terms, resources and methods

References are made to concepts originating in natural language processing (NLP), computational linguistics, and related fields throughout this thesis. For instance the notions of *word* and *term* are used frequently. *Words*, as described by for example Jurafsky and Martin [107], are the basic building blocks of all natural languages, usually represented by clearly distinguishable syntactic entities, such as a sequence of letters written together and separated from other words through spaces. A word conveys some meaning, and can have a grammatical function in a sentence. A *term* can consist of one or more words and is a lexical realisation of a concept, in the context of OL explained by Cimiano [34]. A *concept* is an abstract notion that only truly exists in the minds of human beings, but concepts can be modelled and informally represented as, for example a set of terms, a definition in natural language, i.e. an intensional definition, or a set of examples of concept instantiations, i.e. an extensional definition.

Linguistic resources can be very useful in OL, as we shall see later. One popular linguistic resource is the lexical database WordNet, as described by Fellbaum et al. [5]. The terms present in WordNet are arranged in 'synsets', i.e. sets of term synonyms that are interchangeable in some context, which can be seen as representing some concept. Synsets are related through a

number of different semantic relations. One of the most important being the hypernym/hyponym relation, where a term x being a hypernym of another term y is interpreted as y being 'a kind of' x . Hypernym/hyponym relations are present between all noun and verb synsets.

Further details of natural language processing can be found in reference literature such as Jurafsky and Martin [107]. Some useful methods are however interesting to mention, since these will be briefly discussed later in this thesis. When parsing a text *tokenization* is one subtask. Tokenization means to divide a string of characters into 'tokens', e.g. words. When words have been identified *stemming* can be applied, meaning that the word is reduced from an inflected form to the stem, e.g. both 'fisher' and 'fishing' would be reduced to the stem 'fish'. Stop-word removal is a technique to discard irrelevant words from a text, based on a list of so called 'stop words', i.e. to remove common words such as 'the', 'a', and 'and' that do not carry any meaning in themselves.

2.2.4 The concept of ontology

The notion of 'ontology' has a long history in philosophy as noted at the beginning of this chapter. This thesis deals only with ontologies as defined in the field of computer science. Even in this field there are numerous definitions of the term ontology* in the literature. Among the most commonly used definitions we find:

1. "*An ontology is an explicit specification of a conceptualisation.*" [87]
2. "*ontology: (sense 1) a logical theory which gives an explicit, partial account of a conceptualization; (sense 2) synonym of conceptualization.*" [90]
3. "*An ontology is a formal, explicit specification of a shared conceptualisation.*" [191]
4. "*An ontology is a logical theory accounting for the intended meaning of a formal vocabulary, i.e., its ontological commitment to a particular conceptualisation of the world. The intended models of a logical*

*A remark on the term ontology: Some researchers use the word with a capital O and only in the singular form. This is the classical way of using the word, when referring to Ontology as an area in philosophy. In this thesis ontologies will be discussed according to the definitions above, and the word will be used with a lower case 'o' and the appropriate inflected form.

language using such a vocabulary are constrained by its ontological commitment. An ontology indirectly reflects this commitment by approximating these intended models.” [89]

The above definitions are only a brief selection of definitions proposed by different researchers. The above definitions are expressed in natural language but, in general, definitions range from very informal to precise and mathematical. Mathematically formalised definitions have been proposed by for example Maedche [123] and Cimiano [34]. Mathematical definitions connect the general notion of ontology to a specific mathematical representation, and are thereby suited for direct use by ontology-based systems or ontology engineering tools.

As we can note there is not one single definition of the concept 'ontology'. Guarino and Garetta [90] already in 1995 attempted to clarify the meaning of the definitions existing at that time. Their proposal was the two senses of 'ontology' described in definition number two above. Their main concern was with the notion of 'conceptualization' included in Gruber's definition [87], number one above. Resulting in the relaxation of that definition to Guarino and Garetta's 'sense 1', since a conceptualization represented by a logical theory will always be incomplete in some sense. While these definitions focus on general abstract notions, such as conceptualizations, other definitions exist that are more concrete in nature, treating an ontology as a logical theory and listing the basic elements that can constitute such a theory. Some definitions, especially in the OL field, do not distinguish between concepts and their realisation as terms. Cimiano [34] separates these two notions, the ontology and its 'lexicon', i.e. the lexical realisations of the concepts in the ontology.

From the set of existing definitions we derive that ontologies commonly consist of concepts, relations, axioms, and other constraints. The aim is to create a shared vocabulary. Ontologies usually contain a hierarchical ordering of the concepts, denoted taxonomy or subsumption hierarchy. Ontologies can be used to create a knowledge base through instantiating the concepts of the ontology, sometimes the term ontology is used to also denote this extended structure, i.e. ontology and instances. An ontology has to be computer processable to be used in an application, thereby it has to be represented using some formal representation language.

An ontology belongs to a specific *domain* of knowledge. The scope of the ontology concentrates on definitions of a certain domain, although sometimes the domain can be very broad. The domain can be an indus-

try domain, an enterprise, a research field, or any other restricted set of knowledge, whether abstract, concrete or even imagined. An ontology is usually constructed with a certain *task* in mind, this task focus restricts the content and structure of the ontology. An ontology can, for example, be used with a reasoning engine to classify instances, check consistency of facts, or answer queries. On the other hand there can be different kinds of tasks, where complex reasoning is not the main focus, such as annotating information, or acting as a user interface for structured document browsing. The nature or the task imposes requirements on the content and structure of the ontology.

In this thesis we will not restrict ourselves to a specific ontology representation language or a mathematical definition of the ontology concept. The general definition of an ontology as explained by Studer et al. [191], shown in Definition 1, will be used. It is an extension of the definition by Gruber [87] and conforms to the 'first sense' of the ontology concept as explained by Guarino and Giaretta [90].

Definition 1. *An ontology is a formal, explicit specification of a shared conceptualisation.*

We view the ontology as a formally represented version of such an explicit specification, requiring it to be represented in some computer processable manner. The shared conceptualisation the ontology represents is not a complete conceptualisation of the world, as noted by Guarino and Giaretta [90], but it is restricted to the necessary definitions in order to solve some specific task.

An ontology can be seen as a set of logical formulae, logical axioms, but in this thesis we choose to view an ontology on a more abstract level, having a set of distinct types of elements from which it is built. In Appendix A a meta-model of the notion of ontology is presented for explanation purposes. The intention is to further clarify the terminology used in this thesis and what elements are used in the OntoCase method, but the model is not intended as a definition of the ontology concept. We suggest that the reader informally views the ontologies treated in this thesis as graph structures, where concepts are nodes and relations and restrictions are edges. This is a suitable mental abstraction for the purpose of this thesis. How this structure is actually represented or obtained depend on the representation language used, however this is not treated in detail in the thesis.

Ontology elements

The basic building blocks of an ontology are its concepts. Concepts are in many formal languages defined extensionally, as a set of instances displaying some common characteristics representing this concept. However, three aspects of a concept are often discussed, according to Cimiano [34]:

- the extension,
- the intensional description,
- and the set of lexical realisations.

Informally a concept can be described in two ways; intensionally, by specifying properties that characterise the instances of the concept, or extensionally, by explicitly listing its instances. The set of lexical realisations is a set of terms that may be used to represent the concept, e.g. both the terms 'packet' and 'parcel' may be used to represent the conceptual notion of a package in some ontology. Sometimes a lexical realisation is also called a label of the concept, the concept may thereby have a set of alternative labels. The set of lexical realisations can intuitively be viewed as a set of synonyms, although the exact definition of synonym is sometimes disputed within the field of linguistics. In this thesis we assume that a concept as a minimum is represented by a non-empty set of lexical realisations, but can additionally contain intensional descriptions and an extension, although our proposed ontology construction method does not currently provide the last two explicitly. The term 'class' is often used as a synonym to 'concept', e.g. in the context of the OWL language. In this thesis the terms 'concept' and 'class' are used interchangeably.

Another basic element is the relation. A relation is an association that exist between two or more concepts. In this thesis we will restrict ourselves to binary relations, i.e. relations associating two concepts, since this is a restriction posed by many ontology representation languages, also OWL as we shall see later. A relation can be viewed as a tuple of concepts, but usually this tuple is also associated with lexical realisations of the relation, and possibly a set of 'rules' determining the semantic interpretation of the relation when it is used for reasoning. The specific type of relation usually denoted subsumption or 'subclass of' has several lexical realisations, such as 'subsumption', 'subclass-of' and 'is_a'. This relation additionally has specific semantics. The relation is in most representation languages defined as being transitive, but it may or may not allow multiple inheritance. For

specific types of relations, such as 'subclass of', that are predefined in many ontology representation languages these semantics are inherent in the language, but user defined relations they need to be specified explicitly. In this thesis relations are viewed as tuples, binary associations between concepts, without specific semantics but with a, possibly empty, set of lexical realisations. In the implementation of the method, the specific semantics of the 'subclass of' relations is used by the method, while other relations are not further defined in terms of semantic interpretation. The term 'property' is often used as a synonym to 'relation', e.g. in the context of the OWL language. In this thesis the terms are used interchangeably.

Additional axioms can be defined in ontologies, as additional definitions, constraints and sometimes even rules, although rules are most often considered to be a layer of information on top of ontologies rather than included in an ontology. The nature of such axioms depend on the type of logical representation used. Since the focus of OntoCase at the moment is on concepts and relations, detailed treatment of axioms are outside the scope of this research. Restrictions creating 'anonymous' classes in OWL are not used in the pattern matching process at the moment, since the matching is focused on lexical realisations of the concepts and relations.

Ontology representation

We choose to view ontologies on the abstraction level of concepts and relations and their lexical realisations as labelled nodes and arcs of a graph, for most methods this thesis, rather than sets of logical formulae. Nevertheless, ontologies, and patterns, need to be represented in some computer processable language. Many different logical formalisms exist for representing and reasoning with ontologies, each one having its own benefits and drawbacks. The choice of representation should, in general, be made based on the requirements of the ontology, but additional factors such as standardisation and tool availability can influence the decision. Two different traditions have co-exist within the ontology, and Semantic Web, community. Some researchers prefer frame-style logics, closely related to traditional information modelling and object oriented formalisms, while others prefer to use description logics (DL) based formalisms. Separate from these two traditions a number of more specific logical languages exist with in AI, for special purpose ontologies, e.g. in robotics and other applications.

The Resource Description Framework[†] (RDF) is a W3C recommenda-

[†]<http://www.w3.org/RDF/>

tion for modelling metadata about resources on the web. The RDF model is based on triples consisting of a subject, a property, and an object. This is how we can describe resources in RDF, through stating triples connecting the resource to other resources or literals. URIs are used to identify resources. RDF Schema[‡] (RDFS) is an extension of RDF for defining vocabularies, i.e. schemas, to be used by RDF models. Using RDFS we can build a simple ontology, e.g. defining classes using *rdfs:Class* and a taxonomy of classes using *rdfs:subClassOf*. Both RDF and RDFS have, among other notations, an XML syntax[§] that also uses XML namespaces[¶].

Since 2004 OWL^{||} (Web Ontology Language) is a W3C recommendation for representing ontologies on the web. Due to this, it has emerged as the most popular choice of representation language also for non-web ontologies. OWL is based on DL and is available in several versions, regarding expressivity, where OWL-DL and OWL-lite are two such versions both supporting useful reasoning services. For details of DL the reader is referred to the Description Logic Handbook, edited by Baader et al. [139]. OWL-DL is the more expressive of the two versions, however it is still computationally complete and decidable. If the expressiveness is increased even more, allowing for the full power of RDF(s), also called OWL-Full, there are no computational guarantees. OWL builds on RDF in the sense that it extends RDF, but additionally restricts RDFS if one wishes to stay within OWL-DL, to make use of the reasoners available for that language. As an alternative language, based on the information modelling and object oriented traditions, F-logic as described by Kiefer et al. [112] has been one of the more popular. It is used as the basis for tools like OntoEdit and KAON mentioned later in this thesis. Also the popular ontology editor Protégé uses a frame-based format as its native format, although it additionally provides support for OWL.

Figure 2.3 presents an example of two class definitions in OWL XML-syntax^{**}. The first concept is the class 'Human', including a comment explaining the class in natural language. The class is a subclass of 'Animal', and disjoint with 'Fish', meaning that no instance of this ontology can be

[‡]<http://www.w3.org/TR/rdf-schema/>

[§]<http://www.w3.org/TR/rdf-syntax-grammar/>

[¶]<http://www.w3.org/TR/1999/REC-xml-names-19990114/>

^{||}<http://www.w3.org/2004/OWL/>

^{**}The example intends to give a brief, and incomplete, introduction to the OWL language and its XML syntax, but it is not essential for the general understanding of this thesis. Interested readers, who are not familiar with XML or DL, are referred to the more detailed literature referenced in this section.

both a human and a fish. The following subclass statement is actually a restriction on the 'hasChild' property, stating that for the class 'Human' all values of the 'hasChild' relation has to have the type 'Human'. In this case it is stated as a subclass restriction, but it could also have been modelled as a definition of the class 'Human', i.e. stated as an equivalent class. The second class is the concept 'Gender', defined as a nominal. The 'oneOf' statement includes an enumeration of the instances that constitute this class. The gender class is a subclass of owl:Thing, which is the top class of all OWL ontologies. Currently the OWL language is being extended. The new version of OWL, OWL 2, will contain features such as qualified cardinality restrictions, chains of properties and disjoint properties. In this thesis however, we have used the original OWL-DL language^{††} recommended by the W3C.

```

<owl:Class rdf:ID="Human">
  <rdfs:comment>A concept for representing humans.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <owl:disjointWith rdf:resource="#Fish"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasChild"/>
      <owl:allValuesFrom rdf:resource="#Human"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Gender">
  <owl:equivalentClass>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <rdf:Description rdf:ID="male"/>
        <rdf:Description rdf:ID="female"/>
      </owl:oneOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>

```

Figure 2.3: An example of two class definitions in the OWL XML-syntax.

Both F-logic based languages and OWL have been used to represent patterns and ontologies in this research. During the first iteration of the research, as described in section 4.1.2, F-logic was used as representation language, while during the second iteration OWL was selected instead. The main reason was the level of adoption of OWL by the research community

^{††}<http://www.w3.org/TR/owl-features/>

and consequently the availability of methods, tools and patterns represented in OWL. This has however given rise to some incompatibilities between some evaluations and results presented in later chapters. The initial experiment conducted within the SEMCO project used F-logic for pattern and ontology representation, e.g. allowing to state relations with concepts as property values. In later experiments those patterns had to be remodelled, when using patterns and ontologies represented in OWL, since such constructs are not within OWL-DL. An analysis of the consequences of this remodelling has not been performed, but some effects are noted in chapter 8.

Ontology categories and applications

In the introduction to this chapter the Semantic Web was briefly mentioned. The notion was introduced by Tim Berners Lee et al. in a nowadays famous article in the Scientific American in 2001 [16]. The idea of the Semantic Web is to introduce semantics into the links, documents and data on the web. Everything can be identified by a URI, from people to documents and data items. By having a schema to define the meaning and content of those resources, and their interrelations, the Semantic Web provides a machine interpretable version of the current web. Web ontologies are the means to describe this semantics, whereby the engineering, mapping and usage of ontologies has become important research topics. A discussion around ontology-based ILOG applications was also presented at the beginning of this chapter, but ontologies are used in many other fields than these particular application areas.

Ontologies can be used for many different purposes, such as the usage areas proposed by McGuinness [129]:

- Controlled vocabularies.
- Site organisation and navigation, e.g. browsing and customised search.
- Providing a structure from which to extend information content.
- Word sense disambiguation.
- Consistency checking, validation and verification.
- Completion and correction of insufficient information.
- Interoperability support.
- Configuration support.

Within the enterprise ILOG applications all these areas are relevant. A controlled vocabulary can be used to ensure a consistent use of terminology, e.g. providing a set of keywords for annotating information. Site organisation is important for company intranets and search functions in internal repositories. Checking consistency, verifying that information is correctly entered, and that constraints are not violated, is also important tasks. Supporting interoperability between information sources and personalisation of information are also core tasks of ILOG systems.

Ontologies can be used in a variety of application domains, ranging from autonomous agents to web portals, corporate intranets, and ILOG systems. This variety influences the features needed in an ontology. Sometimes a taxonomy might be enough, providing a controlled vocabulary, but in other cases more advanced features, such as restrictions and general axioms representing business rules, are needed to enhance reasoning capabilities. A study describing ontologies at different levels of complexity is presented by van Heijst et al. [208]. The levels are denoted terminological ontologies, information ontologies and knowledge modelling ontologies. Terminological ontologies deal with structuring terminology, information ontologies deal with the structure and format of information, such as documents, databases and web pages, and knowledge modelling ontologies deal with more complex tasks related to knowledge management.

Ontologies can be classified from yet another point of view, their level or generality as described by Guarino [89] in Figure 2.4. *Top-level ontologies* describe general concepts like space and time, which are most often independent of domains and specific tasks. These ontologies can be shared by large communities. *Domain ontologies* describe the vocabulary of a specific domain and are usually specialisations of top-level ontologies. *Task ontologies* in turn describe the vocabulary of a generic task or activity and may also be specialisations of top-level ontologies. Finally, *application ontologies* are a specialisation of domain ontologies, adapted for specific application tasks.

This thesis focuses on enterprise ontologies, mainly used in ILOG systems, which for example can mean providing a structure for content or supporting interoperability and access to knowledge sources. The focus is on application ontologies, tailored for specific applications within an enterprise, where the domain is governed by the enterprise in question. Uschold et al. [207] developed a top-level enterprise ontology, including many basic concepts that concern enterprises and their activities. This ontology however cannot be used as it is for our purposes, since we deal with de-

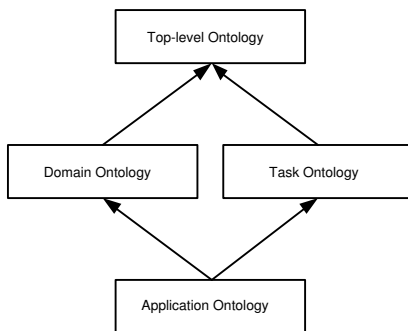


Figure 2.4: Ontologies with respect to their level of generality. [89]

scribing the information present, or needed, for the enterprise in question. The enterprise ontology by Uschold et al. [207] does not contain concepts for specifying the terminology of different enterprise domains. Moreover, it contains only abstract notions of products, services and processes.

Enterprise ontologies are, from our viewpoint, ontologies supporting enterprise applications, rather than ontologies describing general notions related to enterprises. Therefore enterprise ontologies are to be tailored to their intended applications, but can additionally contain definitions of general notions. An enterprise ontology for a product development enterprise can, for example, contain parts describing different aspects such as products, functions and processes. Products is concerned with what the enterprise produces, whether actual physical products or services, and the features, parts, and variants of those products. Functions, or functionalities, are the problems that the products, and product parts, aim to solve and descriptions how they solve them. In this way actual products can be connected to the customer requirements and the functionalities requested in both external and internal requirements. Processes connects the functionalities and products to the organisation of the enterprise, describing how the processes support the realisation of the requested functionalities into products or services. There might also be processes not directly involved in the development, such as marketing and management. Whether or not these are part of the ontology is of course depending on the application. The ontology can also contain parts concerning the internal organisation, people and groups within the enterprise, and competencies of the organisation and the individuals. Definition 2 presents the notion of *enterprise ontology* used in this thesis.

Definition 2. *An enterprise ontology is an ontology describing the domain, or parts of the domain, of an enterprise. It is to be used in an enterprise application aiming to solve a specific problem of that enterprise.*

2.3 Ontology engineering

Ontology engineering is a complex process covering the complete life-cycle of ontologies, from requirements engineering to usage and maintenance of the ontologies. There is currently no consensus on precisely what activities are part of this process, and how to refer to them, although efforts to structure and unify the terminology of the field have been proposed, e.g. by Suárez-Figueroa and Gómez-Pérez [196].

The life-cycle of ontologies can be compared to the life-cycle of other models or software artefacts. Suárez-Figueroa and Gómez-Pérez [194] list a set of possible life-cycle models suitable for ontology engineering, based on the analogy to software engineering. According to their summary of related work, most standards for software engineering define the software life-cycle as containing a concept activity, a requirements activity, a design activity, an implementation activity, a test activity, installation and checkout activities, operation and maintenance activities, and, sometimes, a retirement activity.

A life-cycle model can be selected and used within an engineering project, thus the above activities are mapped to the sequence and arrangement of activities of such a model. Examples of life-cycle models in software engineering are the spiral model and the waterfall model. Different models apply different strategies for performing the activities, e.g. the spiral model is an iterative model while the waterfall model is a sequential model performing all activities in sequence. Knowledge engineering models, and more specifically ontology engineering models, are commonly based on similar ideas or even adaptations of software engineering models as proposed by Suárez-Figueroa and Gómez-Pérez [194]. Although there exist ontology development methodologies there is still a lack of mature and well-tested life-cycle models.

The activities of the ontology development process can be divided into three categories, as suggested by Gómez-Pérez et al. [85]; management activities, development oriented activities, and support activities. Gómez-Pérez et al. further suggest a division of these categories into sub-categories and activities as follows:

- Management activities

- Scheduling
- Control
- Quality assurance
- Development activities
 - Pre-development
 - * Environment study
 - * Feasibility study
 - Development
 - * Specification
 - * Conceptualisation
 - * Formalisation
 - * Implementation
 - Post-development
 - * Maintenance
 - * (Re-)use
- Support activities
 - Knowledge acquisition
 - Integration
 - Merging
 - Alignment
 - Evaluation
 - Documentation
 - Configuration management

Each support activity is carried out during a specific part of the complete development process, but they are all essential to the development process. Knowledge acquisition is performed to elicit all knowledge to be represented in the ontology, this may be carried out both during pre-development stages, during development, and during maintenance. Ontology integration, merging, and alignment activities are performed mainly during development, when reusing ontologies or composing modules, but may also be part of the post-development activities.

In this thesis the focus is mainly on the development activities, pre- and post-development activities are not treated in detail. We focus on providing semi-automatic support for some of the activities during development. Several of the support activities are also highly relevant, such as knowledge acquisition, integration, and evaluation. We also remind the reader of the abstraction levels, mentioned in chapter 1, that needs to be considered when designing the ontology. Analogous to software systems an ontology has its overall architecture, its detailed design and its implemented 'code'. We repeat the questions from section 1.1.3 that summarise what needs to be considered in order to build an ontology:

1. What is the purpose of the ontology? How is it to be applied in a software system?
2. What parts are to form the ontology? How should the architecture of the ontology be formed?
3. What should the ontology contain? What concepts, relations, and axioms?
4. How should the ontology be represented syntactically?

The following sections focus on methods and tools for ontology development and supporting activities, rather than the complete life-cycle of ontologies.

2.3.1 Manual ontology construction

Today the ontology engineering process is mainly a manual process, often using methods quite similar to software engineering methodologies. An ontology can be built completely from scratch, in a cumbersome process of eliciting important concepts and finding generalisations, specialisations, relations and axioms. Reviews of existing methods were presented by Jones et al. [106], Fernández López [64], and Öhgren [150]. A more recent overview can be found in a project deliverable by Suárez-Figueroa et al. [193]. A different approach is to build an ontology almost completely out of existing knowledge sources, as suggested by Fikes and Farquhar [65], Levashova et al. [120], and more recently Hepp and de Bruijn [99]. Recent research tend to agree that reengineering of existing sources is a key to successful ontology engineering, whereby most recent method proposals incorporate both the

above mentioned approaches, see for example the proposed methodology by Suaréz-Figueroa and Gómez-Pérez [195].

Most methods incorporate some form of requirements analysis and specification steps at the beginning of the development. Requirements can be represented as scenarios or be more similar to software requirements, however a common way to semi-formally specify ontology requirements is through a set of competency questions, as described by Gruninger and Fox [88]. A competency question is an 'example question', of the kind that you need the ontology to be able to answer. Competency questions can be expressed as informal natural language questions, or more formally represented in logic or as queries to be submitted to the ontology. An example competency question for an ontology of university personnel could be: *'What are the courses taught by a certain professor?'*

There are numerous tools available for designing and implementing ontologies, such as OntoEdit that has evolved into OntoStudio*, Protégé† and WebProtégé‡, SWOOP§, KAON¶ and more recent additions such as Top-Braid Composer|| and the NeOn toolkit**. All these systems support manual ontology engineering but do not give any substantial assistance to the ontology engineer, aside from a graphical user interface and the usual help functions, see for example the survey of tools by Fensel and Gómez-Peréz [61]. Only recently have plans for tools supporting different life-cycle models and development work flows been proposed, but no such tool is currently available. A few research efforts have also proposed interfaces tailored to novice users, such as the tools by Bernstein and Kaufmann [17] and Dimitrova et al. [51] that both focus on using a controlled subset of natural language for ontology design and editing.

Another recent development is the focus on collaborative ontology editing, apart from pure multi-user and transaction support, i.e. discussing and reaching consensus about modelling decisions and changes, annotating and tracking modelling decisions and changes, proposing improvements, voting and rating. An ontology of collaborative ontology design was proposed by Gangemi et al. [79], as a formal metamodel of collaborative ontology design that can be used to characterise methods and work-flows of ontology de-

*<http://www.ontoprise.de/en/home/products/ontostudio/>

†<http://protege.stanford.edu/>

‡<http://bmir-protege-dev1.stanford.edu/webprotege/>

§<http://code.google.com/p/swoop/>

¶<http://kaon.semanticweb.org/>

||<http://www.topquadrant.com/topbraid/composer/index.html>

**<http://www.neon-toolkit.org/>

sign. Tool support for collaboration exist within the collaborative variant of Protégé, described by Tudorache and Fridman Noy [204], and for ontology editing over peer-to-peer networks, presented by Xexéo et al. [221]. Cicero is a collaborative environment for tracking design rationale and supporting collaborative design that will be integrated into the NeOn toolkit, see description by Dellschaft et al. [49].

2.3.2 Ontology learning

Semi-automatic ontology construction, or ontology learning (OL), has been briefly explained previously. In this section we present in depth explanations of the involved tasks and discuss existing systems. OL is a quite new field in ontology research. Researchers and practitioners have realised that building ontologies from scratch, and by hand, is too resource-demanding and time-consuming. In most cases there are already different knowledge sources that can be utilised in the ontology engineering process. Such existing knowledge sources can be documents, databases, taxonomies, web sites, and other similar structures. The key question is how to extract the knowledge encoded in these sources automatically, or at least semi-automatically, and reformulate it into an ontology. Simperl et al. [181] have recently proposed a methodology for applying OL within an ontology engineering process, as described at the beginning of section 2.3, by specifying the processes and activities that are specific when applying OL. Although the tasks specified by Simperl et al. [181] cover the ones listed in section 1.1.3, the tasks discussed in this thesis focuses on the technical problems of OL, while Simperl et al. mainly focus on the organisational aspects.

OL research tries to develop algorithms to extract ontological elements from different kinds of input, it is denoted 'learning' since many approaches apply some kind of machine learning (ML) techniques. According to Maedche [123] and Cimiano [34] the field of OL can be divided into a 'layer cake' of methods and algorithms, see Figure 2.5, through which more and more expressive elements can be extracted and included in the proposed ontology, or presented to the user for validation. Our view, as presented in section 1.1.3, includes these layers as tasks for element extraction in the element extraction step.

At the bottom of the layer cake is term extraction, synonym identification and concept formation, then follows taxonomy induction, general relation extraction, relation hierarchy induction, and finally the top levels concern axiom schemata and general axiom or rule induction. Most OL

systems existing today rely heavily on techniques from natural language processing (NLP) and computational linguistics for pre-processing a text corpus, see overviews of methods by Maedche [123], Cimiano et al. [35] and Cimiano [34]. Techniques from NLP, computational linguistics, and text mining are used for extracting different elements from the text, including relevance or confidence values representing the confidence with which the elements suggested to the user have been correctly extracted. Most existing techniques focus on natural language text or semi-structured text as input, although re-engineering of different kinds of resources is now receiving an increasing amount of interest from the research community. In this thesis OL will be interpreted in a broad sense, as a synonym to semi-automatic ontology construction and the focus will be mainly on ontology learning from text, although our focus is on the ontology composition rather than the text processing and element extraction. The following section presents some basic techniques used for OL and then a short survey of existing methods and systems is presented.

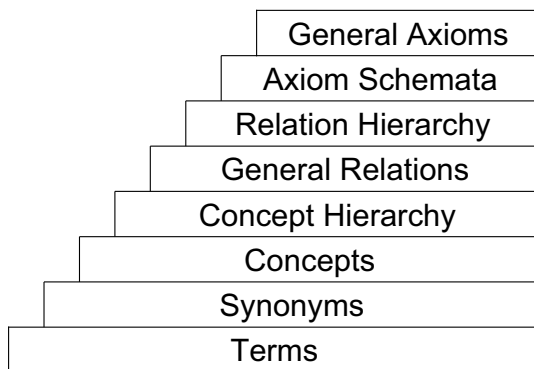


Figure 2.5: The Ontology Learning layers.

Basic methods

Many of the semi-automatic approaches rely on basic techniques that have been developed in other research fields. The techniques have often originated in data mining, text mining, NLP, computational linguistics, or machine learning. Some of the most important basic ideas are presented below in brief. Standard NLP techniques like chunking, parsing, and part-of

speech tagging are excluded from this description. Such methods are considered outside the scope of this thesis, since the methods proposed later in the thesis are not directly using such NLP techniques. These techniques are often relevant prerequisites for the basic OL methods however, for details the reader is referred to reference literature in the NLP field, such as Jurafsky and Martin [107].

Term extraction and concept formation

Most of the existing approaches use natural language texts as input and start by extracting terms from these texts. The texts are parsed using a natural language parser. Some approaches use domain specific texts while others deal with general text corpora. Most commonly the methods start by extracting terms according to their frequency of occurrence in the texts, although this frequency count is usually modified to represent some notion of 'relevance'. Some use algorithms, see for example Navigli et al. [141], to filter out words that are common in all texts, thus not domain specific, concepts that are only used in one single document and not in the whole corpus, and terms that are simply not frequent enough. This is the classical TFIDF-measure (term frequency, inverse document frequency) from Information Retrieval (IR), as explained by Baeza-Yates and Ribeiro-Neto [10]. Another approach to term extraction is to use preexisting linguistic patterns to extract terms that are part of important linguistic constructs, see for example methods by Wu and Hsu [220].

An important part of term detection is discovering multi-word terms. The most common way to assess the relevance of a multi-word term is the method using the C/NC-value method developed by Frantzi et al. [71]. The C-value proposes a method to assess the 'termhood' of a set of words by studying both its frequency in the text, but also its occurrence as a subset of other candidate terms, the number of such occurrences, and the length of the candidate term. The NC-value in addition incorporates the term contexts, surrounding words, in the assessment. By using such methods not only single word terms but compound terms can be extracted, as candidate lexical realisations of concepts.

Synonym detection is the problem of clustering terms into sets of synonyms. It is usually noted that true synonyms are very rare, most terms differ slightly in meaning although they may be acceptably exchangeable in a sentence. Such subtle differences are not usually considered in OL synonym detection and when forming concepts, commonly terms are grouped

with the intention of being used to represent a concept in a certain context, rather than to represent a set of true synonyms. This is similar to how synonyms are treated in for example WordNet [5] where terms are collected in so called 'synsets'. A synset is a collection of terms that have a similar meaning in a certain context. Until recently concept formation has been mostly seen as the process of term clustering and synonym detection. Recent approaches have however attempted to extract more complex concept definitions from text. An example of such a method is the LExO method suggested by Völker et al. [213] [212] where complex concept definitions and restrictions are extracted from natural language definitions and descriptions of terms, for example in sentences extracted from dictionaries.

Term classification and taxonomy extraction

A task that is addressed in many existing OL approaches is to extract a taxonomy for the ontology, i.e. extract subsumption relations between the formed concepts. There are several approaches to classifying terms, and some can also be used to extract more arbitrary relations. One system that attempts to classify words according to their context is SVETLAN, described by Chalendar and Grau [46]. The input is texts automatically divided into so called thematic units, i.e. different contexts. The sentences in the texts are parsed and specific patterns of words are extracted, the nouns in these sentence-parts are then aggregated into groups depending on their use with the same verb and the thematic unit or group of thematic units it belongs to. Gamallo et al. [72] use a similar method, where sentences are parsed, syntactic dependencies extracted, which can then be translated into semantic relations using predefined interpretation rules.

Another such system is Camille presented by Wiemer-Hastings et al. [219], which attempts to learn the meaning of unknown verbs by looking at how they are used and stating semantic constraints according to that. The output is hypotheses of the verb-meanings. To further arrange terms into a hierarchy an approach using a concept lattice is proposed by Sporleder [183]. This approach takes a set of terms as input, together with attributes and attribute values that are attached to the terms. The attributes have possibly been automatically extracted from texts. The terms are arranged in a hierarchy according to the attribute values, and the lattice is pruned to minimise redundancy. This is somewhat similar to formal concept analysis, described further below. Arranging concepts hierarchically can also be done using clustering algorithms, e.g. as proposed by Faure and Nédellec [58].

Formal concept analysis (FCA) is another approach that has been used in addition to the similarity and clustering techniques listed above, as for example presented by Cimiano [34], Cimiano et al. [36], and Völker and Rudolph [214]. In order to apply FCA for hierarchy induction the verbs used in connection with the terms, representing the concepts, to be ordered are collected as attributes of the term. Applying FCA on these attribute vectors will construct a concept lattice, which is transformed to a concept hierarchy, where the leaves are the terms and the intermediate nodes are named by the verbs applicable to the terms below the node in the concept hierarchy.

A very common approach to relation extraction is the used of lexico-syntactic patterns. Such patterns were first proposed in 1992 by Hearst [98], and are today usually referred to as 'Hearst-patterns'. They have since then been used in numerous settings. An example of a Hearst-pattern is NP_0 such as $\{NP_1, NP_2, \dots (and|or)\}NP_n$, where NP_i stands for an arbitrary noun phrase. The pattern would for example, match the following sentence; 'animals such as cats and dogs'. This particular pattern is a pattern representing a subsumption relation, i.e. we can conclude that cats are a kind of animal. Similar patterns could also be developed for other types of relations, and even for domain specific relations. Another popular pattern, or rather heuristic, is the commonly used so called 'head heuristic', sometimes also denoted 'vertical relations heuristic'. This heuristic is very simple but quite useful for OL. The basic idea is that modifiers are added to a word in order to construct more specific terms, i.e. the term 'graduate student' consists of the term 'student' and the modifier 'graduate'. Using the head heuristic we can derive that a 'graduate student' is a kind of 'student'.

Co-occurrence theory and association rules

A problem that many approaches for semi-automatic ontology construction struggle with is the extraction of arbitrary relations. One of the most used approaches is to apply co-occurrence theory or association rule mining to derive possible relations in an ontology. This has been discussed for example by Srikant and Agrawal [184], Ding and Engels [53], Heyer et al. [101], and Cherfi and Toussaint [33].

The basic idea of co-occurrence theory, sometimes called collocation theory, is that if two concepts often occur 'near' each other they probably have some relationship. Some approaches, like Heyer et al.[101], only consider co-occurrence in the same sentence but it could be extended to textual en-

vironments in general, e.g. the same document. Association rules are an extension of co-occurrence theory, when the co-occurrences are formulated as rules. For example a rule can be that if a document contains the terms t_1, \dots, t_i then it also tends to contain the terms t_{i+1}, \dots, t_n , see suggestions by Cherfi and Toussaint [33]. Such rules can be used to extract additional relations, while at the same time updating the rules themselves.

The main problem using co-occurrences is that the nature of the relation is not clear, it is an arbitrary relation. Approaches have thereby been proposed to find also suitable labels of such arbitrary relation, such as the approach by Kavalec and Svatec [110]. They propose to look for verbs used in connection with terms representing the two concepts in order to find the correct relations that hold in that specific context.

Below a number of existing OL systems are briefly described, to give an overview of the state of the art in the field.

SOAT

SOAT is an OL system proposed by Wu and Hsu [220]. Using templates of part-of-speech tagged phrase structures, in Chinese, concepts and relations are collected from texts. A text corpus from the domain is needed and a seed word, for example the name of the domain. The output is a prototype of the domain ontology, to be validated by an ontology engineer. Additionally, the extraction rules might need to be updated during the process, and that can only be done by the ontology engineer.

Four different relations between concepts are extracted; category, synonym, attribute and event. The category hierarchy roughly corresponds to a taxonomy. Event is the actions that can be associated with a concept, Wu and Hsu [220] explain this by exemplifying that the concept 'car' can be driven, parked, raced, washed and repaired. The extraction algorithm takes as input a parsed domain corpus with part-of-speech tags, and performs the following steps:

1. Select a 'seed-word' that forms a potential root set R , performed by the user.
2. Begin the following recursive process:
 - a) Pick a keyword A as the root from R .

- b) Find a new related keyword B by using the extraction rules and add it to the domain ontology according to the rules.
- c) If there are no more related keywords, remove A from R .
- d) Put B into R .
- e) Repeat steps 2(a)-2(d) until either R becomes empty or the number of nodes generated exceeds a predefined threshold.

Advantages of this approach according to the authors are that it is highly automated and apparently very efficient [220]. Disadvantages are that it requires heavy preparations and adaptations, in the form of constructing correct extraction rules. The method is also quite static because it can only extract predefined types of relations between concepts.

Text-To-Onto and Text2Onto

A fairly elaborate approach is the Text-To-Onto system developed at the University of Karlsruhe by Maedche et al. [124] [127] [125] [123] [126]. Text-To-Onto was further developed by Cimiano and Völker [37] and Cimiano [34] and in its new version renamed to Text2Onto*. The system is intended to support both constructing ontologies and maintaining them. The idea is to get better extraction results and to reduce the need for an experienced ontology engineer by using several different extraction approaches and then combining the results. The architecture of the original Text-To-Onto system was based on the following parts:

1. Data import and processing
2. NLP System
3. Algorithm library
4. Result presentation
5. Ontology engineering environment

The idea of the data import and processing is to be able to use different kinds of structured or unstructured data as input to the system. There can be existing knowledge sources like databases, web pages, documents or even already existing ontologies. All these sources are then pre-processed and transformed into an appropriate form, possible structure information

*<http://ontoware.org/projects/text2onto>

is retrieved. Any resulting texts are then used as a text corpus for the remainder of the process. The NLP step uses a shallow text parser. The result from the parsing process is used in the third step, by the algorithms in the algorithm library. For the task of ontology learning there were originally two kinds of algorithms included in the Text-To-Onto library:

- Statistical and data mining algorithms
- Pattern-based algorithms

The first kind uses frequencies of words in the text to suggest possibly important concepts and a hierarchical clustering algorithm to derive the concept hierarchy. For extracting non-taxonomical relations a modified version of a standard association rule algorithm is used, the naming of the relations has to be done manually. The second kind of algorithms uses pre-defined lexico-syntactic patterns to extract both taxonomical and non-taxonomical relations.

In the algorithm library component there are also algorithms used for ontology maintenance, mainly ontology pruning and ontology refinement. The pruning algorithm studies the frequency of concepts and relations in the domain specific corpus compared to their frequencies in a generic corpus, this in order to prune concepts and relations that are not domain specific. This is mainly aimed at adapting more general ontologies to a specific domain. Ontology refinement is done similarly to ontology learning but is also based on the assumption that unknown words often are used in the same way as known words, so that a similarity measure can be applied to determine how similar the two words are.

The result presentation step lets the user accept or discard the suggestions that the system offers and also to name the arbitrary binary relations found. On top of this the system uses an ontology engineering environment, in this case KAON, which lets the user manually edit the extracted ontology.

The more recent version, called Text2Onto, adds several algorithms, such as FCA-based hierarchy induction, specific methods and patterns for extracting part-of relations, and instance extraction. The main improvements of Text2Onto compared to Text-To-Onto, apart from the added algorithms in the library, is a revised architecture that focuses on change management and result combination. The Text-To-Onto/Text2Onto approach is highly flexible, and contains a rich library of algorithms, covering most state of the art approaches in element extraction for OL.

OntoLearn

The OntoLearn system, presented by Navigli et al. [141], uses a different approach. This method builds on the general linguistic 'ontology' WordNet, see Miller et al. [134] and Fellbaum et al. [5] for details, and the SemCor knowledge base containing tagged sentences, to interpret terms found in the extraction process. The resulting ontology is a refined part of WordNet.

The approach is more specialised towards language processing than for example the previously described system, Text-To-Onto. The OntoLearn architecture consists of three main phases:

1. Terminology extraction
2. Semantic interpretation
3. Creating a specialised view of WordNet

The system is part of a larger architecture, for editing, validating, and managing ontologies. Input to the system could be anything that can be viewed as natural language texts, for example web pages or ordinary documents.

The first phase uses shallow parsing techniques to extract possibly relevant terminology, both in the form of single words and complex phrases, from a text corpus. The frequency of the candidates in the domain corpus are measured relatively to a corpus across several domains, this yields a domain relevance score. A second measure is the domain consensus, which measures in how many different documents the term or phrase occurs.

The second phase performs semantic interpretation, which in turn means both semantic disambiguation and extraction of semantic relations. The semantic disambiguation starts by arranging the sets of terms extracted into small trees, according to string inclusion. Then, for every word in these trees, a semantic net is created using the appropriate sense of the word, a WordNet synset, and all relations concerning that concept in WordNet are included, up to a certain distance in the graph. A gloss and a topic is created for every word, using the WordNet concept definition and sentences from the SemCor knowledge base. To connect all these semantic nets, possible intersections are investigated using predefined semantic patterns, and taxonomic relations inferred using WordNet.

To extract semantic relations a predefined inventory of semantic relation types is used and by using inductive machine learning the appropriate relations are associated with pairs of concepts. Inductive machine learning means that a learning set is manually tagged and then the inductive

learner builds a tagging model. In the third phase the resulting forest of concept trees is integrated, either with an existing domain ontology or with WordNet, which is then pruned of irrelevant concepts.

This approach is specialised towards language applications and one of its major drawbacks is that several resources need to be specified in advance, like semantic patterns for connecting the semantic nets and the relation types available for the non-taxonomic relations. Another drawback is that it uses several quite specific pre-existing structures, such as SemCor. An advantage is that it is highly automated, performs most tasks without any user intervention and that glosses are generated for the concepts.

TextStorm ad Clouds

A similar application that also uses the lexical resource WordNet is the TextStorm system presented by Oliveira et al. [153]. Another system, Clouds, is also presented in the same paper, it is an extension of TextStorm and constructs rules to evolve the ontology further through a dialogue with the user. The TextStorm system parses and tags a text file, using WordNet, and then extracts binary predicates from the text corpus. The predicates symbolise relations between terms, extracted from sentences. The synsets of WordNet are used to avoid extracting the same concept, denoted by a different term, several times. The output of TextStorm is not an ontology per se, but simply these predicates. It is the Clouds system that supports the actual building of the 'concept map', i.e. ontology, in cooperation with the user. It uses a machine learning algorithm to pose questions to the user and draw conclusions depending on the answer.

ASIUM

The ASIUM system, by Faure [59] and Faure and Nédellec [58], is a system directed mainly at linguistic applications and the system uses linguistic patterns and machine learning. The specific novelty of this approach when it emerged was the conceptual clustering algorithm used to generate the structure of the ontology. In addition to this system another system was also suggested, the Mo'K workbench described by Bisson et al. [20], which is a development and evaluation workbench for conceptual clustering algorithms.

The method used in ASIUM can be divided into three steps, the final two steps are performed repeatedly and in parallel:

1. Extraction of instantiated syntactic frames
2. Frame learning and clustering
3. Validation

The first step parses the text corpus used as input to the application, and together with a special post-processing tool generates instantiated syntactic frames. The instantiated syntactic frames are instances of common sentence patterns consisting of certain combinations of verbs, prepositions, and head words. In case of ambiguities all possible frames are stored, since validation by the user is not done at this stage.

A learning algorithm is applied, using the basic assumption that "words occurring together after the same preposition and with the same verbs represent the same concept" [59]. Frame instances are extracted and new frames are learned at the same time. A frequency measure is also used to determine what concepts are more reliable and more important than others. To start the actual clustering step, base clusters are formed by putting together head words found after the same verb and preposition. Different clustering algorithms can then be applied, and different similarity measures for the clustering algorithms can be used.

One of the suggested clustering algorithms is called ASIUM-Pyramid and it is a bottom-up and breadth-first algorithm. The name comes from a restriction to only generate 'pyramids', i.e. trees where every node has exactly two children or none at all. The distance measure is used to determine the pairwise distance between the basic clusters and two clusters are aggregated if their similarity exceeds a given threshold. The user validates all the generated clusters at each level, then the process is repeated until no more clustering can be performed.

The weakness of this approach is that it is very much language dependent. Another weakness is the output, which in this case is restricted to a 'pyramidal tree', only consisting of a taxonomy. Advantages are that the user validates each step in the process before the next step is performed, instead of validating only the finished ontology. This may generate more work for the user during the process but it might produce a better result.

Adaptiva

The Adaptiva system by Brewster et al. [25] is yet another system based on linguistic patterns and machine learning, but with a more iterative and cooperative approach than the previously described systems. Brewster et

al. explicitly consider what the system is to be used for and who the users are, and state that a user should be able to do three things:

1. Draft an ontology or select an existing one.
2. Validate sentences illustrating particular relations in the ontology.
3. Label a relation shown in an example sentence and recognise other relations of the same kind.

The process of constructing an ontology is divided into two stages that each consist of three steps:

1. Learning taxonomic relations
 - a) Bootstrapping
 - b) Pattern learning and user validation
 - c) Cleanup
2. Learning other relations
 - a) Bootstrapping
 - b) Pattern learning and user validation
 - c) Cleanup

The bootstrapping process involves selecting a text corpus and either drafting or selecting a seed ontology, which in the second stage most likely is the taxonomy from the first stage. The pattern learning starts by the system using the seed ontology to present a set of example sentences to the user for validation. The examples that are approved by the user are then used to build generic patterns which in turn are used to extract new sentences from the text corpus. These are again presented to the user for validation and so on, until the user stops the process. The cleanup step is where the user can edit the ontology directly, merge or divide concepts, relabel concepts and relations and ask the learner to find all relations between two given nodes.

One of the stated advantages of this method is that the output is not only an ontology but also a trained learning system for further developing and evolving the ontology in the future, which is a good point. Another advantage is the simplicity of user tasks, the user only has to have a vague idea of what an ontology is, the rest is taken care of by the system. Disadvantages can be that the user has to identify what kind of relations the

example sentences represent, this may not be a trivial task, and that the only relations that are extracted are those expressed within one sentence, not between sentences.

SYNDIKATE

The SYNDIKATE system by Hahn and Romacker [94], also presented by Hahn and Markó [92] [93], is more directed towards evolution and maintenance of ontologies, and other knowledge sources, than to construct an ontology 'from scratch'. The system has several uses, not only ontology construction, but grammar learning and other tasks in NLP.

The system builds on hypotheses and different quality measures to determine the likeliness of those hypotheses. Texts are parsed and a parse tree is generated. For each unknown concept in the tree a new hypothesis is formed, using among other things the already existing part of the ontology. Different quality labels are attached to the hypotheses. One is linguistic quality, which is a measure of how well the hypothesis conforms to known phrasal patterns and other structural properties. The conceptual quality in turn, reflects the hypothesis' conformity to the existing part of the ontology and alternate concept hypotheses. Alternate hypotheses of the same unknown term divides the set of all hypotheses into hypothesis spaces, which represents different choices of alternate hypotheses.

The advantages of this approach, as stated by the authors [92], are that no specific learning algorithm is needed since the learning is carried out by a terminological reasoning system, and also that the method is entirely unsupervised. This is also one of the few approaches that use a quality-based learning which might reflect reality better than a system that attempts to determine the only 'true' solution. Possible drawbacks are that this approach could generate too many hypotheses so that the approach gets unmanageable and the calculations involved become too resource-demanding. From the papers it is also somewhat unclear what the actual output of the system is and how that output can be used, for example a set of hypothesis spaces might not be useful as basis for some applications, sometimes there might be a need for deciding exactly what a concept means. The result may be viewed more as a help for constructing an ontology.

OntoGen

OntoGen is yet another tool for OL. It is proposed by Fortuna et al. [67] [68]. This is tool to build light weight ontologies, called 'topic ontologies' by the

authors. The approach is highly interactive and the developers have focused mainly on the user interface of the tool. The system extracts 'keywords' from a text corpus and can apply several concept clustering algorithms for supervised or unsupervised clustering of the topics to generate relations between topics. Interesting features are the customization and collaboration features, i.e. users can choose to apply their own term weighting methods instead of general TFIDF and work on several ontologies at once, in cooperation with other users. Although this approach is quite interesting in terms of its user interaction features, the ontologies produced are simple and requires a lot of user intervention to generate.

LExO and RELExO

LexO and RELExO, together with a method for learning disjointness, are as far as we are aware the only available OL systems that try to generate expressive ontologies, rather than only terms and relations for light weight ontologies. LExO takes as input a set of sentences, preferably similar to dictionary definitions, and outputs their representation as OWL axioms. The example given by Völker et al. [212] is the sentence "A farmer is a person who operates a farm". The sentence is parsed by a dependency parser, and the parse tree is translated into an XML-based format, whereby a set of rules can be applied. The output is a DL expression, such as $Farmer \equiv Person \cap \exists operate.Farm$, i.e. a formal definition of the class 'Farmer'. RELExO adds to this method the possibility to use FCA relation exploration to enrich the concepts added through the LExO method.

Hypotheses for disjointness axioms can be generated by applying a few general heuristics. Völker et al. [212] propose three kinds of heuristics:

- Taxonomic overlap
- Semantic similarity
- Patterns

Taxonomic overlap can be investigated in the ontology structure itself, but evidence can also be gathered from sources on the web and OL methods for learning subsumption relations. Different semantic similarity measures can be applied, and lexico-syntactic patterns used for natural language texts can also discover indications of disjointness.

These methods are very promising, and certainly novel, but still quite restricted at the moment. The input needed for generating concept defini-

tions is very specific. If definitions of all concepts already exist in natural language, then it is also of great help for manual ontology engineering. It is the lack of such definitions that is one of the main motivations of the OL field in the first place. However, more general methods, such as the disjointness extraction discussed above is, even at this early stage of research, already very useful.

Boxer

Boxer represents the border between state of the art research in NLP and OL. Boxer, as presented by Bos [23], is a system for deep semantic analysis of natural language. Based on a grammar and background information, such as a predefined set of roles represented by verbs, Boxer produces a discourse representation of the input sentences. On an abstract level it can be viewed as translating natural language into logical formulae, based on some background knowledge of the language. This reflects the general direction of NLP and computational linguistics, to focus more on deeper semantic analysis as computing resources increase and better algorithms emerge. Although this can be seen as the ideal OL approach, translating natural language directly into logics, there are many problems with such an approach. The method is highly dependent on the represented grammar and the set of predefined roles that can be identified, and with which the elements of the text are tagged. Nevertheless, the approach is worth mentioning, because it is similar to approaches such as LExO presented earlier, trying to generate rich and expressive structures directly from natural language texts.

OntoLT

OntoLT is a plug-in for the ontology engineering environment Protégé, presented by Buitelaar et al. [29]. The plug-in is intended as a toolbox, or middleware, between NLP systems and ontology editors. It supports the definition of patterns for mapping between constructs in tagged text and ontology elements. In this sense OntoLT is not a learning approach in itself, rather an environment allowing the user to define rules for extracting ontological elements from linguistically pre-processed text. However, it is often referred to an OL tool in literature and is thereby included here.

Ontology enrichment

Above a set of ontology learning systems were presented, mainly based on the idea to construct an ontology 'from scratch'. Ontology enrichment can be viewed as a special case of ontology learning, with a specific precondition: an initial ontology needs to already be present. Some of the already discussed OL systems could also be used for this purpose, these areas are actually just two views of the same problem.

One specific ontology enrichment approach is proposed by Faatz and Steinmetz [57] where new possible concepts and their place in the taxonomy are extracted from web documents. Using a distance measure to determine how closely linked the concepts in the existing ontology are, the method then tries to find new concepts that can be put into this hierarchy using mathematical optimisation algorithms. A similar approach also using a distance measure, but adding to that a word overlap measure, and an information content measure, is described by Stevenson [187].

An additional form of ontology enrichment would be to add instances to the ontology, so called ontology population. This is commonly viewed as a task of OL, even though this construct a knowledge base rather than only the ontology. Since our proposed approach is not related to ontology population, and does not treat the instance level, we will not describe ontology population further in this thesis.

2.4 Chapter summary

This chapter presented basic definitions of ontology and the parts that are commonly viewed to constitute ontology elements. Ontologies can be used as means for knowledge representation in semantic systems, such as on the Semantic Web or within ILOG systems. Ontologies have a long history as noted at the beginning of this chapter, and ontologies can be of many different forms, but today ontologies are usually a formally represented set of well-defined concepts and relations that are used to create a knowledge base of facts. Different forms of reasoning can be applied on this knowledge base in order to facilitate applications such as improved information retrieval and structuring of enterprise information.

2.4.1 Manual ontology construction summary

In order to benefit from such improvements the prerequisite is that an ontology of the domain, tailored to the specific application, is available. Many ontology projects still construct ontologies more or less from scratch, or by manually reengineering data from other non-ontological resources. Manual methodologies for ontology development exist, and tools are available to assist in the actual modelling process. Manually engineered ontologies can be perfectly tailored to the task at hand, but in many cases it can be too expensive and time-consuming to construct an ontology in this manner. For tasks such as information retrieval, perhaps it is not always necessary to use a perfect ontology, a mediocre ontology might still provide enough semantics to improve application performance, or as put by Brewster et al. [24]: *"Good ontologies are the ones that serve their purpose. Complete ontologies are probably more than what most knowledge services require to function properly. The biggest impediment to ontology use is the cost of building them, and deploying 'scruffy' ontologies that are cheap to build and easy to maintain might be a more practical and economical option."* Partly as a response to such issues the field of ontology learning has emerged, aiming to provide semi-automatic methods supporting different parts of the ontology engineering process.

2.4.2 Ontology learning summary

Most existing OL approaches build on roughly the same foundations, in NLP, data mining, and machine learning. Usually the approaches consider terms and taxonomic relations in the extraction process, some consider arbitrary relations, but very few consider more advanced tasks, such as relation naming or axiom extraction. The level of user involvement also differs greatly, from almost completely automated to involving the user in each step. This is a bit misleading though, because the completely automated approaches are probably the ones most in need of evaluation and validation after the construction process is finished. Whether the user is involved during the process or at the end, the important question is to what extent a user is assisted in this task. User interfaces have not been prioritised in most cases, and sometimes the results provided to the user can be very large and unstructured, thus perhaps confusing the user rather than helping to make sense of domain concepts. The lack of proper user interfaces is also noted by Simperl et al. [181] in their use case study applying OL tools.

A common characteristic of many of the approaches is that they mainly extract single concepts and relations, only a few attempt to construct larger parts of the ontology at once. Many of the approaches also consider the concepts and relations one by one, mostly the user has to validate single concepts and relations and not suggestions of 'ontology components'. From a user perspective, evaluating single terms and, possibly unnamed, relations is very difficult, while seeing a concepts or relation in a context gives much more information to base a decision on. In light weight ontologies the labels and surrounding relations are the only existing evidence of how to interpret the semantics of a concept, thereby the relation structure is very important.

All OL methods based on text corpora also suffer from the inherent problem that general background knowledge is not explicitly specified in the texts, as explained by Brewster et al. [26]. Some general knowledge, whether domain specific or more general, is always omitted and assumed to be held by the reader. Brewster et al. [26] argue that transferring knowledge through a text is most often a process of knowledge maintenance rather than knowledge creation. Due to this fact, it is sometimes impossible to extract all relevant knowledge to be represented in an ontology from a text corpus, in such cases some more general background knowledge must be used in addition to the texts. One such possible source of knowledge is ontology patterns. Some of the OL systems presented use patterns in the element extraction process but those patterns are mainly lexico-syntactic patterns for extracting certain types of relations. None incorporate ontology content patterns, which we will be our focus in the following chapters.

Chapter 3

Patterns and knowledge reuse

For providing solutions supporting ILOG, knowledge representation through ontologies is a key issue. The challenge for an enterprise that wishes to apply an ILOG system or method is then to construct an accurate and up-to-date ontology describing their organisation and information content. Constructing ontologies has classically been a purely manual task, performed by knowledge engineers in cooperation with domain experts. In chapter 2 different approaches to automate the tasks in ontology development that have emerged in the field of ontology learning (OL) were described. Our research focuses on knowledge reuse through the introduction of ontology patterns in OL. This chapter presents some background on reuse in general and patterns in particular, and show how patterns aim to reduce the effort to construct solutions and improve the solution quality. At the end of the chapter, the case-based reasoning methodology is mentioned in brief, since this has provided inspiration to the OntoCase method. To set the stage for the presentation of related work this chapter first includes definitions and descriptions of basic concepts, e.g. the general notion of pattern, patterns in other areas, and the theoretical notion of reuse.

3.1 Reuse

Reuse is commonly agreed to be a way of improving the quality of your work, in combination with reducing the complexity of the task and thereby possibly also the time and effort needed to perform it. General discussions about reuse have been published by Sutcliffe [199] and Buschmann et al.

[31]. One way to facilitate reuse is by using patterns, as described later in this chapter, but there are many other aspects to reuse that are not very often discussed theoretically. A thorough explanation of reuse mechanisms is presented by Sutcliffe [199], where the author discusses the background of reuse as a concept, why it is desirable, but also why it is a hard thing to accomplish. The main obstacle that a reuse methodology has to overcome is the lack of motivation among developers. Before the process has been established, and reuse libraries and other facilities are in place, it will require an increased effort from developers in order to establish the reuse organisation. A strong motivation to share your ideas is needed, without getting any immediate personal benefit from it.

Sutcliffe [199] describes the reuse process as divided into two steps: *design for reuse* and *design by reuse*. These are the two sides of the 'reuse coin', one cannot exist without the other. To facilitate 'design by reuse' there must first be a well established 'design for reuse' process. These two processes can be illustrated as in Figure 3.1.

What to reuse is also an interesting question, because reuse can be applied on different levels. For example, reuse in software engineering can, according to Sutcliffe [199], be done in at least three stages:

- Requirements reuse, e.g. reusable models of the domain or generic models of the requirement specification.
- Design reuse, e.g. reusable models, data structures and algorithms.
- Software component reuse, e.g. reusable software classes and editable source code.

Reuse can also be divided into categories by looking at what the developer knows of the reusable objects, again see the discussion by Sutcliffe [199]. These categories are denoted *black-box reuse*, *glass-box reuse* and *white-box reuse*. Black-box reuse reduces the task complexity and learning burden of the developer, since the content of the 'black-box' is not visible, the developers only have to learn how to use its (hopefully) well-defined interfaces. White-box reuse is the opposite, the whole interior of the module is known to the developers and they can modify it at will.

Glass-box reuse lies somewhere in-between. The interior functions of the object are visible but not possible to change. Although black-box reuse is the most ideal case, with respect to the possible gain, it will obviously not be possible in the case of knowledge reuse, such as reuse of ontologies.

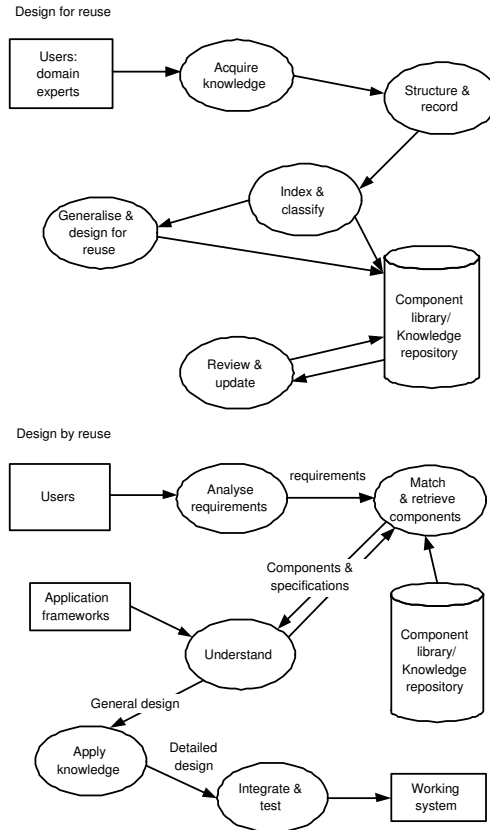


Figure 3.1: The reuse processes. [199]

As stated by Sutcliffe [199]: *"in order to reuse knowledge it has first to be understood"*, therefore knowledge reuse is always glass- or white-box reuse.

Reuse has been suggested within knowledge engineering for several decades, but it is not until recently the notion of patterns has been adopted on a broader scale. However, patterns have been used in many other areas throughout history, and in the engineering sciences at least for several decades. Most likely the reader has some general idea of what a patterns is, possibly resembling a template for solving some specific problem, whether it is a design problem in software engineering or a modelling problem in ontology engineering. Since the notion of patterns is central to this thesis a separate section, section 3.2, is devoted to the notion of patterns in gen-

eral, and ontology patterns in particular. In the rest of this section we will instead focus on the more general notion of reuse and summarise how reuse has so far been incorporated in ontology engineering.

3.1.1 Ontology reuse

Reuse is commonly seen as a self-evident way of improving your work, both because it means using existing and, hopefully, well-tested solutions and because it generally is believed to shorten the development time and cost. In ontology engineering there are mainly two different perspectives held by researchers and ontology developers. The first states that ontologies are so domain and application specific that almost nothing can ever be reused for another application case. In this view integration of existing ontologies is less relevant in the development methodology, see for example statements by Sugumaran and Storey [197] and Uschold [206] that point in this direction. The other extreme states that ontology engineering is almost impossible to do 'from scratch' and therefore the ontology development process is always just a process of combining already existing parts of ontologies and other knowledge sources. Statements by Fikes and Farquhar [65] and Levashova et al. [120] point in this direction. Most researchers would place themselves somewhere in the middle of this scale where the idea is to find existing ontologies, or parts of ontologies, that might fit the application case and then combine, adapt, and extend them seems feasible. Nevertheless, parts of the ontology could also be built 'from scratch' if needed.

The problems of ontology reuse are analogous to the problems of reuse in software engineering presented by Sutcliffe [199]. For example, how does one find the appropriate ontologies to reuse? Even with ontology libraries and search engines present, described below, it can be hard to find appropriate candidates for reuse. How does one choose the right one, or the appropriate parts from several different ontologies? How can these parts be used, integrated or merged? On the representation language level, the last question, of translation, integration, and merging, has been quite well researched, and recently also approaches taking into account the semantic level have emerged. However, the results produced by such methods are usually uncertain and far from perfect. The rest of this section discusses some different categories of approaches to ontology reuse more in depth.

Generic components

The idea of using small pre-existing building blocks to compose an ontology, or a knowledge base, is similar to the idea of using patterns, described in later sections. In the area of ontology reuse so far there has been only a few recent attempts in this direction. One idea, using a library of generic concepts, for building ontologies, is presented by Barker et al. [13]. The main idea is to construct a library of generic concepts to let a domain expert choose among when building an ontology. Only those concepts that are not specific to any domain, have a clear description, and can be unambiguously represented by a single English language term should be considered for the library, according to Barker et al. [13]. Another similar approach is the DOGMA methodology and the tool DOGMA-MESS [118] for collaborative ontology construction and evolution, based on a library of generic facts, denoted lexons, that can be reused for constructing and evolving ontologies.

This idea is similar to the many attempts at creating general top-level ontologies, such as the WordNet dictionary described by Miller et al. [134] and Fellbaum et al.[5], CYC presented by Lenat [119], SUMO proposed by Niles and Pease [143] and Pease et al. [157]. It is also similar to the attempts involving semantic patterns for ontologies described further in later sections. The main issue concerning these kinds of approaches is the balance between reusability and usefulness. Very small and very generic components can be highly reusable, but cannot represent a large amount of useful information and can probably easily be constructed from scratch. On the other hand, complex and highly useful components are less reusable.

Modularisation

The idea of modularisation has been fruitful for example in the software engineering field, and can be viewed as a step towards component-based reuse, compare to reuse of software components as discussed by Sutcliffe [199]. Stuckenschmidt and Klein [190] propose a set of requirements that should hold for ontology modules:

1. Loose coupling, which means that nothing can be assumed about different modules, they might have very little in common, neither concepts nor representation language, and therefore as little interaction as possible should be required between the modules.
2. Self-containment, which means that every module should be able to exist and function, this could involve performing reasoning tasks or

query-answering, without any other module.

3. Integrity, which means that even though modules should be self-contained they could depend on other modules so there should be ways to check for, and adapt to, changes in order to ensure correctness.

These three requirements are then implemented for DL ontologies, using different techniques, by Stuckenschmidt and Klein [188]:

1. View-based mappings, which means that the different modules are connected through conjunctive queries.
2. Interface compilation, which means that in order to provide self-containment using the view-based mappings the result of the query is computed off-line and added as axioms in the querying module, thus enabling local reasoning.
3. Change detection and automated updates, which means that a record of ontological changes, and their impact, is kept and also the dependencies between modules so that changes can be propagated through the system.

This results in a modular ontology built from self-contained ontology modules where some concepts are internally defined in a module and other concepts are determined using queries over other ontology modules. On this ontology reasoning can be performed and there is also a system for handling changes in the modules. The approach is very specific, and there are no guidelines on how to decide what is a module and what should be part of it. Neither is anything said about how to reuse modules in a new ontology or how to find the appropriate module to reuse from a specific ontology. Some more guidelines could be found in approaches such as the one proposed by Schlicht and Stuckenschmidt [174], where some practical and measurable criteria of how to form 'good' modules are proposed.

Another modularisation approach is described by Rector [166], also concerning ontologies represented in DL. The author proposes an extensive set of rules on how to develop and represent an ontology to achieve explicitness and modularity. The modules are distinguished by the notion that all differentiating notions of the taxonomy of one module must be of the same sort, e.g. functional or structural. Another requirement is that all modules should only consist of a taxonomic tree structure, no multiple inheritance is allowed within a module.

More recently several language extensions for OWL have emerged to additionally support modularisation and reasoning with modular ontologies. A survey of such extensions was presented by Wang et al. [217]. An example of such a language is P-OWL, proposed by Bao et al. [12]. P-OWL stands for package-based OWL, where an ontology is divided into a set of packages, i.e. modules, with well-defined interfaces. The language then supports information hiding and reasoning over packages.

To summarise, there exist many approaches to modularisation, but mainly focusing on how to represent modules and how to technically use them. Few guidelines have been proposed on how to modularise an ontology, content-wise, and how to actually reuse modules from modular ontologies. Similar problems occur as for the general case of ontology reuse, e.g. how to find appropriate modules, how to select the right ones, how to reuse and combine them. The technical issues related to partitioning of an ontology, that are addressed by current modularisation approaches, are certainly useful. At least an ontology engineer is no longer left with only the possibility of including a complete ontology through `owl:import`, when finding an interesting part of an existing ontology. However, the availability of modules and the technical possibility to combine and reasoning over them does not solve the general reuse problem.

Finding and selecting reusable ontologies

Some approaches for ontology selection and adaptation were presented already early in the history of ontologies within computer science, such as the KARO system proposed by Pirlein and Studer [159], which propose methods for finding and adapting reusable parts in ontologies. The authors define an ontology as a collection of general top-level categories with associated relations and assumptions. It is these assumptions about the ontologies that renders the approach more or less irrelevant in today's situation, where most ontologies are also domain dependent, except for top-level ontologies. Another similar paper is the one by Visser and Bench-Capon [209] which aims to show that ontologies are highly reusable within one domain, but this again assumes very high-level ontologies which are again not very relevant to the scope of this thesis. Despite this, a few good general points are stated by Pirlein and Studer [159], that also concur with the recommendations for software engineering given by Sutcliffe [199]:

- To be able to reuse ontologies they have to be developed with reuse in mind, so that for example modelling decisions and assumptions are made explicit.

- An appropriate knowledge processing environment must be present, to be able to select parts of an ontology and to adapt them.
- The development process of ontologies must be structured in a way so that reuse of ontologies can be incorporated in a well-defined way.

Classically reusable ontologies have been stored in ontology repositories, as proposed by Fikes and Farquhar [65], and there are still such repositories being developed and maintained for certain domains. The BioPortal ontology repository for biomedical ontologies described by Noy et al. [147] is one example. Repositories usually provide browsing support and simple search functions. However, most of the ontologies today are present on the web, hence finding appropriate ontologies is today addressed mainly by ontology search engines. Such search engines usually have simple search interfaces, providing keyword search for concept identifiers and labels. The SQORE approach by Ungrangsi et al. [205] additionally provides a 'query by example' interface using a small model as a search criteria. By using approaches for ontology matching to search the web or an ontology library, selection could be supported similarly to the match-making by Billig and Sandkuhl [18], the conceptual graph matching of Zhong et al. [227], semantic matching by Yeh et al. [223], or what is suggested in general terms by Sutcliffe [199]. The Watson search engine by dAquin et al. [43] additionally provides an interface for visualising the content of the retrieved ontologies in order to ease the process of selection. Nevertheless, none of these approaches assist in retrieving only relevant parts of an ontology, and there are inherent issues in languages like OWL where imports are restricted to importing only complete ontologies as mentioned previously.

Ontology search engines is an area growing fast, today the list of existing search engines include OntoKhoj presented by Patel et al. [156], OntoSelect[†] presented by Buitelaar et al. [27], Swoogle^{††} presented by Ding et al. [52], OntoSearch and the subsequent ONTOSEARCH2^{‡‡} presented by Zhang et al. [226] and Thomas et al. [200] respectively, Watson* presented by dAquin et al. [43], and SQORE presented by Ungrangsi et al. [205]. However, it is not enough to only find a set of possibly reusable ontologies, we also need to select which one(s) to reuse. Similarly as when searching for documents on the web we would expect some kind of guidance, which ontologies are most

[†]<http://olp.dfki.de/ontoselect/>

^{††}<http://swoogle.umbc.edu/>

^{‡‡}<http://www.ontosearch.org/>

*<http://watson.kmi.open.ac.uk/WatsonWUI/>

relevant to our query, i.e. the case at hand. All of the above mentioned search engines provide an ordering of the search results, but usually there is no detailed motivation for this ranking presented to the user.

An elaborate ranking scheme, called *AktiveRank*, is proposed by Alani and Brewster [6]. Four measures are used; class matches, centrality, density and semantic similarity. The class match measure combines exact and inexact string matching to match concept labels to search terms. The centrality measure considers the matched concepts, from the previous step, and produces a value showing how representative those concepts are within the ontology based on their placement in the taxonomic hierarchy. The intuition behind the measure is that the more central concepts are in the taxonomy the more important and representative they are. The density measure describes the degree of detail, e.g. the number of relations, in the context of the matched concepts. The intuition behind this measure is that the more 'dense' the context, the description and definition, of a concept the clearer is the semantics of the concept and the more useful the concept would be to reuse. Finally, the semantic similarity measure is a measure of the proximity, in terms of path lengths traversing the ontology relations, of the matched concepts in the ontology. The intuition behind this measure is that matches spread out through an ontology are less important than matches clustered in one part.

A similar approach, used in the *OntoSelect* application, is proposed by Buitelaar et al. [27]. In this case the ontologies are evaluated with respect to three criteria; coverage, structure and connectedness. In this case the aim is to select an ontology for document annotation, thereby the coverage is measured with respect to the terms present in the document collection, instead of a keyword query, and both concept and relation labels are considered. Structure is a class-to-property ratio where the intuition is that the more properties the ontology contains the more 'advanced', and thereby the more useful it is. The connectedness measure tries to reflect the number of other ontologies to which the current one is connected, mainly through imports, and the reliability of those ontologies.

A recent overview of ontology selection was presented by Sabou et al. [169] where ontology selection approaches were classified with respect to three categories, i.e. approaches addressing ontology popularity, richness of knowledge, and topic coverage of the ontologies. While approaches such as *OntoKhoj* and *Swoogle* use algorithms similar to Google's *PageRank*, i.e. links between pages, for determining popularity, *OntoSelect* also utilises the connections between ontologies, i.e. imports. An approach using an open

rating system for determining ontology popularity has been proposed by Lewen et al. [121]. The knowledge richness is addressed in ranking schemes such as *AktiveRank* and *OntoSelect*. Most approaches take into account some kind of topic coverage, but usually this is a very simple coverage of the search terms entered. More complex matching is however done in approaches such as *SQORE*, by Ungrangsi et al. [205], and *ONTOSEARCH2*, by Thomas et al. [200], where the logical structures of both the query and the retrieved ontologies are taken into account. Nevertheless, the open issues identified by Sabou et al. [169] are still not completely covered by any of the current methods. The main open issues are:

- Semantic evaluations
- Considering relations
- Combining knowledge sources

We can also note that there exist no, automatic or even semi-automatic, methods for selecting ontology patterns, which is the main focus of our method proposed later in this thesis. Our method mainly addresses the first and second issue above as we shall see in later chapters.

Combining ontologies

When reusing existing resources, such as existing ontologies, a key task is how to combine the resources. Methods and tools for matching, aligning, merging, and integrating ontologies have been present during a number of years, early surveys were presented by Fensel and Gómez-Peréz [61] and Kalfoglou and Schorlemmer [109], Shvaiko and Euzenat [177], and a more recent book by Euzenat and Shvaiko [56] also contains a comprehensive summary of the field. When combining knowledge, mismatches can occur either on the level of syntax and semantics of the representation languages, i.e. language-level mismatches, or on the level of how something is modelled, i.e. ontology-level mismatches.

Euzenat and Shvaiko [56] attempts at a terminological clarification, i.e. explaining terms such as matching, alignment, mapping, merging, and integration. Ontology matching is according to Euzenat and Shvaiko [56] the step initiating a merging or integration process, but can also be done by itself to compare two ontologies or to reach an alignment. The aim matching is to find a set of correspondences between the ontologies. A mapping is a functional correspondence, as described by Kalfoglou and Schorlemmer

[108] and Euzenat and Shvaiko [56]. Matching can also be the starting point for ontology alignment, i.e. for developing an agreement between the different ontologies but still keeping the original ontologies intact, or for finding translation rules between the different ontologies, as described by Ichise et al. [103], Mitra et al. [136], and Mena et al. [131].

Some studies, such as by Noy and Musen [146], Pinto and Martins [158], Keet [111], and Klein [113], distinguish at least two different approaches for combining ontologies. Either the aim is to combine two ontologies of the same subject area (domain), in which case the process is called merging, or the aim is to combine two ontologies from different subject areas (domains), in which case it is named integration. In a merging process the source ontologies are unified into a new ontology where it is difficult to determine which parts have been taken from which source ontology, since the domain is the same. Fusion is a term rarely used but can be seen as a specific way to merge ontologies, where the individual concepts loose or change their previous identity.

Ontology matching tools are commonly used in order to first match and then reuse ontologies, examples of early tools are Chimaera, as described by McGuinness [130], and PROMPT, as described by Noy and Musen [144] [145]. These tools provide an interface supporting the user in finding overlaps in ontologies and by suggesting how to integrate them. Nevertheless, it is up to the ontology engineer to do the actual integration or merging. Also different methodologies for manual alignment, merging, and integration, especially on the language level, exist since many years, e.g. approaches by Russ et al. [168], Mihoubi et al. [133], and Pinto and Martins [158]. However, during the past few years the field of ontology matching has received a great increase in research interest and an increasing number of proposed approaches, especially for automating these processes.

One such system is GLUE that uses machine learning algorithms to find the most probable matches for concepts in two ontologies, proposed by Doan et al. [54]. The system uses several similarity measures and several learning algorithms to perform the matching process. FCA-Merge is another approach, presented by Stumme and Maedche [192]. The approach is based on FCA and is, as the name states, aimed at merging ontologies not only to find correspondences. The main idea is to use a concept lattice, together with instances of the concepts present in texts, for extracting formal contexts for the ontologies, determining similarities and then assisting the user in building a merged ontology. Another approach relying on FCA is presented by Kokla and Kavouras [114]. IF-Map discussed by Kalfoglou and

Schorlemmer [108] is yet another approach, this time based on information-flow theory, which generates a merged ontology with assistance from the user. A similarly well-founded approach is described by Schorlemmer [175].

In a recent paper by Shvaiko and Euzenat [178] ten challenges in current ontology matching research were discussed:

- Large-scale evaluation
- Performance
- Missing background-knowledge
- Uncertainty
- Selection and configuration
- User involvement
- Explanation
- Collaborative matching
- Alignment management
- Reasoning with alignments

These are challenges that have only partly been addressed so far, especially by earlier systems such as the ones mentioned above. The issue of large-scale evaluations has to some extent been addressed through the current evaluation challenges and competitions, see below, but test data still need to be increased in size, and the comparability of systems and tools need to increase. This is also connected to performance, where matching generation time and space complexity need to be addressed, in order to arrive at efficient automatic tools, e.g. for online web ontology matching. A hard problem is the issue of missing background knowledge, i.e. knowledge not explicitly represented within the ontology. Some systems utilise background knowledge when matching, but usually this is domain and task independent knowledge such as general thesauri or other linguistic resources.

Ontology matching is an uncertain task, and this uncertainty needs to be represented and reasoned with. Automatic selection and configuration of appropriate matching methods for a case at hand is also essential, as well as guiding users during the matching process, or reducing the user involvement all together. Another important issue from a user perspective

is matching explanation, for the user to be able to evaluate correspondences and trust the given alignment the reasoning behind it must be explained by the system. Collaboration has also received an increased interest within this field, and methods for collaboratively matching ontologies are needed. Finally, alignments need to be managed and used, thus tool support is needed also for this.

Since the introduction of the Ontology Alignment Evaluation Initiative* (OAEI) and the first alignment competition in 2004, numerous algorithms, methods and tools have been proposed and tested through the initiative. Only during the past two year's competitions 23 different tools and algorithms have participated. The initiative provides datasets for benchmarking and during the yearly competition additional datasets from different domains are provided. This initiative has partly emerged as a response to the challenges listed above, and the shortcomings in earlier research approaches and systems. Based on the results from the last competition we take four representative examples of recent approaches that have performed well:

- ASMOV, presented by Jean-Mary and Kabuka [105].
- Lily, presented by Wang and Xu [216].
- RiMOM, presented by Zhang et al. [225].
- DSSim, presented by Nagy et al. [138].

ASMOV by Jean-Mary and Kabuka [105] can be run completely automatically and it applies four different kinds of similarity measures that are combined during the matching process; lexical similarity, internal and external concept structure, and instance similarity. For computing the lexical similarity a thesaurus is used as background knowledge, and the structure and formal definitions of the ontology, such as the taxonomy, other properties, and restrictions, are taken into account in the measures called internal and external concept structure. The potential correspondences are pruned through checking for inconsistencies and structural mismatches between the ontologies.

Lily by Wang and Xu [216] also uses several different strategies in combination, differentiating between sizes of ontologies in order to optimise the method's performance and between ontologies containing different amounts of lexical information. The authors use a notion called semantic subgraphs to represent the context of the elements of the ontologies, this is similar

*<http://oaei.ontologymatching.org/>

in intention to the internal and external concept structure aspects used by ASMOV. The web is used as background knowledge for semantically connecting the ontologies. Propagation of similarity values can be done through a graph representation of the ontology. This method also includes a 'debugging' step, where correspondences are checked and corrected, as far as possible using only automatic methods.

RiMOM by Zhang et al. [225] uses a multi-strategy approach just as the previous two methods. In this case the method selection is based on three measures; structure similarity, label similarity, and label meaning. The structure similarity roughly assesses if the taxonomic structures are similar, the label similarity counts identical terms in the ontologies, and the label meaning measure counts the number of concept labels that are present in WordNet. Based on these rough characteristics the rest of the process is tailored to the ontologies at hand. The actual matching algorithms include lexical matching of labels based on string matching or background knowledge such as WordNet, graph-based structural measures studying paths in the ontology graphs, and different combination and propagation strategies.

DSSim by Nagy et al. [138] is also a multi-strategy approach, applying both syntactic and semantic matching methods to assess possible correspondences. WordNet is used as background knowledge, to interpret the terms present in each ontology. This method focuses specifically on the combination of the different measures, and how to handle the inherent uncertainty. Matching methods are considered as agents with a certain expertise, and methods for voting and belief combination are used to arrive at a plausible conclusion regarding a correspondence.

Outside the scope of the OAEI challenges, additional methods have been proposed for semi-automatic matching and for understanding the nature of correspondences. One such proposal is the use of correspondence patterns to match ontologies proposed by Scharffe et al. [173]. Usually correspondences are detected in the form of equality or subsumption, but these patterns represent more complex correspondences. An example is when a concept in one ontology can be mapped to a concept in another ontology with a property restriction, i.e. Bordeaux wines are wines where the territory property has to have the value Bordeaux, as exemplified by Scharffe et al. [173]. The developed patterns are available in an online repository, but so far such patterns are mainly used manually.

In summary, we can note that there exist numerous methods and tools for ontology matching. However, most of them apply similar techniques. Some type of syntactic and linguistic matching is commonly applied, to

match concept and relations names, and sometimes additionally labels, comments and other linguistic structures present. Some methods apply simple string matching, while other methods include advanced features for detecting abbreviations and dealing with compound terms. On the semantic level background knowledge is commonly used, trying to interpret the meaning of concepts and relations. Background knowledge may be general, such as the linguistic knowledge incorporated in WordNet or information present on the web, or it may be domain specific, such as biomedical thesauri. Selection, configuration, and combination of different matching methods is a key issue, as noted also by Shvaiko and Euzenat [178].

Our research focuses on matching between extracted ontological elements, produced by typical OL systems, and general ontology content design patterns. This is in fact a kind of ontology matching. However, we have a very specific setting that lets us tailor our approach to this task. On one hand we have the ontological elements extracted by an OL system, e.g. terms and relations extracted from a text corpus. On the other hand we have the small, self-contained and general ontology content design patterns, i.e. small general ontologies containing only a few concepts and relations.

The set of extracted elements is in fact an ontology, although it is usually light weight in terms of complexity, sparsely connected, and very 'flat' in terms of taxonomical depth. It is also uncertain, since it is extracted by means of an automatic method. Many of the above mentioned approaches rely on structural matching methods, using an environment of the concepts and relations, to improve on lexical matching. This not always possible for the kind of sparse and 'flat' ontologies we are dealing with. At the same time the input ontology is also uncertain, meaning that it is not certain that the complete structure will be included in a resulting ontology, since it can contain errors. Most existing matching approaches assume that the input is reliable. The ontology content design patterns are in fact also ontologies. In this case we are dealing with very general ontologies, i.e. containing abstract concepts, and in contrast to the extracted input these ontologies are reliable and well connected. The patterns are also quite small, so more computationally expensive methods might be feasible to apply.

Ontology matching, as described above, commonly assumes some overlap between the ontologies, or at least that they are on a similar level of abstraction. In our case we know that this is not the case, and we can use this knowledge to develop a specific matching method that suits this case. The method we propose is also tailored to the fact that on one side we have quite small ontologies, the patterns, which increases the choice of algorithms

possible. The patterns are also available beforehand, in a catalogue, and can be pre-processed, even manually if necessary. All these factors form the explanation why we do not choose to reuse any single existing ontology matching approach for OntoCase, but instead develop our own ranking scheme. Nevertheless, our solution is heavily inspired by common methods used in ontology matching.

3.2 Patterns

As mentioned in the introduction to this chapter, patterns are a specific case of reuse. In this section we will study the notion of pattern more closely, and we start by considering the term pattern and what it denotes. Patterns may be regarded from several perspectives, two possible perspectives consider ontology patterns as subject of pattern mining and recognition or as engineered templates, respectively.

In the *pattern recognition perspective* patterns are recurring sets of entities that can be found in some set of structures, e.g. recurring sets of concepts and relations in ontologies. In this sense, there are no requirements on what a pattern may contain, its level of abstraction, or how it may be structured or retrieved. On the other hand, in the *patterns as templates perspective*, patterns are commonly carefully engineered templates that represent a consensus perspective on how to solve a specific problem. The templates have to be sufficiently general and abstract in order to be reusable in many cases and they have to represent some notion of best practices, i.e. a 'good' or at least commonly accepted way to solve the problem.

In ontology engineering both kinds of perspectives are applied, and additionally hybrid perspectives are common. The pattern recognition perspective and different kinds of hybrids are for example used for OL, similarly as in text and data mining, e.g. when discovering linguistic patterns for extracting ontological elements. The patterns as templates perspective is more common in manual ontology engineering for guiding ontology design processes, teaching novice ontology engineers best practices and communicating between developers. Ontology patterns have been inspired by patterns in other areas, such as software patterns and data model patterns, and thereby there are many different kinds of patterns available for ontology engineering.

Patterns, in the template and best practices sense, aim to give benefits in at least three ways, see the discussion by Menzies [132]:

- Reuse

- Guidance
- Communication

Reuse benefits are provided by patterns that can be used as templates, or even partial solutions, from which a designer can bootstrap a solution to the current problem. The essence of the reuse benefit is to be able to build better systems by basing the new solutions on old and 'well-proven' solutions. Guidance benefits can be provided by patterns that point at common problems, and possibly suggest well-proven ways to solve those problems. In this case the pattern provides hints for the developer, what are the important issues and what are the pitfalls, and can guide the development into the right direction. The communication benefit occurs when patterns act as a means for describing existing systems or solutions, and communicating these to others. Communication benefits can occur as the introduction of a common vocabulary between developers, but also by providing a structured way to introduce best-practises to novice developers.

In addition to the three benefits described by Menzies [132] one can imagine that the reuse benefit may introduce more beneficial things than simply the increased quality. The development process might in some cases become faster and easier to perform. Additionally, guidance might not only come in the form of explicit guidance stated within the patterns themselves, but can also be the presence of a set of patterns where the developer can find inspiration when thinking about the problem and designing the new system, or ontology. A drawback can be that such a pattern catalogue also might restrict developers, especially developers with limited experience, and prevent them from inventing new and innovative solutions on their own.

Although this introduction to patterns most likely has given the reader an intuitive feeling of what we mean with the term pattern, the notion is treated more in detail in the next section, also connecting the pattern idea to its origin and relations to other fields.

3.2.1 What is a pattern?

Intuitively everyone has an idea of what a pattern is, it is something re-occurring that can be recognised from one time to another and from one application to another. This is something we all use in our daily lives and in our profession as well. We seldom invent completely new solutions, since problems often resemble other problems we have encountered before we

reuse parts of those solutions, we use old solutions as patterns. We also recognise new patterns in our surroundings that can be useful.

More formalised patterns, that can be recognised and used by a whole community of people and presented in text-books, have been used in several fields, such as architecture, economics, and computer science. The most renown patterns in computer science are probably software patterns. Ideas for these patterns came from the architecture field already in the 70's, through the work of Christopher Alexander and his books on pattern languages in architecture; "A Pattern Language" [9] and "The Timeless Way of Building" [8]. In his books Alexander proposed to start making implicit knowledge of how to construct useful and esthetically appealing buildings and towns explicit, and to store it as patterns for others to reuse.

This idea of making assumptions and experience explicit, in order to be reusable by others, was transferred into the software field. The probably most well-known book in the software pattern community is the book on design patterns written by Gamma et al. in 1995 [73]. The patterns treated describe common design solutions in object oriented software design. Generally, this kind of patterns can assist on the following issues according to Buschmann et al. [31]:

- A pattern addresses a recurring design problem that arises in specific design situations, and presents a solution to it.
- Patterns document existing, well-proven design experience.
- Patterns identify and specify abstractions that are above the level of single classes and instances of components.
- Patterns provide a common vocabulary and understanding for design principles.
- Patterns are a means of documenting software architectures.
- Patterns support the construction of software with defined properties.
- Patterns help you build complex and heterogeneous software architectures.
- Patterns help you manage software complexity.

Although these statements refer to the software engineering community they can easily be generalised to allow for patterns in almost any construction context. Buschmann et al. [31] also state that a pattern is made up

of a *context*, when or where the pattern is useful or valid, a *problem* that arises within that context, to which this pattern is connected, and finally the proven *solution* that the pattern proposes for this problem. This has to be presented in a formalised way so that the patterns can be communicated to a community of people. Often the description template of software patterns consists of a set of headings, as suggested both by Buschmann et al. [31] and Gamma et al. [73]:

1. Pattern name
2. Also known as (other well-known names of the pattern)
3. Problem and motivation
4. Context and applicability
5. Intent, solution and consequences
6. Structure
7. Participants and dynamics (describing the different parts of the pattern and their relations)
8. Implementation (guidelines for implementing the pattern)
9. Variants
10. Examples
11. Known uses
12. Related patterns

This template for describing a pattern could easily be adapted to many other kinds of patterns, not only software patterns, and has also been used for ontology patterns. The level of formality and how to express the different parts in the list may vary, but the structure is usually similar, e.g. how to illustrate the structure of a pattern may vary greatly between different kinds of patterns while the heading is still part of all templates.

The previous discussion in this section deals with patterns used for construction, for producing a solution to a design problem. Another kind of pattern is the notion of a pattern as an observed regularity within some set of objects, and that may be used to discover new instances of that regularity. In the fields of pattern mining and recognition, computational linguistics,

and information extraction patterns are commonly identified through mining large datasets. In this case it can be the patterns themselves that are the product of the process, and what the patterns in turn can tell the researcher about the object of study. A common example is the mining of association rules, that has for example been used to discover relations in shopping behaviour of people entering supermarkets. A possible association rule would be that people who buy milk and bread also tend to buy butter, and this discovered pattern can then be used by the supermarkets to arrange their offerings in a more convenient way. We also saw examples of the use of association rule mining in OL system in the last chapter.

Pattern mining has been used in ontology learning (OL), e.g. by Brewster et al. [25] in order to automatically learn patterns, denoted rules as above, for extracting ontological elements from text. Such patterns are related to the linguistic pattern first proposed by Hearst [98], that were mentioned in section 2.3. Hearst's patterns are partial sentences with 'gaps' where we can expect to find a filler in a text, and in this manner find evidence of a certain relation between terms. A simplified version of one of Hearst's patterns for hyponym, comparable to subsumption, extraction is " NP_1 and other NP_2 ", where NP_1 and NP_2 are two noun phrases. If this pattern is found in the text it can be concluded that NP_2 is a hyponym of NP_1 . For example the partial sentence 'dogs and other animals' would match this pattern, and we can agree that dogs are a kind of animals. Although Hearst only proposed one specific kind of patterns, similar approaches are today widely used in for example OL, and further developed such as by Brewster et al. [25] and Maedche [123].

As we have seen above there are two different views of patterns. The first kind of patterns are general template-like structures encoding experiences and best practices in order to provide a solution to a common problem, while the second kind is concerned with regularities in some set of object, regardless of any general problem or level of generality. When considering the ontology engineering field the first kind of patterns can readily be adapted to fit ontologies and be used in the same way there. Modelling is a hard problem and to encode modelling best practices seem like a obvious solution. Also the second kind can be transferred to ontologies and has already been used in OL.

3.2.2 Patterns in different fields

Patterns have, as stated earlier, been adopted by many areas of computer science and neighbouring sciences. In this section usage areas of patterns are presented. Sometimes patterns are also grouped based on similar naming although inherently different. The list is not complete but may be seen as examples of pattern usage today.

Software patterns

The most well-known kind of patterns in computer science today are probably software patterns. Today, structured software development projects using an object oriented paradigm but conducted without the use of some kind of patterns are rare. Patterns are believed to vouch for reusability of solutions, product quality, and management of the system complexity. These patterns are often divided into different kinds, according to when in the development process they are used and what level of abstraction, or granularity, they operate on. The most common categories are:

- Analysis patterns, e.g. proposed by Fowler [69], Fernandez and Yuan [62], and Kolp et al. [115].
- Architecture patterns, e.g. described by Shaw [176], Fowler [70], Geyer-Schulz and Hahsler [83], and Buschmann et al. [31].
- Design patterns, e.g. proposed by Gamma et al. [73], Buschmann et al. [31], and Björk et al.[21].
- Programming language idioms, e.g. described by Buschmann et al. [31].

Analysis patterns are used at the very beginning of the software engineering process. These patterns reflect issues such as conceptual structures of business processes, e.g. for different business domains, and how to transform these processes into software. One example of an analysis pattern is the *account pattern* that can be used for bank accounts, but also for other records where there is a need for keeping track of not only the current value but also details of each change that affects this value, as explained by Fowler [69]. The pattern could be illustrated using UML notation, see Figure 3.2, where the entries record each change. Similar to Fowler's analysis patterns [69] are the ones suggested by Fernandez and Yuan [62].

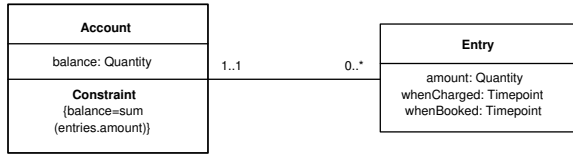


Figure 3.2: The account pattern. [69]

At another level of granularity there are also analysis patterns describing the overall structure of organisations, since modelling the organisation is often a task performed early in the analysis process. Such patterns are described by Kolp et al. [115]. An example is the *joint venture pattern* that represents the situation when several enterprises join together in order to perform a specific task, which is coordinated by a joint management. An illustration of this pattern can be seen in Figure 3.3 We choose to keep the original notation used by Kolp et al. [115], which is based on the modelling framework *i** where actors are represented by circles, dependencies are represented by arrows and the ovals, clouds and hexagons represent the nature of the dependencies.

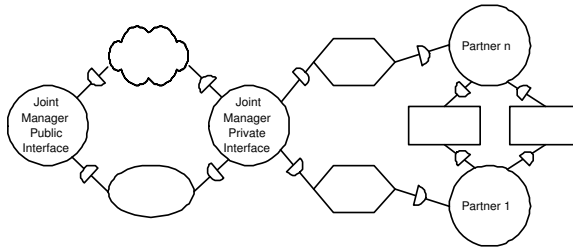


Figure 3.3: The joint venture pattern. [115]

Architecture patterns show overall structuring principles for software systems, how to divide them into subsystems, the responsibilities of the subsystems, and their relations, as described by Buschmann et al. [31]. Even though Geyer-Schulz and Hahsler [83] name their patterns analysis patterns they present detailed descriptions of software architectures, similar to those by Shaw [176], and can thereby be classified as software architecture patterns. Architecture patterns are sometimes denoted architecture styles. Although these terms may differ in meaning, a pattern might be expected to be more detailed than a style, they are not differentiated in this thesis.

An example of an architecture pattern is the *layered architecture pattern*, as described by Shaw [176]. This pattern illustrates a type of architecture suitable for systems that involve distinct services that can be classified hierarchically, see Figure 3.4 using the informal notation also used by Shaw. Usually the interaction between layers are procedure calls. The layers are then further decomposed using other patterns.

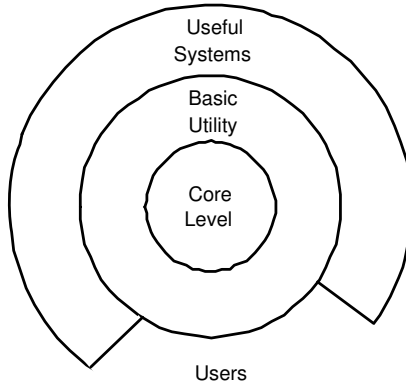


Figure 3.4: The layered architecture pattern. [176]

A slightly different kind of architecture patterns is presented by Fowler [70]. These are not as high-level as the architectural patterns presented by Shaw [176], but not as detailed as the design patterns of Gamma et al. [73] either. They resemble more a further development of Fowler's own analysis patterns in [69] into reusable architectures. The patterns are specific for building enterprise applications. An example of this kind of pattern is the *domain model pattern* [70], used to insert a model of the domain into the application in order to capture complex business logic. The overall structure of the pattern can be expressed using UML, as seen in Figure 3.5.

When further developing the details of a system design patterns are used. These are more detailed and low-level patterns that describe ways to design the architectural components and their interactions, but they are still independent of programming languages, as defined by Buschmann et al. [31]. An example of a design pattern is the *observer pattern*. The observer pattern can be used when changes need to be propagated to other objects in a dynamic way, observers can be dynamically registered and unregistered. The observer pattern can be illustrated using UML notation, as can be seen

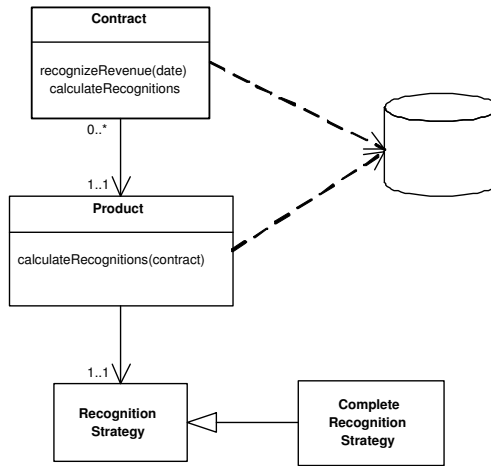


Figure 3.5: The domain model pattern. [70]

in Figure 3.6. There have also emerged design patterns specific to certain software domains, such as the game design patterns developed by Björk et al. [21], describing certain interactions common when designing computer games.

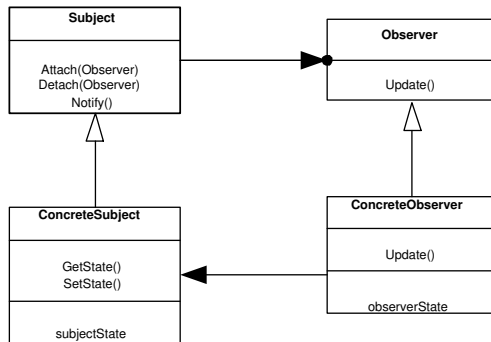


Figure 3.6: The observer design pattern. [73]

Programming language idioms are language specific patterns describing ways to implement certain aspects of components, or interactions between them, using language specific features, as described by Buschmann et al. [31]. An example, also found in [31], is an idiom for implementing the singleton design pattern in C++. The purpose of the design pattern is to

ensure that only one instance of a class exists at run-time, and the idiom describes an appropriate way to solve this in the programming language C++. The idiom is depicted in Figure 3.7 through a simplified variant of the pattern description template mentioned previously in this chapter.

Name	Singleton C++
Problem	Implementation of the Singleton Design Pattern in C++
Solution	Make the constructor private. Declare a static member variable <i>theInstance</i> that refers to the single existing instance of the class. Initialise this pointer to zero. Define a public static member function <i>getInstance()</i> that returns the value of <i>theInstance</i> . The first time <i>getInstance()</i> is called it creates the single instance with <i>new</i> and assigns its address to <i>theInstance</i> .
Example	<pre> class Singleton { static Singleton *theInstance; Singleton(); public: static Singleton *getInstance() { if (! theInstance) theInstance = new Singleton; return theInstance; } }; //... Singleton* Singleton::theInstance = 0; </pre>

Figure 3.7: Singleton Idiom in C++ from Buschmann et al. [31].

Data model patterns

Data model patterns are somewhat similar to software patterns in the sense that they represent an effort in standardising common and well-proven solutions to help modellers be more effective and increase the quality of the models. Data modelling can of course also be part of software engineering, since a database can be a component of a software system, so these two areas are strongly interconnected.

Data model patterns were first presented by Hay [97]. The author motivates his patterns with the usual arguments of better communication between modellers, reducing the complexity of the modelling task, increasing the quality and reusability of the models. He also divides possible conventions of modelling into three levels; syntactic conventions, positional conventions and semantic conventions. Syntactic conventions are modelling languages, like UML or other notations, positional conventions decide the overall organisation of the symbols on a page and their interconnections, and finally semantic conventions deal with grouping of entities according to their meaning and how different business situations are perceived. It is the

authors intent that mainly the final category can benefit from the presented data model patterns.

An example proposed by Hay [97] is a pattern for modelling the buying and selling of products and services. A slightly simplified example of this pattern, illustrated using UML, can be seen in Figure 3.8. Hay’s patterns have also been further developed by for example Silverston [180] [179], who additionally present categories of domain specific patterns.

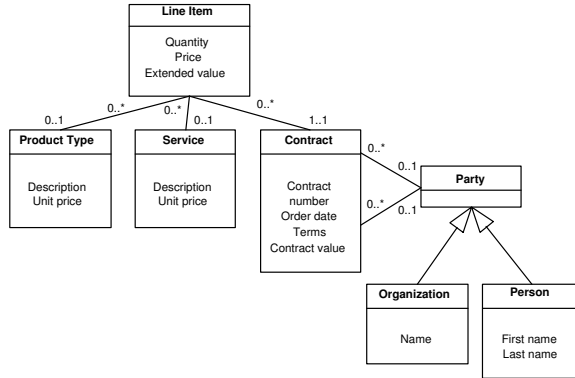


Figure 3.8: Products and services data model pattern. [97]

Semantic patterns

The term semantic pattern is not really an ‘application area’ or research field, but mainly a term to denote all patterns that aim to abstract from a representation structure or language, to an abstract modelling language. The word ‘semantic’ is used to illustrate that these are patterns trying to show the semantics of representation specific features. This is a very broad definition, which would include almost anything, like programming languages in general, because they abstract from hardware representation, but let us restrict this term to denote those abstractions that have actually been denoted patterns.

One example of such patterns is languages for abstraction. A high-level language may try to abstract away the specifics of some representation language, which can be a programming language, an ontology representation language etc. There are of course a large number of examples of such ‘semantic patterns’ but this is actually one area where there has also existed ontology patterns for quite some time, e.g. considered by Stuckenschmidt

and Euzenat [189] and Staab et al. [185]. Staab et al. [185] name these ontology patterns *semantic patterns* and in this way aim to abstract away the specifics of ontology representation languages. Stuckenschmidt and Euzenat [189] consider the additional problem of language expressiveness, the ontology patterns are expressed in an abstract language that needs to be at least as expressive as the ontology representation languages it is used with.

An example of a semantic pattern, described by Staab et al. [185], is the *locally inverse relation* pattern. This pattern aims at capturing the idea of restrictions on the inverse of relations, so they are only defined between certain concepts, i.e. restrictions on range and domain of the inverse relations. This is natural in for example object-oriented modelling, because properties and relations are defined locally in the object, but in ontology languages such as RDF everything is defined globally and relations exist independently of the concepts they belong to. In the notation of Staab et al. [185] this pattern, or language primitive of the more abstract language, is written $LOCALINVERSE(r_1, c_1, r_2, c_2)$, with r_1, r_2 denoting binary relations and c_1, c_2 their ranges.

Scripts and frames

The notion of scripts was proposed by Schank and Abelson [172] as a part of the early development in AI knowledge representation. However, scripts were also a cognitive theory of how human memory works. A script is a generalised prototypical situation, involving a set of primitive acts or events. Schank [171] additionally used the idea of scripts to introduce methods for interpreting and explaining new situations in terms of already encountered situations, much like the associative capabilities of the human mind. Scripts and primitive acts can be seen as a kind of patterns, representing typical and recurring situations that can be used to interpret new and unknown situations. The classical example of such a script, or pattern, is the restaurant script as exemplified by Schank and Abelson [172]. One scene of the script is the actual eating of the food, which can be described in a slightly simplified form as:

Scene 3: Eating

Cook TRANS food to waiter, waiter TRANS food to S, S INGEST food.

The primitive acts involved are the TRANS act, which is a transference of an object from one actor to another, and the INGEST act, which is the

ingestion of the actual food. The authors defined a relatively small set of primitive acts that could describe most common situations.

Frames, as proposed originally by Minsky [135], are similar to the scripts discussed above, although less focused on cognition, memory, and natural language and more on actual knowledge representation and usage by machines. Minsky defines a frame as data structure to store prototypical situations, with information about connections to related frames and procedural information on how use the frame and what happens if the current situation does not precisely match the frame. Minsky is usually attributed much of the later developments within frame-based systems and frame-based logics, that still survives as knowledge representation formalisms today. However, these early ideas are very much related to patterns as we view them today.

The FrameNet project, originating in linguistics using the so called frame semantics, is an ongoing effort to produce a complete set of semantic frames representing typical usages of English words, as presented by Baker et al. [11]. This research is not explicitly based on the type of frames and scripts mentioned above, but the general idea is quite similar.

Knowledge patterns

The term knowledge pattern denotes different things depending on the community. In the business related parts of the knowledge management (KM) community, a knowledge pattern can illustrate how to structure the management of knowledge within a company, by for example appointing teams of knowledge leaders or harvesting knowledge from workers, as defined by Davis [45].

In more technically oriented areas of KM, the term knowledge pattern can denote patterns for building knowledge bases, as used by Clark et al. [38], or reusable problem-solving methods, as used by Puppe [165]. Also Motta and Zdrahal [137] proposed reusable problem-solving methods, although they were not denoted patterns. The knowledge patterns of Clark et al. [38] are quite specific and are closely related to semantic patterns, rather than to knowledge-base architectures. An example of one part of such a pattern can be seen in Figure 3.9, expressed using Prolog notation. This part, or axiom, represents the concept of how free space in a container can be calculated. The whole *container pattern* would contain more axioms applicable to containers. The pattern can then be specialised, through a so called morphism, to represent a specific kind of container, such as a hard disk or a water bottle.

$free_space(Container, F) : -$
 $isa(Container, container),$
 $capacity(container, C),$
 $occupied_space(Container, O),$
 $F \text{ is } C - O.$

Figure 3.9: The 'free space axiom' in the container knowledge pattern. [38]

The knowledge patterns by Puppe [165] on the other hand lie somewhere between software architecture patterns and design patterns, but for problem-solving methods instead of software. They can be described as patterns used as common building blocks in problem-solving methods, and they can be expressed in a similar way as software patterns. An example of such a pattern is the *heuristic decision tree pattern*. The pattern consists of two phases, first the selection of a problem area and then the investigation of that area in order to find a solution. A UML diagram representing the first phase of this pattern can be seen in Figure 3.10.

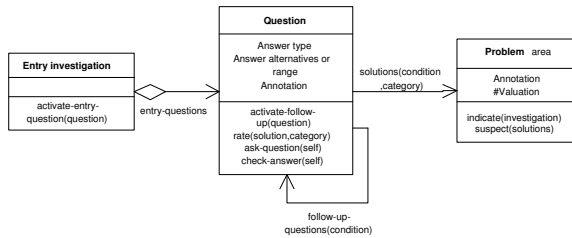


Figure 3.10: The heuristic decision tree pattern. [165]

Another category of knowledge patterns has been proposed by Sutcliffe [199]. In his book Sutcliffe describes a whole theory of reuse by patterns, or as he calls it the *domain theory object system model*. This is a model for formalising reuse both at the engineering level but also at the knowledge level. The theory includes object system models (OSMs) which are very similar to for example analysis patterns in software engineering. The differences lie mainly in the scope of the approaches, since Sutcliffe [199] presents a more theoretically well-founded approach and also covers more aspects of reuse, e.g. how to find, select and finally adapt and incorporate reusable parts.

The OSM library contains a hierarchy of patterns, with explanations and illustrations, that aim to be somewhat full-covering mainly for business applications. An example of an OSM is the *object inventory* OSM as illustrated using a UML diagram in Figure 3.11. The pattern can be used for

many different inventory-type application cases where resources are transferred, outbound to a customer or inbound to the owning company. The pattern also models replenishment of resources from suppliers.

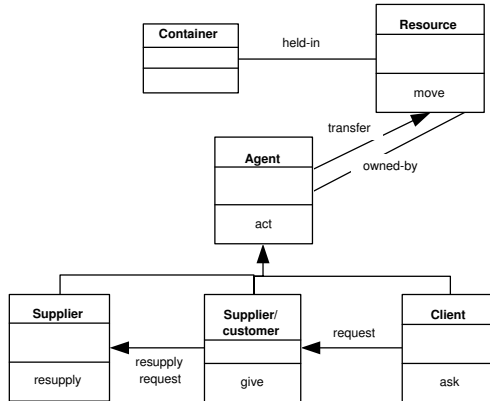


Figure 3.11: The object inventory OSM. [199]

Pattern mining and recognition

This kind of patterns mainly belong to the second category mentioned at the beginning of this chapter, i.e. patterns that are the product of, and not the means of, a process. There is a whole research field devoted to this, often denoted the pattern mining and recognition field. To this field could also be classified parts of the area of linguistic patterns. To find the linguistic patterns, pattern recognition techniques can be used. This is also true for some cases of programming language idioms and semantic patterns.

Just to remind the reader of the linguistic patterns described in brief previously, linguistic patterns are language dependent patterns that represent the structure of the language. Different constructs in a grammar can constitute such patterns. Linguistic patterns are used in parsers, translators, and also in systems for ontology learning as proposed by Wu and Hsu [220], Maedche and Volz [127], Faure and Nédellec [59], Hahn and Markó [92], Cimiano [34], and others. Faure and Nédellec [59] use linguistic patterns such as: $\langle verb \rangle ((\langle preposition \rangle | \langle function \rangle) \langle headword \rangle)$ to extract so called instantiated syntactic frames. Such an extracted frame could look as follows: $\langle totravel \rangle \langle subject \rangle \langle Bart \rangle \langle by \rangle \langle boat \rangle$, extracted from the sentence 'Bart travels by boat'. Previously, we described

other kinds of linguistic patterns, such as the patterns proposed by Hearst [98] for extracting subclass relations from text, and the more general OL patterns by Brewster et al. [25].

The idea of the pattern recognition field is to find regularities, and thereby patterns. Whether these patterns then are used for some other purpose, e.g. comparison to other patterns, machine learning, text parsing, or if the mere fact that they have been found is enough, is differing. One application area for pattern recognition is computer vision. To take pictures with a camera is easy but to make the computer 'understand' what is embedded in the pictures is hard. The researchers in this area try to describe patterns of for example faces in general, or a specific person's face, and then use pattern recognition algorithms to correctly match this pattern to the pictures where faces, or this particular face, appears.

Similarly there are researchers studying patterns in graphs. Many things can be represented as graphs, for example web usage logs, web page structures, chemical compound structures and ontologies. To find patterns, or re-occurring structures, in these graphs can give important information. In the web usage logs, or usage logs on ordinary computers, patterns can show frequent usage of the system, and possibly be used as a basis for intrusion detection mechanisms in the network and computer security areas. Finding frequent substructures in chemical compounds can mean finding toxic substances or simply determining similarities of compounds.

Examples of such subgraph discovery algorithms has been described by Inokuchi et al. [104], Kuramochi and Karypis [116], and Holder et al. [102]. There are also two kinds of algorithms, those that find frequent patterns in a forest of smaller graphs and those that find re-occurring patterns in one large graph. Other differences are the similarity measures used to decide if a substructure resembles another one, the purpose and output of the algorithm, this can be the discovered patterns or a compressed version of the original graph where the patterns where used for the compression process. A small experiment using such algorithms for pattern mining in ontology graphs was conducted by Thörn et al. [201].

Ontologies can also be used for pattern recognition tasks, this is mainly done in the information retrieval (IR) and information extraction (IE) areas. IR is an area that has developed fast, along with the web developing as a huge information source available around the world. Examples of IR applications can be to automate search and customisation of content on the web, and concepts such as semantic search has emerged. Semantic search using pattern recognition and graph patterns, in addition to keywords, can

be found in work by for example Zhong et al. [227]. To perform semantic search or other forms of advanced IR many different approaches have been developed, i.e. graph-matching algorithms to find 'semantic matches', such as the approach of Billig and Sandkuhl [18] and Yeh et al. [223].

3.2.3 Ontology patterns

Ontologies were mentioned in a few places in connection with different kinds of patterns above, but it is not until the last few years that ontology patterns have received proper attention. Early research within the scope of this thesis was at the forefront of defining types of ontology patterns and developing patterns for ontologies, as we shall see in coming chapters. Nevertheless, today there exist a number of different kinds of ontology patterns. These patterns have their origin mainly in the scripts, frames, knowledge patterns, such as the approach by Clark et al. [38], and the semantic patterns presented above, and have in addition been inspired by software patterns, both in naming and presentation.

Recently a typology of patterns was presented in a deliverable of the NeOn project by Presutti et al. [164]. Note that this is still largely unpublished research and presented at a later date than the typology that will be described in chapter 5 of this thesis. Therefore the typology presented in chapter 5 does not build on this parallel proposal, although the two typologies largely cover similar types of patterns. The deliverable is focussed on what is called ontology design patterns, which in the terminology of this thesis means 'patterns for ontology design', where design denote the development of an ontology. There is no direct correspondence to the notion of *design patterns* in software engineering, that focusses on the *design phase*. An ontology design pattern is by Presutti et al. defined as: "*An OP (OntologyDesignPattern) is a modeling solution to solve a recurrent ontology design problem. It is an dul:InformationObject that dul:expresses a DesignPatternSchema (or skin). Such schema can only be satisfied by DesignSolutions. Design solutions provide the setting for oddata:OntologyElements that play some ElementRole(s) from the schema.*" Where 'dul' and 'oddata' are namespaces, 'dul' being the DOLCE top-level ontology. An information object is a piece of information encoded in some language, and a design pattern schema is the description of an ontology design pattern. For details on the definition see complementing definitions of Presutti et al. [164].

The typology can be illustrated as seen in Figure 3.12, using a UML notation. The main categories of ontology design patterns are lexico-syntactic

patterns, structural patterns, content patterns, presentation patterns, reasoning patterns, and correspondence patterns. The typology is extensive and covers many areas of ontology engineering, but it is not clear if it actually aims at being complete. Neither do the authors provide any rationale for choosing exactly these categories and the differentiating notions and level of generality in the 'pattern taxonomy' are mixed, and neither well motivated nor well explained in the referred publication. The authors seem to take a purely practical perspective and simply list every case where patterns can be identified at present. Nevertheless, the categories cover many interesting aspects of ontology engineering.

Lexico-syntactic patterns are patterns that connect language constructs, mainly in natural language, to ontological constructs. Lexico-syntactic patterns, such as the ones proposed by Hearst [98], was mentioned previously. Presentation patterns include naming conventions and annotation schemas for ontologies, and are to be seen as best practices how to present and document ontologies and their elements. Reasoning patterns are common ways of doing reasoning over ontologies, such as classification or normalisation as defined by Vrandečić and Šure [215]. Correspondence patterns can be reengineering patterns or mapping patterns, where reengineering patterns focus on correspondences between different formalisms for transforming some source model, even a non-ontological model, into another ontology representation. Mapping patterns on the other hand are related to the possible correspondences between two or more ontologies, as investigated by Scharffe et al. [173].

Structural and content patterns are mainly addressing actual modelling issues within ontologies, both on the logical language level and a more abstract design level. Structural patterns deal with the logical structure of the ontology, the expressiveness of the modelling language and issues related to this, both on a local level, as for logical patterns, and a global level, for architectural patterns concerning the structure of a complete ontology or ontology module. To such logical patterns we could refer the semantic patterns discussed previously in this chapter, as well as the OWL-specific design patterns that solve specific expressiveness issues in OWL that were developed by the W3C. An example of such patterns is the best-practice for representing classes as property values, see the W3C website [3]. There are also domain-specific logical patterns, such as presented by Reich et al. [167] for the biomedical domain. These patterns aim to implement certain logical functionalities of the ontologies, such as information encapsulation and part-whole hierarchies.

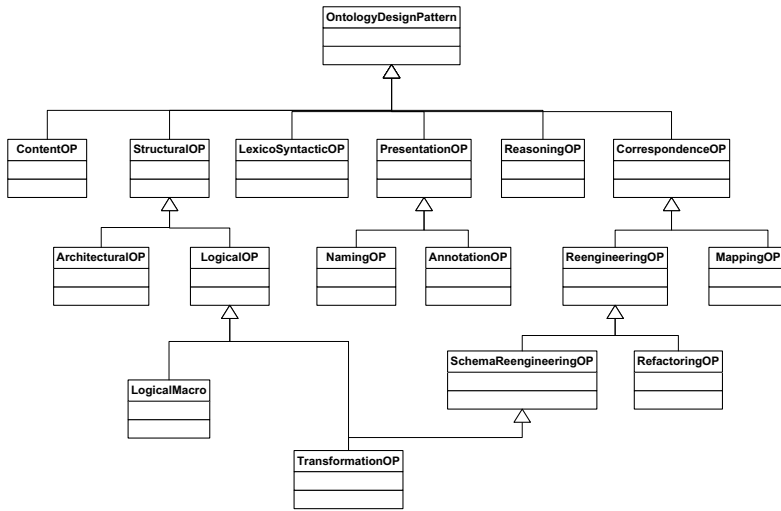


Figure 3.12: The typology proposed by Presutti et al. [164].

Content patterns are related to structural patterns, because they also restrict the structure of the model, but in addition they provide solutions for modelling specific concepts. One might say that a structural pattern, or at least a logical pattern, can be viewed as a logical template specifying a certain logical structure, but with an empty signature, i.e. any concrete concepts and relations may be used to realise the structure. A content pattern on the other hand will additionally contain actual concepts and relations that may be specialised when the pattern is used. Such patterns have been proposed by Gangemi et al. [74], and further explained by Presutti and Gangemi [163] and Gangemi and Presutti [81].

A content pattern is only guaranteed to be valid when specialising the concepts and relations, not necessarily for generalisation. For example, for a pattern containing the concepts 'project leader' and 'project team' connected through the 'leads' relations the concepts can be generalised to 'leader' and 'team' and the relation is still valid, in this case even without changing its label. Now, consider the case of the two concepts 'board' and 'board member' with the specific relation 'secretary of board'. When generalising to 'group' and 'group member' there is no intuitive way of generalising the secretary-relation to general groups. Specialising the board-pattern we can arrive at an ontology containing 'company board', 'company board member' and the 'secretary of the board' relation is still valid.

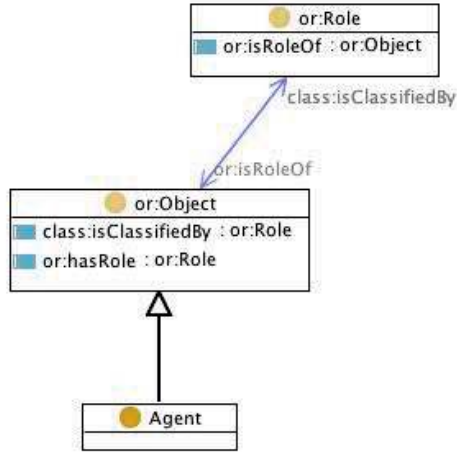


Figure 3.13: The agent role content pattern. [164].

An example of a content pattern is the *agent role pattern* that can be viewed in Figure 3.13, using the UML notation of the ontology engineering environment TopBraid Composer[†]. The pattern states that objects are classified by their roles and that a role is a role of certain objects. This part is actually the object role pattern, a more general content pattern that has been specialised into the agent role pattern seen here, hence the namespace prefix *or:* that illustrates the different namespace of the imported pattern. In agent role the objects involved are agents, living or artificial. Patterns can also overlap, whereas objects may appear in many patterns for different purposes. This specific pattern has its origin in the DOLCE Ultra Light top-level ontology but is represented as a self-contained ontology.

Presutti et al. [164] additionally propose a set of operations that may be used on patterns, such as composition of patterns, cloning of elements, and specialisation. The main idea is that the content patterns are imported, as reusable building blocks implemented primarily in OWL, into the ontology to be constructed, specialised, expanded and composed to fit the case at hand.

[†]<http://www.topquadrant.com/topbraid/composer/>

3.3 Case-based reasoning

Case-based reasoning (CBR) is, according to Aamodt and Plaza [4], aiming to use previous experiences to solve new problems. It is worth noting that this is a similar idea compared to patterns, and it can also be viewed as a specific reuse methodology, which is the reason for including it in this chapter.

CBR is generally depicted as a cycle of four phases, i.e. retrieve, reuse, revise and retain, all using the stored knowledge in the central case base. In addition to the stored cases, experiences or partial solutions from previously solve problems, the case base might also contain general domain knowledge. A specific branch of CBR is textual CBR (TCBR) that focuses on approaches using natural language texts. Weber et al. [218] describe four open research questions of this area, containing how to get from a textual representation of a case to a structured representation, and how to automate approaches. As we shall see later in this thesis our approach has been inspired by CBR and addresses similar research questions, in the setting of semi-automatic enterprise ontology construction. Recent developments of CBR also use 'soft' computing, as noted by Pal and Shiu [155], in order to simulate the human decision-making process.

A number of variations of case-based reasoning were noted by Aamodt and Plaza [4]. A 'true' case-based reasoning method is distinguished by the complexity of the case structure and stored information, and also by the ability to modify and adapt the retrieved solutions to a new situation. Although an extensive literature search has been conducted, no approach using CBR for ontology engineering has been found. Some related approaches can be noted though, most similar seems to be the concept map extension and knowledge acquisition techniques applied by Leake et al. [117] that uses previously created concepts maps as past cases when constructing and extending new ones. That approach store complete concept maps as cases, and does not use the notion of patterns.

3.3.1 Benefits of CBR and when to use it

A summary concerning some of the benefits and drawbacks of using case-based reasoning is presented by Pal and Shiu [155]. Some of the benefits listed are to reduce the load on knowledge acquisition tasks, learning from the past, reasoning with incomplete, imprecise or insufficient information, and reflecting human reasoning and means of explanation. Based on these

benefits the authors suggest some guidelines as to help determining when CBR is the right methodology to choose [155]. The guidelines are expressed as 5 questions to ask when considering to use, or construct, a CBR-inspired system:

1. Does the domain have an underlying model?
2. Are there exceptions and novel cases?
3. Do cases recur?
4. Is there significant benefit in adapting past solutions?
5. Are relevant previous cases obtainable?

If there is no clear underlying model that can be completely understood, there are many exceptions to the rules that govern the domain, but similar cases still reoccur, then the problem might be suitable for a CBR solution. Additionally there must be past solutions available and it must be clear that it is more beneficial to reuse these than to start over.

3.4 Chapter summary

In this chapter we have presented general notions of reuse and, in particular, patterns. Patterns constitute a specific way of reusing encoded experience, whether extracted from existing data or engineered from best practices. A large amount of research exist concerning patterns in many different areas, and patterns have also been proposed for knowledge representation and ontologies. Below we summarise the two focus areas, ontology reuse and ontology patterns.

3.4.1 Ontology reuse summary

Ontology repositories exist and are usually equipped with search and browsing capabilities. Nevertheless, selection and reuse of large ontologies is not straight forward. There exist ontology search engines and ranking schemes, assisting the user in finding and selecting ontologies on the web. However, it is still up to the user to formulate the correct query, to realise when something interesting has been found, to evaluate the details of the ontology to be reused, and to adapt and reuse it.

Reuse of top-level ontologies, consisting primarily of general common-sense knowledge, has been performed in many ontology engineering projects, in such cases it usually remains to specialise an further develop the ontology, adding domain-specific knowledge. When reusing more specific ontologies, documentation, metadata, and annotations are sometimes scarce or none-existing, whereas the reuse task can be quite difficult. Assuming that a set of suitable ontologies have been retrieved, there exist numerous approaches for matching, and aligning or merging such ontologies. However, most approaches are focused on merging ontologies, rather than integrating ontologies of different coverage, and assume some overlap, or at least close 'connection' between the ontologies to be matched. Recent approaches are often automated, to some extent, and produce reasonable results when applied on typical matching tasks, such as the OAEI-datasets.

Modularisation of ontologies has already been adopted as a general 'good practice', at least by the research community. The past few years have seen the emergence of several language extensions, and other approaches to modularisation, including rules and guidelines on how to distinguish a 'good' module and how to extract modules automatically. However, there is a lack of guidance as to what content a module should or should not contain and how to select and adapt modules at the time of reuse. Modularisation provides a way of reusing manageable, and to a large extent, self-contained 'pieces' of an ontology, still there are no specific guidelines how to find and reuse modules, and there is a lack of tool support for such tasks.

The existing methods for finding, matching, selecting, and reusing ontologies are not really suited for small general patterns, although content patterns can be represented and treated as small ontologies. Trying to find a general content pattern through an ontology search engine a user would have to search for quite general terms in order to find the pattern, while ideally the user would be able to search for a specific term that he wishes to model and still get the general pattern as a result. For example, a user that wants to build an ontology about students and researchers that teach at a university might search for terms such as 'student', 'professor' and 'teacher'. A suitable pattern for modelling the roles of the people involved in this scenario could be the agent role pattern mentioned previously, but that pattern would not be retrieved by a standard search engine using those search terms. A person reviewing the pattern easily realises that student, professor and teacher are roles that people can take in different situations, at the university, and that people are a kind of agents. However, such matching is usually not done by approaches for finding and selecting ontologies.

On the other hand this is what ontology matching is focussing on. Most approaches focus mainly on finding equivalences, but many approaches also successfully propose subsumption relations between concepts of the different ontologies. Ontology matching also takes into account the richer structure of the ontologies involved, compared to the keywords used for search and retrieval mentioned above. Ontology matching techniques are essential for finding and retrieving content patterns, with respect to a 'query' which is in essence an ontology. However, none of the existing matching approaches or tools are tailored to the specific case of pattern matching, i.e. matching ontologies that are known to most often be of different size and different levels of abstraction. Nevertheless, specific techniques for ontology matching were reused when developing the details of the OntoCase method, as we shall see later in this thesis. These are combined with a ranking scheme inspired by the ontology search engines present today.

3.4.2 Ontology pattern summary

Most of the research on ontology patterns has developed during the last couple of years. As far as we are aware only the approaches of semantic patterns and patterns specific to biomedical ontologies were present at the beginning of this decade. With the emergence of the Semantic Web, that has brought ontologies into more or less common software practise, also the ontology pattern development has caught speed. As an example it can be mentioned that during the past year several new types of patterns have been proposed in literature, and ontology engineering environments are starting to provide support for modular and pattern-based ontology development. However, ontology patterns is a field in its infancy. An online pattern catalogue for content patterns has been launched[‡], but so far it contains only a few general content patterns, and the community is far from the rigour of the software patterns community where conferences and workshops are devoted entirely to developing and discussing software patterns and best practices.

Currently, lexico-syntactic patterns are widely used in many areas, also ontology engineering and specifically ontology learning. This is additionally true for some of the structural patterns, e.g. logical patterns addressing some specific language such as OWL or F-logic. These have for example been widely used to provide graphical user interfaces to ontology engineering tools, where the user can for example edit logical axioms in graphical

[‡]<http://www.ontologydesignpatterns.org>

manner. Content patterns and correspondence patterns have been proposed and are now being developed in several research projects. With respect to the rest of the ontology patterns discussed in this section, e.g. architecture patterns, reengineering patterns, and presentation patterns, they seem to be still on the level of ideas.

The current state could be described as a 'kick-start' for the ontology patterns community. Since the Semantic Web is now partly being realised, and ontologies are widely used in this context, methods suited for non-ontology engineers, such as web developers, for constructing ontologies are needed. Patterns as encoded best practice might be one way of meeting this need, although other methods and tools are also needed. However, method and tool support for actually using the patterns are lacking, as well as an agreement on their definitions, characteristics, types, purposes, and benefits. The research presented in this thesis has grown along with this rapid development of the ontology pattern community, and provides one piece of the puzzle in order to put patterns into use and show their benefits for real world cases. The main focus of the rest of the thesis is on what was here denoted ontology content design patterns, and OntoCase aims to provide some initial tool support for selecting and reusing such patterns.

Chapter 4

Method and evaluation strategies

Practically oriented research in computer science is often hard to characterise in terms of research methodology. There are few formalised research methodologies proposed for engineering-related approaches in computer science. In the parts of computer science closely related to mathematics and mathematical logics this may be a perfectly valid situation, since these disciplines inherently include high rigour and requirements to prove results mathematically, whereby other methods may be superfluous. At the other end of the scale, in fields of computer science more related to the social sciences like business informatics and human computer interaction (HCI), methods are more well-developed and firmly established, since these fields rely on the study of human beings rather than artificial objects. In between these two extremes there is however a gap. Engineering-style computer science research is performed in many areas of computer science but neither the formal mathematical proofs nor the human centred social science methods suffice in these cases.

In this chapter we first give a general philosophical overview of research methodologies that are considered relevant for this work. This is then compared to more social science-related methodologies, such as design research in informatics, and from this comparison a general outline of the research philosophy for this thesis is sketched. Subsequent sections present specific techniques used for this research are presented, such as evaluation methods for ontology evaluation, and finally the practical realisation of our research is outlined.

4.1 Research methods

Traditionally, science has been about discovering some kind of 'truth' about the world, although the existence and nature of such 'truths' have been frequently debated, as for example discussed by Chalmers in his book on research philosophy [32]. Although scientific results are something that commonly is thought of as 'proven' knowledge, the nature of the proof differs greatly from one field to another. The nature of the problems studied differ also between fields, what is considered a relevant and valid problem in one field might not be accepted in another field.

As mentioned above, engineering-style computer science is usually not able to completely prove results mathematically or logically. Such results are mere proposals of solutions that are in some specific sense 'better' than previous ways to solve the problem, but they usually cannot be proven optimal. Sometimes the solution cannot even be proven to be a valid solution by other means than to actually provide an example implementation, a so called *proof of concept implementation*. The focus of such research is on methods for showing that the proposed solution actually works and is better than previous proposals. This means that the methodological focus is not so much on the process of producing the solutions but on evaluating the feasibility of the solution and establishing an amount of improvement compared to previously proposed solutions. One way of showing improvement is by conducting experiments.

In contrast, one could put such solutions in a larger perspective, for example as tools to assist humans in solving a specific kind of problem. In this case, methods from social sciences may be applicable. In this research we have chosen to study the ontology construction process from a technical perspective. Without, at this stage, considering the ontology engineer, the 'end user', and user interaction as a factor in experiments and evaluations. It is by no means our intention to diminish the importance of viewing the user as an essential success factor, it is merely a matter of limiting the research focus. Hence we do not include the development of a graphical user interface for the proposed methods, or user studies of the method put into a larger perspective. Due to this fact we will focus on research methods related to engineering-style research and experimentation in the rest of this chapter.

4.1.1 Experimentation in computer science

What is often stated to differentiate development of ad-hoc solutions from scientific research, and scientific results, is performing experiments. Experiments can show characteristics of a proposed solution, or prove the excellence of whatever has been developed. As discussed by Basili [14] both deductive and inductive research paradigms exist. While the development of a model followed by analytically proving that it is correct follows the deductive paradigm, the proposal of a model and the subsequent provision of sufficient evidence that the model is correct, e.g. through proof of concept implementations and their evaluation, conforms to the inductive paradigm. The conclusion that the method is correct and better than another method is in this case induced from a set of evidence, but cannot be mathematically or logically proven. It is especially within this inductive paradigm that carefully designed experiments are needed.

Basili [14] proposes that there is a difference between evolutionary experiments and revolutionary experiments. Where evolutionary experiments try to show that some solution is in some sense better than all the previous ones, revolutionary experiments try to show that a completely new solution solves some previously unsolved problem. Zelkowitz and Wallace [224] discuss different ways of experimenting, based on three main categories of collecting data:

- Observation
- Historical collection
- Controlled experiment

The first category involves experiments such as case studies and field studies, where the main data collection is done by observing the object of study in a 'real world' setting, generally with a minimum of interference by the observer. Historical data collection is similar, but is instead based on data already collected in some previous situation. Finally, controlled experiments are what we usually denote by 'experiment', where the setting and possibly some of the variables studied are controlled by the researcher. Controlled experiments are usually replicated in order to isolate different variables and to achieve statistical validity.

Zelkowitz and Wallace [224] propose four (overlapping) kinds of controlled experiments that can be applied in software engineering research:

- Replicated experiment

- Synthetic environment experiment
- Simulation
- Dynamic analysis

In replicated experiments the intention is to control all variables that can affect the outcome and replicate the experiment so that the hypothesis can be verified with statistical significance. If the object of study is complex and contains many different variables, such experiments may be very costly to perform, at least in a 'real world' setting. A way to reduce this complexity is to use a synthetic environment, where the experiment is instead set in a smaller but artificial environment. More variables can then be controlled by the researchers, and the focus put on the variables of study, but at the expense of external validity and potential to generalise experiment results. Simulations completely remove the natural setting and replace it with an artificial model of reality. This reduces the complexity even more, but since it is not clear how well such a simulation model reflects reality it introduces even more uncertainty in the results. Finally, dynamic analysis studies focus on some product of the object of study, for example the produced software in the case of software engineering, and suggests performing analyses on this in order to derive conclusions about the object of study, e.g. the software engineering process.

For the research in this thesis the objects of study are a method for ontology construction and the ontologies produced by such a method. Since the method is not complete, in terms of covering the complete process, and the implementation does not contain graphical user interfaces, the application of the method in 'real world' settings will not be possible. Consequently, observational experimentation methods are ruled out, as well as historical methods since this is a new method and has not yet been applied. Instead, focus will be on controlled experiments, mainly replicated experiments in synthetic environments. The focus will be on studying the outcome of the method, the ontologies, rather than measuring variables of the process itself. Thereby, the experiments are related to dynamic analysis as explained above, but the analysis is performed in a laboratory environment, since the ontologies have not been used in any real-world applications, which means that this is a combination with the so called synthetic environment category.

4.1.2 A broader perspective on research method

During the 20th century debates arose in several design-oriented fields and eventually brought many such fields into the realm of 'science' and 'research', for example by introducing proper research methods. One such example is architectural design, see for example the summary by Cross [41], which was from the beginning considered as merely a craft and a subjective expression of the skill and inventiveness of the engineer. One research method that proposes to use the approach of design research for information technology research is the method described by March and Smith [128]. This method has been further developed for example in business informatics, where a more recent method for design research has been proposed by Hevner et al [100].

A general idea behind design science is to extend the frontier of science by constructing (designing) and evaluating new and innovative artefacts. The complete process of research can be seen as a cycle of two phases; producing artefacts and then evaluating them. In addition to the design science approach, March and Smith [128] describe the natural science approach of also theorising and justifying the theories. An important part of design science is the artefacts produced. Artefacts can be of four different kinds, according to March and Smith [128]:

- Constructs, e.g. new concepts forming a new vocabulary.
- Models as propositions about constructs, e.g. new representations.
- Methods, e.g. a new algorithm or methodology.
- Instantiations, e.g. a prototype system.

In addition to a set of artefacts, Hevner et al. [100] describe a set of guidelines that should be considered when conducting the two steps of design science; building artefacts and evaluating them. The guidelines can be summarised as follows:

1. The research should produce an artefact.
2. The artefact should be a solution to an important problem.
3. Evaluate utility, quality and efficacy of the artefact.
4. The artefact should provide a clear research contribution.
5. Apply rigorous methods in both design and evaluation phases.

6. Search for effective solutions without considering all possible solutions.
7. Present the research to both technology-oriented and non-technology oriented audience.

The first two guidelines state the main focus of design science, to create an artefact as an effective solution to some important problem. Since design science is mainly used in business informatics research this is translated into the requirement of solving some important business problem, existing in real-world enterprises. For the research to be relevant and rigorous the third guideline states that the artefact has to be properly evaluated. The fourth guideline adds the aspect of novelty and innovation. The artefact not only has to solve the important problem in a sufficient way, it also has to contribute to the overall body of knowledge of the research area. The following two guidelines are constraints on how the research is carried out. Rigorous methods have to be used, both for constructing the artefact and for evaluating it. The goal of design science is not to find the optimal solution for the problem at hand, but merely to find *one* new and innovative solution that fulfils the previously stated criteria. Finally, the research should be communicated both to the research community and to external parties that might benefit from it.

Despite the fact that this thesis does not apply design science as a whole, some inspiration from the above framework proved useful. The intention of our research is to produce all four kinds of artefacts. Examples of produced constructs are the categories of ontology engineering patterns as a vocabulary for discussing such patterns. Models and methods are the patterns themselves and the OntoCase framework where they are used, and an instantiation is produced as OntoCase is implemented as a proof of concept implementation.

The guidelines presented above are only relevant to a certain extent, but may very well act as a starting point for discussing the realisation of this research. Guidelines 1, 4, and 6 were the major focus of the research, i.e. to produce an artefact that solves the problem considered and provides a clear contribution to the research community. Guidelines 3 and 5 were considered, but with main focus on the artefact evaluation and not the design part. This means that no rigorous software development method was used to produce the proof of concept implementation, but that the main effort was instead focused on applying rigorous evaluations of the result. The complete guideline 3 was not applied either, since our focus was on quality and efficacy of the solution and not utility; neither the proposed method nor the

ontologies were applied in any real-world setting outside research projects. Guideline 7, considering communication, was only partially followed since our focus was on the research community and not other interested parties. Guideline 2 was not considered, since there is no proof as far as we are aware that ontology engineering as a whole solves any important business problem. We have not attempted to find such evidence, instead we rely on the collected body of research in the field that shows the technical benefits of ontology-based systems.

Evaluation

Evaluation is an important part of all research, but especially important in design science. Since this kind of research does not propose a theory of the nature of the world nor an optimal solution to the problem at hand, there is great need to show that at least the solution proposed does in fact solve the problem, it is actually useful, and the quality of the result is good enough. If these requirements are not fulfilled any solution, however naive or low quality, can be proposed as a research result. In addition, the evaluations are usually seen as incrementally feeding the design process itself, so the two phases (design and evaluation) are actually iterated throughout the complete research project. In our case, two main iterations of the research process were conducted, each including the design of a set of artefacts and subsequent evaluation of the artefacts.

When evaluating artefacts Hevner et al. [100] list five kinds of methods for evaluations:

- Observation
- Analysis
- Experimentation
- Testing
- Description

Observational evaluations include, for example case and field studies, where the artefacts are put in their intended environment and the results are observed and analysed. Analytical evaluations involve studying the properties of the artefacts, such as complexity, performance, and fit to overall architecture. Experimentation involves simulations and controlled experiments,

where the artefacts are put to use, but in a constructed environment. Testing intends to identify defects in the artefacts through functional or structural tests, trying to verify the functionality and the expected properties of the artefact. Finally, the utility and efficacy of the artefact can be argued through theoretical arguments based on existing research or constructed scenarios.

To show the utility of the proposed solution observations seems most appropriate, to put the artefact to use and analyse the results. This is not always possible in a research setting and thereby experimental and descriptive evaluations have to be used instead. In our research, the artefacts have not been observed in real-world settings, but merely used in controlled experiments and their utility established through informed arguments and described scenarios.

Studying efficacy involves determining that the artefact actually achieves the desired and intended effect, that it sufficiently solves the problem. Observations can be used to show this, and properties can be measured to make sure the problem is sufficiently solved. Analytical methods can partly establish the efficacy of the artefacts, together with experiments and testing, to determine that the correct effects are achieved. As for the quality of the artefacts, this is closely connected to efficacy, but in addition some more testing can be used. We have applied mainly analytical methods and experiments of different kinds, for example to determine the efficacy of the pattern ranking algorithm.

In addition to evaluating the artefacts and the research results produced, there can also be a need for evaluating the method used to arrive at those results. As stated earlier, rigour is an important part of any design research, but rigour must be evaluated, too. In order to evaluate the method used to draw conclusions about the validity and applicability of the results the guidelines of design research can be used as criteria, but to practically perform such an evaluation they need to be broken down further into more specific questions and criteria.

Practical methodology

Actual practical process descriptions for carrying out research in computer science and information technology are rare, but again we can find inspiration in design science for business informatics. In this field, for example Nunamaker and Chen [148] and Hasan [96] propose similar methodologies for systems development as an information systems research method. The

methodologies consist of five stages, while Nunamaker and Chen propose a more linear process Hasan relaxes it and introduces iterations and overlap:

1. Construction of a conceptual framework.
2. Development of a system architecture.
3. Analysis and design of the system.
4. Building of the system.
5. Observation and evaluation of the system.

Using these five steps we note that the first four actually detail the design phase of design research, while the fifth step is concerned with the evaluation phase. In the first four phases, different artefacts are produced. This general framework inspired the methodology that was applied as the research process in this thesis, using two iterations. These iterations, and the practical steps taken, are described in section 4.3.

4.2 Evaluation methods

The evaluation methods used when validating and testing the research results against the problems posed by the research questions is of great relevance to the rigour of the research. Several different kinds of experiments and evaluations have been conducted, and the general ideas of those evaluation methods are described below. We can identify two main evaluation directions to deal with, the evaluation of the research itself, and the detailed evaluations of the different artefacts that need to be performed in each iteration of the research process. Below we first deal with some general notions that can be used for evaluating the research process, and then we discuss the detailed methods used for evaluating artefacts.

4.2.1 Research evaluation

To study the research performed and evaluate it as such, the set of evaluation criteria suggested by Burstein and Gregor [30] is relevant. The intention of evaluating the research as such is mainly to establish that the research is relevant, has been conducted with sufficient rigour and provides a real contribution to the field. Burstein and Gregor suggest five types of criteria for evaluating research efforts:

1. Significance of the study
2. Internal validity
3. External validity
4. Objectivity and confirmability
5. Reliability and auditability

Below, each of these criteria will be treated and a set of questions listed, which are slightly adapted versions of the questions proposed by Burstein and Gregor [30]. These questions are revisited at the end of this thesis in order to show the strengths and weaknesses of this research.

Significance is concerned with the relevance and the contribution of the research, as well as the quality of the research results. For the research to be significant it has to provide a better solution to the problem than whatever existed beforehand in the research community, thereby also providing some new knowledge to the research field. To determine the significance of our research we pose the following questions:

- Who are the beneficiaries of these results and what relevance do these research results have to the beneficiaries' problems?
- What are the main research contributions of this research to the field?
- How is this solution better than previously proposed solutions to the same kind of problems?

Internal validity is concerned with the evidence put forward to support the research conclusions and the credibility of the arguments made. Credibility is supported by things such as rigorous experiments and good causal arguments that support the conclusions, but can be reduced if negative evidence is ignored or reasoning is not explained in detail. Questions that should be asked with respect to internal validity are:

- Does the method actually solve the problem? Completely or only partly?
- Does the method meet all its requirements?
- Is the method compared to alternative methods, and are results from such comparisons presented?

- Is negative evidence sought for and presented?
- Does the evidence support the claims for the research results sufficiently?

The notion of external validity is connected to the generalisability of the results, in the sense that if the external validity is low then the results can only be shown to hold for the specific cases tested. In order for research results to be applicable to a larger class of problems the external validity needs to be high. Questions to be asked to confirm external validity are:

- Is the method based on existing theories? Does it confirm and support those theories?
- Are assumptions, personal opinions and biases clearly stated and analysed?
- Are the procedures and methods used clearly described?
- Are there limitations to the evidence collected thus suggesting limited generalisability?

The final two criteria can actually be contained within the previous three statements, but Burstein and Gregor [30] suggest treating them separately due to their great importance for research quality. The research needs to be as objective as possible and all biases and subjective assessments need to be made explicit. It is also a characteristic of good research that it is repeatable, or at least confirmable, by other researchers. This is closely connected to reliability and auditability, in order to establish that things have been done with reasonable care. Questions regarding the objectivity and reliability are:

- Are the research questions clear?
- Are basic constructs and basic assumptions clearly specified?
- Are all methods, experiments and procedures described in detail?
- Is it clear what data is used for testing and experiments?
- Are all assumptions made explicit during the process?
- What biases do the persons and methods involved in the research have?

- Is the status and role of the researcher explicitly described?
- What limitations are imposed by biases or subjectivity in the research results and conclusions?

4.2.2 Result evaluation

The definitions, characteristics, and typology of ontology patterns were mainly evaluated through peer review of papers where these notions have been published, and by presentations at research seminars. To some extent these theoretical results were also evaluated through analytical methods, by checking for coverage and soundness of the definitions and classifications, and by descriptive methods, analysing scenarios and motivating when to use the different kinds of patterns. The implementation of the OntoCase method was not evaluated as an artefact in itself, instead the quality of the method is analysed based on the results produced, i.e. the produced ontologies.

Ontology evaluation methods and measures were used for studying the quality and characteristics of the output of the proposed OntoCase method. Gangemi et al. [77] describe an overall framework intended to cover all aspects of ontology evaluation and selection criteria. The framework contains a meta-ontology describing the elements that can be evaluated, and an ontology of ontology evaluation and selection methods. Gangemi et al. also try to structure the area of ontology evaluation, and propose three levels of evaluations based on a semiotic perspective, i.e. depending on the view of the ontology. The three levels are denoted:

- Structural evaluations
- Functional evaluations
- Usability evaluations

If the ontology is considered as an information object, then structural evaluations can be applied based on the syntax and semantics of the ontology as it is represented. Such evaluations take into account the syntactic structure of the ontology but also the semantics, e.g. in the form of formal definitions. However, at this level the ontology is considered as a syntactic unit, outside of its intended context. If the ontology, on the other hand, is considered as an information object *and* a language, i.e. the intended conceptualisation, functional evaluations can be applied. Functional evaluations are connected

to the purpose of the ontology and the context where it will be applied, e.g. it is checked against requirements and intended tasks. Finally, the ontology can be treated from a usability perspective, where the ontology is treated as a semiotic object intended to carry some specific meaning. At this level evaluations concern understandability and reusability of the ontology, as well as user satisfaction and efficiency and efficacy of the ontology application.

Hartmann et al. [95] provides another overview of the state of the art in ontology evaluation. In addition to the above, they identify three different stages of evaluation, namely evaluating an ontology in its pre-modelling stage, its modelling stage, and after its release. The first stage involves evaluating the information the ontology will be based on, the second stage includes checking the ontology correctness while building it, and the final stage involves comparing existing ontologies and monitoring ontologies in use. These stages can be connected to the levels of evaluation suggested by Gangemi et al. [77], since some types of evaluations can be performed in only one stage of the ontology development life cycle while others may fit several stages.

Evaluation methods for the pre-modelling stage include methods for assessing relevance of text corpora and evaluation of ontology requirements. Such methods are not methods for evaluating ontologies, but rather evaluation methods that can be used to evaluate knowledge sources in the initial stages of ontology development. Structural evaluations can be applied both during and after ontology construction. Such methods include measures of ontology characteristics, consistency checking, and structural evaluations of the taxonomy. Functional evaluations are mainly performed after the ontology is developed, and can involve comparisons to 'gold standards', correctness and suitability evaluations performed by domain experts, and testing the ontology against the requirements. However, such evaluations can also be used during construction, e.g. for testing modules of the ontology. Usability evaluations can also be performed during and after construction, although they are more common after the complete ontology is ready. Usability evaluations include checking the understandability of the ontology, evaluating annotations and comments, analysing how it is perceived by domain experts, and to evaluate how well the ontology actually performs in the intended system.

Gangemi et al. [75] detail the categories and also propose general evaluation principles. In their framework the measures are grouped under 9 principles, representing sets of ontology characteristics:

- Cognitive ergonomics.
- Transparency, i.e. explicitness of organizing principles.
- Computational integrity and efficiency.
- Meta-level integrity.
- Flexibility, i.e. context-boundedness.
- Compliance to expertise.
- Compliance to procedures for extension, integration, adaptation etc.
- Generic accessibility, computational as well as commercial.
- Organizational fitness.

Cognitive ergonomics can be evaluated based on 8 parameters, where four affect the principle positively and four negatively. Gangemi et al. claim that depth, breadth, tangledness and the number of anonymous classes all affect the cognitive ergonomics negatively if their values increase. On the other hand, the class-property ratio, interfacing (user interface-related annotations), presence of patterns, and number of annotations all increase the cognitive ergonomics as their values increase. The intuition is that a less complex ontology, with high number of properties, that is well annotated, i.e. commented and explained, will be easy to understand and to use.

Transparency can be evaluated based on 9 parameters, that all increase the transparency as their values increase. The parameters are, modularity, axiom-class ratio, patterns, specific differences, partitioning, accuracy, complexity, anonymous classes, modularity design. For a detailed account of all parameters, see the report by Gangemi et al. [75]. The intuition behind 'transparency' is that the more formal definitions and axioms, the more clear is the ontology in terms of its conceptualization, making all assumptions explicit.

Generic accessibility is based on four parameters, accuracy, annotations, modularity, and logical complexity. An increase in logical complexity will generally reduce the accessibility, while the other three parameters work in favour of accessibility. The intuition is that if the ontology is accurate with respect to its requirements, contain annotations to explain any implicit assumptions, and is modular, it will constitute an ontology which is easily accessed and used. Organisational fitness is similar but focuses more on a

particular organisation, and the coverage of that specific domain. Compliance to expertise in turn means that the ontology is accurate with respect to the expertise of one particular user.

In this thesis we assume that the input to our method is given, e.g. the set of documents and the pattern base, whereby evaluations that are of interest concern the ontology during and after its construction.

Structural measures

The idea of continuous evaluation within the ontology construction process is quite natural. This is comparable to software engineering, where the code modules are tested and validated during construction, before composing the system and evaluating it as a whole at the end. For such tasks mostly structural evaluations are proposed. In this section we give an overview of some of the most common structural measures.

Structural methods can be used for evaluating the syntax and semantics of ontologies during construction, as mentioned above. There are simple methods that measure cohesion of ontology concepts and modules, such as the measures proposed by Yao et al. [222]. The approach is simple but provides an intuitive idea of how the ontology is organised, by computing average values of, for example the number of taxonomic relations per concept. Gangemi et al. [77] propose several such measures in their survey, such as breadth, depth, tangledness, fan-outness, cycle ratio and density. In line with these suggestions an ontology can also be characterised based on simple statistics, such as size.

There are guidelines for manually assessing the correctness of the ontology, in particular the taxonomy. One approach is described by Gómez-Pérez [84] and Gómez-Pérez et al. [85], focussing on concept definitions and taxonomic relations. The guidelines are quite brief, thus some expert knowledge concerning ontology engineering is definitely needed in order to perform the evaluation, but also domain and task knowledge is needed in order to evaluate criteria such as completeness. The idea is to spot and correct common development errors, such as circularity in the taxonomy, or an instance belonging to two disjoint concepts.

The main types of errors to be considered according to Gómez-Pérez [84] are the following:

1. Inconsistency errors
 - Circularity errors

- Partition errors
 - Semantic inconsistency errors
2. Incompleteness errors
 - Incomplete concept classifications
 - Partition errors
 3. Redundancy errors
 - Grammatical redundancy errors
 - Identical formal definitions of classes
 - Identical formal definitions of instances

Circularity errors occur when a class is defined to be a subclass of itself, this can either through a direct statement or through a chain of subclass statements. A partitioning of a concept into subclasses can be disjoint and/or complete, partitioning errors violate such constraints, e.g. a concept that is a subclass of two classes in a disjoint partition. A semantic inconsistency is an incorrectly classified concept in the taxonomy, such as the concept 'wheel' being a subclass of 'car'. In this case the relation should have been 'part-of'. Incomplete concept classification means that some concepts in a partition have been overlooked, and partitioning errors of the incompleteness category involve the lack of an explicit disjoint or complete partition. Grammatical redundancy errors occur when there are several definitions with the same meaning included in the taxonomy, such as 'SUV' being both a direct subclass of 'vehicle' but also a subclass of 'car', which in turn is a subclass of 'vehicle'. The final two errors occur when two classes or instances have the same formal definition but differ in their naming.

Another structural approach is the OntoClean methodology, presented by Guarino and Welty [91] and also described by Gómez-Pérez et al. [85]. The methodology aims at exposing inappropriate or inconsistent modelling choices by using metaproperties to characterise the modelled knowledge. Three properties are discussed; rigidity, identity, and unity, and these can be used to evaluate whether or not subsumption has been misused in the ontology. Rigidity, for example describes the philosophical notion of a property being essential to an instance, such as the property of being a person, which is true at all times for a specific instance, while the property of being a student might change from true to false in the lifetime of an instance. Identity is connected to the ability to distinguish instances from each other

through identity criteria, and unity is related to the notion of being 'a whole'. Using such properties, to characterise the modelled concepts, problems can be discovered in the ontology that might later lead inconsistencies or to errors in reasoning services provided using the ontology.

Most manual ontology engineering methodologies also provide, more or less detailed, guidelines to assist the ontology engineer. These are perhaps not evaluation methods as such but can be seen as checklists, or best practices, in order to produce a better ontology. Such guidelines may for example include naming conventions for elements in the ontology and guidelines of module division and structuring of the taxonomy, e.g. specifying consistent levels of detail. Such guidelines are part of most well-established methodologies, as noted by Gómez-Pérez [85]. Recently researchers have started to investigate how to use ontology patterns, which can be encoded best practices, for evaluating ontologies. The idea has been proposed as future work in some research literature, but so far no concrete method has been published.

Functional and usability-measures

In continued analogy to software engineering, it is not only important to evaluate the pieces during construction but also the ontology as a whole. Although the structural techniques presented in the last section apply also for post-development evaluation, functional and usability-measures complement them in order to evaluate additional aspect of the ontology. This is both true for an ontology built from scratch as well as ontologies considered to be reused or integrated into a current project, i.e. ontology selection.

An overview of ontology evaluation and selection methods was presented by Sabou et al. [169], and a general discussion about ontology selection is present in section 3.1.1. A method dealing with manually comparing and selecting ontologies is the OntoMetric framework described by Lozano-Tello and Gómez-Pérez [122]. The method uses a multilevel framework of characteristics as a template for information on existing ontologies. Five dimensions are used; content, language, methodology, cost, and tools. Each dimension has a set of factors, which are in turn defined through a set of characteristics. The evaluation results in an overall score of the suitability of the ontology in a specific case. In order to tailor the method, a subset of the dimensions and factors can be used depending on what is deemed important in the specific project at hand, and the factors can also be ordered and given different weights. To ease the evaluation of ontology concepts 'glosses', i.e.

natural language explanations, can be generated, as described by Navigli et al. [140], to let domain experts evaluate concepts without the aid of ontology experts.

A similar approach, using quality factors and an ontology of knowledge quality, is described by Supekar et al. [198]. Here the focus is more on 'objective quality', while OntoMetric focuses on subjective usefulness and selection based on the case at hand, and the stakeholders involved. Yet another similar approach is presented by Davies et al. [44], where the authors also suggest that the meta-models of the ontologies can be used to compare them. Other possible comparison criteria could be metadata of the ontology, such as author and construction date. If the ontology is constructed by an authority in the field this might mean more when choosing among ontologies than would some abstract measure of its correctness. In the position paper of Orbst et al. [154] the authors even suggest establishing an ontology certification authority, but this is still a future vision. Approaches such as the one by Lewen et al. [121], using open rating systems to rate ontologies and propagate trust values can become useful in the future when those systems are available and used by a broad community.

As previously mentioned, Gangemi et al. [77] [76] aim to unify all evaluation frameworks by describing ontology evaluation in a formal way. Their meta-ontologies provide a similar framework, as the ones mentioned above, describing available evaluation methods and evaluation criteria. Such a formal model can be used to select appropriate measure and decide what to evaluate.

A good way of comparing and evaluating ontologies is of course to test how well they perform on certain tasks. Such an approach is suggested by Porzel and Malaka [161]. However, this approach is based on comparing to a 'gold-standard', which can be hard to develop and agree on. A 'gold standard' is an ontology that is considered correct and perfectly suited for its task, a kind of standard for other ontologies to reach up to. With an enterprise focus this is usually not a feasible approach, since there is not one single correct way of modelling the enterprise, and to build a gold standard that could apply to all enterprises is not feasible. As noted by Brewster et al. [24] there are no standard tools for evaluating ontologies in specified task environments. This means that, currently, ontologies cannot be compared objectively based on task performance either.

Especially when evaluating ontologies constructed semi-automatically, it can be interesting to analyse how well the ontology represents the input data. Measures that compare the ontology to the content it is supposed

to represent are discussed by Gangemi et al. [77], mainly as functional measures evaluating how well an ontology represents the given domain. Brewster et al. [24] motivate why classical information retrieval methods and measures, such as precision and recall described by Baeza-Yates and Ribeiro-Neto [10], cannot be used directly to evaluate ontologies or ontology construction methodologies in general, although other authors in fact use modified versions of these for special cases, such as Navigli et al. [140]. Instead, Brewster et al. [24] suggest an architecture for evaluating the fit of an ontology to a certain corpus of texts. This is done by extracting information, expanding the information and then matching it against the ontology. Dellschaft and Staab [50] have also addressed the problem of evaluating ontologies extracted from existing information, and they propose a set of measures that can be used when there is a 'gold standard' to which the resulting ontologies can be compared.

Selected methods

Evaluation during construction would in our case mean evaluating the partial results after the different steps of the method have been performed. The method could also be run iteratively, refining the input and thresholds for each iteration. In the first phase of the OntoCase method, the input text corpus was processed using an existing OL system, selected as representing a typical state of the art OL system. This process has not been evaluated separately, since an existing research prototype has been used that has previously been evaluated by its developers. The first step to be dealt with involves pattern matching, ranking, and selection. This step was initially evaluated during its development, using a set of 'toy' examples and through a small experiment involving three expert ontology engineers manually choosing patterns for a specific case at hand. The pattern reuse step, including pattern specialisations and composition, has not been evaluated separately. The main evaluations of the OntoCase method presented in this thesis were experiments where OntoCase was run in one iteration, without evaluating intermediate results, but then thoroughly evaluating the resulting ontologies. Hence, this section focuses on the selection of those ontology evaluations methods.

Relevant evaluation principles were first studied. In this case cognitive ergonomics, transparency, generic accessibility, and organisational fitness were deemed most interesting. These principles are overlapping and in some respects also conflicting, as we shall see below, but by considering

several principles we can achieve interesting results and illustrate some of the trade-offs involved. When combining the set of parameters from the selected principles, as proposed by Gangemi et al. [75], we get the following list, where + stands for a positive influence and - for a negative one:

- depth	+ class-property ratio	+ interfacing
- breadth	+ axiom-class ratio	+ shared differ. notion
- tangledness	+ modularity	+ partitioning
- anonymous classes	+ patterns	+ user satisfaction
	+ annotations	+ recall
	+ accuracy	

The logical complexity parameter was removed since it occurred both as a positive and negative parameter. From the negative side we can conclude that we are looking for small and simple ontologies that are straight-forward to understand. However, in our case the depth of the input ontologies, produced by other OL methods, were exceptionally low. This leads to the conclusion that even though we agree with the general principles, in our specific case we are actually trying to increase the average depth, since we are trying to connect a lot of unconnected concepts.

The anonymous class parameter is connected to the axiom-class ratio, and the partitioning parameter, all three consider the logical complexity of the ontologies. Since we are working with quite light weight ontologies, and the method proposed does not explicitly match and add any axioms, except those already present in the input or in the pattern, these parameters are not considered relevant for evaluating the specifics of this method. However, some parts related to this was anyhow included as a functional evaluation criteria in the SEMCO experiment as we shall see later. A similar reasoning is applied to the annotation evaluation parameter. Annotations are in fact included by OntoCase, but only by directly copying what is present in a pattern, whether this is appropriate or not, thus measuring the amount of annotations would be misleading.

Modularity and patterns are also closely linked, and in our case we choose to focus only on patterns instead of general modularity. Patterns aim at making ontologies more modular, but this is not a hypothesis we will try to prove in this thesis. However, we are particularly evaluating a method applying design patterns, whereby we would introduce a bias by evaluating the presence of such patterns in the ontologies. Interfacing is concerned with the ontology's connection to user interface specifications, which is clearly outside the scope of OntoCase. Also the shared differen-

tiating notion parameter was eventually not considered, due to practical problems of how to reliably measure such a value. In the end we are left with a set of parameters covering some structural and functional aspects. These were used as a guideline when selecting evaluation approaches and measures.

All the evaluations presented in chapter 8 were conducted using the results of the complete OntoCase implementation, i.e. the ontologies produced. After the first research iteration, the initial OntoCase method was tested in the SEMCO project as described in chapter 6. Another ontology was produced in parallel, using a manual methodology. The two ontologies were evaluated using several evaluation approaches, both intended for ontology expert and domain expert review, to get a broader view of the ontologies and indirectly also to evaluate the construction methods. The manually constructed ontology was not used as a gold standard, since it was also developed to test a specific development methodology and thereby it was not suitable as a model for comparison. Instead, benefits and drawbacks of both ontologies were concluded.

First, a general comparison of the ontologies was needed to get an idea of differences and similarities and to cover basic structural parameters. This comparison was done based on intuitive measures, such as number of concepts, the average number of attributes per concept, the average number of subclasses per concept, and the average number of associations per concept, as for example suggested by Gangemi et al. [77]. Note that the ontologies from this first experiment were represented in F-logic, hence the terminology above, e.g. 'attributes' and 'associations'. Also, the cohesion metrics from Yao et al. [222] were used, since we felt that they complement the other measures well. These metrics were: number of root classes, number of leaf classes and average depth of inheritance tree.

We chose not to apply any formal measure of tangledness, but to evaluate this by inspecting the ontology graphically. On the other hand, we were missing some parameter dealing with structural correctness. To be more complete evaluations were performed, by internal ontology experts from our research group, using the two most well-known approaches for taxonomic evaluation, presented by Gómez-Pérez [84] and Guarino and Welty [91]. Internal ontology experts were used for these evaluations, mainly because of their previous knowledge of the evaluation methods. Since we were evaluating both the ontologies and, indirectly, the methodologies for creating them, the idea was that the errors discovered could give valuable indications of advantages and disadvantages of each construction method.

Finally, to evaluate the functional parameters, i.e. the content of the ontologies and their fit to the intended scope, a subset of the OntoMetric framework suggested by Lozano-Tello and Gómez-Pérez [122] was used. For our purpose, only the dimension *content* was deemed interesting, and only one level of characteristics for each factor. Some characteristics were not applicable to both ontologies and since the evaluation at this stage of OntoCase's development was mainly a comparison, these were left out of the framework. Characteristics such as 'number of concepts' and 'number of relations' covers the recall parameter, while 'essential concepts' and 'essential properties' covers the accuracy aspect, and the overall evaluation can be seen as a user satisfaction evaluation. The computation of the final score was not performed, since the number of factors and characteristics were low enough to give a general impression. Domain experts from the company in question formed the evaluation team, but internal ontology experts prepared the material, assisted through the evaluation and collected the results.

In our opinion the most desirable functional evaluation would have been to apply the ontologies in their intended application context. This was however not possible, since the resulting application, i.e. the ArtifactManager, was not introduced into the work processes of the industry partner. It was a planned for a future extension of the project, together with the development of additional applications, as explained in section 6.2.4. Automatic gloss generation, as mentioned previously, could also be a future improvement of the OntoMetric method, but at this time no such tool was available. Finally, the reason for not using a data-drive approach, although specifically suggested for OL methods, was to avoid introducing a bias in favour of the semi-automatically constructed ontology. Since the semi-automatically constructed ontology was already based on textual information, comparing the two ontologies to other documents of the same kind could give an advantage for the semi-automatically constructed ontology over the manually constructed one.

For the evaluations after the second iteration of our research, the final evaluations of OntoCase, the intention was to use the same methods as for the initial evaluations in order to achieve comparable results. However, some changes had to be made, both due to experiences from the first set of evaluations and practical reasons, such as that domain experts from the SEMCO-partner were no longer available because the runtime of the project had expired. One such practical reason was the change in representation language for the patterns and hence also for the constructed ontologies.

Both the basic structural measure and the questions asked in the domain expert evaluations had to be adapted, in order to fit the OWL language terminology rather than F-logic. Another such practical restriction was the size of some of the constructed ontologies. The ontologies of the SEMCO experiment were quite small, a few hundred concepts at most, and therefore it was reasonable to ask domain experts to consider the complete ontology and assess the amount of 'essential concepts' or if the number of properties was adequate. In some of the other evaluations the size of the ontology had increased with a factor 10 compared to SEMCO, to a few thousand concepts. Then it was no longer reasonable to assume that a domain expert, or even an ontology engineer, could have the kind of overall knowledge of the ontology to accurately answer such questions. With these motivations in mind we briefly describe the set of evaluations that were performed for each dataset in the final evaluations.

The first evaluation was conducted as a re-run of the SEMCO experiment, imitating the setting as closely as possible. A similar set of structural measures were collected, as for the initial SEMCO ontology. This time the set contained the number of concepts, the number of root concepts, the number of leaf concepts, the average depth of inheritance, the average number of related concepts, and the average number of subclasses. As before, tangledness was only assessed and discussed informally. Next, both the taxonomic evaluation and OntoClean were again applied. An indication of the recall of the method, with respect to the input ontology extracted by an existing OL tool, was provided by a coverage calculated based on the included terms and relations. Finally, the functional criteria were analysed theoretically through discussing the domain expert evaluation performed in the initial experiment, and the changes in the new version of the ontology. The new version of the ontology was also compared to the final version of the SEMCO ontology, which was a hand-crafted combination of the two ontologies of from the initial evaluation.

The second evaluation, involving the JIBSNet ontology, was done in a quite similar manner. First the basic structural characteristics were collected, regarding the number of concepts, the number of non-taxonomic properties, the number of root concepts, the number of leaf concepts, the average depth of inheritance, the average number of related concepts, and the average number of subclasses. The coverage over the input ontology was again computed and discussed, and the structural evaluations considering taxonomic relations were again applied. However, these evaluations and the evaluations by domain experts from JIBS were performed on a randomly

selected set of concepts and properties, since the ontology was too large to be evaluated completely. The domain expert evaluations were performed by letting the experts assess the randomly selected subsets of concepts and properties, and a criteria of agreement was applied, as suggested by Gangemi et al. [75]. The number of correct and incorrect concepts and properties were analysed to draw similar conclusions as from the domain expert evaluation in the SEMCO case, related to the accuracy and user satisfaction parameters.

In the last evaluation the situation was slightly different, concerning the agricultural ontologies of the FAO. In this case again no domain experts were available at the moment of conducting the experiment, whereby the accuracy and user satisfaction-related evaluations were performed by an ontology engineer. This reduces the reliability of the results, but was considered a better option than to omit this part altogether. However, this also had the effect that the taxonomic evaluations could not be performed with enough accuracy, and thereby they were omitted. It was simply too hard for a non domain-expert to assess, for example the rigidity of domain specific agricultural term. Assessing its relevance with respect to agriculture could be performed using thesauri and other resources, but evaluating the inherent properties was deemed to uncertain. In this case the evaluations consisted of the collection of general structural measures as before, and assessing the accuracy of a randomly selected set of concepts and properties. In addition to this, a small experiment was also performed to investigate the interplay between OntoCase and other OL methods. These ontologies were evaluated together with the rest of the FAO ontologies, but also the patterns that were included were assessed. Similarly to the assessments made about concepts and properties, the patterns were tagged as correct or incorrect to include in the ontology.

4.3 Description of the research process

Based on the above mentioned research methodology and evaluation strategies, the research was carried out during five years at Jönköping University, in cooperation with Linköping University. In practise this research was divided into two iterations, both consisting of a set of steps inspired by the process presented in section 4.1.2. Step 1 being construction of a conceptual framework, step 2 architecture development, step 3 analysis and design, step 4 system building, and step 5 evaluation.

In addition to the general research methodology the research was guided by a study process for PhD studies applied in the research group. The process contained four phases, denoted orientation, conceptualisation, elaboration and finalization. The PhD student was intended to find and elaborate the research questions in the orientation phase by studying related literature and current approaches in the area. The phase was concluded by writing a state of the art report, also containing a discussion of open issues and the proposed research questions. In the second phase, the core of the solution was developed, i.e. the general framework of the solution and novel theoretical concepts. In the third phase, the general framework was detailed and all subproblems were resolved, if necessary new techniques had to be developed. In the finalization phase the work was finalised, evaluated, and the thesis written. It was not possible to follow this model completely linearly in this research, whereby some of the phases were revisited. In the first iteration, the focus was on orientation and conceptualisation, and also partially covered the third phase, elaboration. The second iteration revisited the orientation and conceptualisation phases, but focused on elaboration and finalization. The process and its approximate chronology can be seen in Figure 4.1.

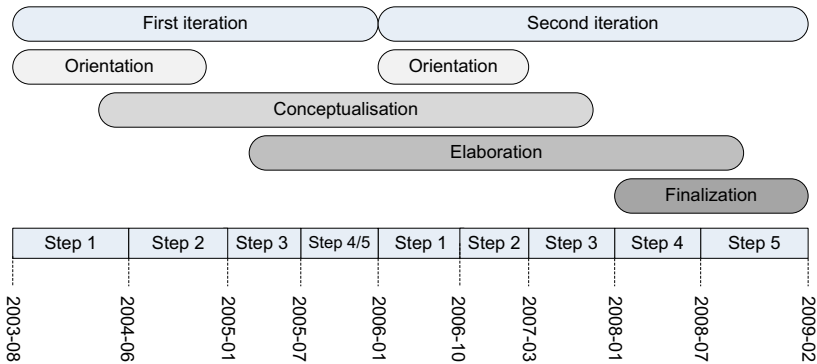


Figure 4.1: The research process with an approximate time line.

4.3.1 First iteration

In the first iteration of the research the focus was mainly on finding open issues, defining the problem and providing an initial solution proposal. Test-

ing the initial proposal would then guide the further development of the proposed solution and point out possible improvements.

Literature study

The first iteration started with a thorough literature study and identification of open issues and relevant real-world problems within the field. The initial idea proposed as a subject for this thesis was to try to automate the process of ontology construction using something similar to software design patterns, but tailored to ontologies. During the initial literature study, mainly two directions were studied; patterns and ontology engineering. The initial research idea led to the study of literature related to the following notions:

- **Ontology engineering**
 - Classical knowledge representation and reasoning.
 - Origin and definition of the concept of ontology.
 - Characteristics and usages of different kinds of ontologies.
 - Knowledge acquisition and manual ontology engineering methods.
 - Ontology learning methods and their origin.
 - Ontology patterns.
- **Reuse**
 - Theoretical foundations of reuse.
 - Reuse methodologies.
 - Software reuse.
 - Knowledge and ontology reuse.
 - Specific practical methods for ontology modularisation, search, matching, and integration.
- **Patterns**
 - Theoretical notions of patterns and the origin of software patterns.
 - Software patterns.
 - Data model patterns.

- Knowledge engineering patterns.
- Experimental results of pattern usage.
- Pattern matching and selection.

Important lessons learnt at the beginning of the research were that ontologies have been around for quite a while, although not in the same form as today, but mainly it is the terminology that has changed. For example during the 80's, knowledge bases were constructed to support expert systems within AI. Some of those knowledge bases would with today's vocabulary be called ontologies. The changing vocabulary and different fields picking up and exchanging ideas makes the study of previous literature difficult, whereby the first task was to get an idea of the history of computer science ontologies and the terminology used in this field.

Related fields using patterns and existing methods for ontology engineering patterns were considered in the study of pattern research. Ontology engineering was first considered in general but then the focus was put on semi-automatic methods and reuse methodologies within ontology engineering. The literature study resulted in a 'state of the art' report published as a technical report at Jönköping University and a set of open issues that should be further investigated. This also marked the end of the first part of the orientation phase. The open issues listed at that point were:

- Semi-automatic ontology construction
 - Extraction of more complex ontological elements, such as general axioms.
 - Named relation extraction.
 - Composition of the ontology from the extracted parts.
 - Adding background knowledge and refining the ontology.
- Ontology reuse
 - Improved ontology matching, matching 'pieces' of ontologies.
 - Assistance for searching ontologies.
 - Assistance for ontology selection.
 - More generic modularization approaches.
 - Ontology documentation and metadata.
- Patterns

- Higher level patterns for ontologies.
- Draw on (transfer) experiences from software patterns and data model patterns to ontologies.
- Pattern usage, how to find, select, and reuse patterns.

A set of research questions were chosen from these open issues, as the focus of the continued research effort. The initial questions implied studying the nature of ontology patterns and how they could be developed and used, as small building blocks, for improving current OL approaches.

Conceptual framework development

The construction of a conceptual framework (step 1) in the first iteration initially focused on defining the nature of ontology patterns. Using the software engineering field as an inspiration, and going back to the origin of patterns, pattern categories on different levels of abstraction and granularity were defined. From software engineering it was clear that as the field had progressed and systems became more and more complex patterns on more abstract levels were needed. Analogous to this development we predicted that ontology patterns, not only on the syntactic level, would be desirable. Patterns corresponding to design and architecture patterns in software engineering were consequently envisioned. From the literature on ontology applications and categories of ontologies we could derive a need for generic descriptions of the ontology as a whole. This would address the issue of generic reference architectures for ontology applications and the reuse of ontologies based on standard sets of requirements.

Initially five levels of patterns for ontologies were proposed; syntactic patterns, semantic patterns, design patterns, architecture patterns and application patterns. This would later prove to be a less appropriate division, because the levels were divided based on different dimensions, abstraction in some cases and granularity in others, but this was not changed until the second iteration of the research process.

The development of the initial idea of how ontology patterns should be used in semi-automatic ontology construction belongs to the conceptual framework. Patterns here were seen as a way to add additional structure and additional information to the elements that were, at that time, possible to extract from a text corpora by OL systems. If the extracted elements are seen as scattered pieces of information, the patterns should be seen as the glue that put them together into something reasonable and useful to

the ontology developer. The idea was to focus on design and architecture patterns, in order to try to connect single ontological elements into a structure, and to give the ontology an overall architecture. We chose to focus on ontology design patterns in the form of small pieces of ontologies.

Architecture development

The development of a system architecture involved proposing the outline of an initial method for using patterns in semi-automatic ontology construction, and to set the requirements for that method. The existing OL systems were taken as the basis on which to apply the pattern approach. The systems were studied and then the assumption was made that what could be extracted with reasonable reliability was a set of terms and a set of unnamed binary relations. This was considered as the input requirement for the development of the pattern-based approach. The process was then divided into two steps; the matching of patterns to the extracted terms and relations, and the composition of the ontology based on the matching results. Additionally the requirement was that apart from the input of texts and patterns, and the setting of system variables, the process should be as automated as possible.

Analysis and design

In a brief analysis and design phase the initial idea was detailed into steps, and existing implementations possibly suitable for design pattern matching and pattern composition were tested. The pattern matching phase included the obvious steps of matching the extracted terms to the labels of the concepts and the extracted relations to the relations of the patterns. The naive method for term matching is to use simple string matching. More thought was required for relation matching, but eventually it was assumed that if a relation existed between two terms and there were reasonable matches among the pattern concepts, then that relation might also be a match to any relation between the pattern concepts. Matching scores were computed for the patterns based on fractions of relations and concepts matched.

The ontology composition was based on the principle of including only matched parts of the patterns and assuming pattern overlap wherever the same concept label was used. Basically, this resulted in an approach for combining a set of patterns, rather than combining the extracted elements. Elements not matched were simply discarded, and matched terms were used as synonyms of the pattern concepts. The output of the complete method

was an ontology, that was intended to act as a starting point from where an ontology engineer could continue to build and extend it into a complete enterprise ontology. The reasoning behind the analysis and design of the method steps was to keep it simple, and first try existing approaches and tools before developing new ones.

System building

The system was built by integrating existing components and tools in a more or less ad-hoc fashion, in some cases still requiring manual input formatting. The most elaborate, and freely available, OL tool at that time was selected to be used for the text processing. String matching tools were used for the term to concept matching, and the ontology modelling environment connected to the OL tool was used for ontology composition. In this step, the idea was to keep everything as simple as possible for the initial experiments rather than trying to build an actual integrated system.

Evaluation

The method was tested and evaluated through a controlled experiment within a research project (SEMCO) with an industrial partner. The evaluation was performed as a comparison with a manually engineered ontology for the same application case, with the intention of discovering the shortcomings of the naive method and identifying required improvements. The evaluation was carefully set up, based on a literature study of available ontology evaluation methods and tools and the resources that were available within the project. The focus was mostly on comparing the two ontologies and discovering their characteristics, and thereby characteristics of the methods used to construct them, rather than evaluating the objective quality of the ontologies and methods. Thereby the evaluation methods were selected based on getting as much useful information as possible to discover the shortcomings of the initial approach.

The evaluations were conducted in cooperation with the ontology engineer producing the manually constructed ontology, and evaluation judgments had to be done in agreement, in order not to be biased towards any one of the ontologies. The evaluations done in cooperation with employees at the participating company were done in several sessions, lead by the two ontology developers. The people participating were product developers at the company in question, which turned out to be a suitable target group since they were educated and familiar with modelling, and had a

good overview of what the company actually did and the kinds of products the company produced. The results from the evaluations provided a list of drawbacks and benefits of each ontology, and construction method. The list of drawbacks was used as a basis for suggesting improvements and evolving the set of research questions, while the set of benefits acted as a motivation for the initial idea and showed that the idea of pattern-based semi-automatic ontology construction was truly feasible.

4.3.2 Second iteration

Based on the evaluation results the set of research questions was specialised and slightly modified at the start of the second iteration. Using these new research questions, and the set of suggested improvements, the process again started with a literature study.

Literature study

In the second iteration a new, but considerably smaller literature study was performed, for example concerning the methodology of case-based reasoning (CBR). One of the issues discovered in the first iteration was the problem of having enough patterns as input to the process. In the first iteration the patterns were manually engineered, but one idea that arose was to be able to extract patterns, or at least pattern candidates, automatically from existing ontologies and to provide feedback to the patterns used. This idea together with the need to put the proposed method into a larger framework initiated the study of CBR. CBR seemed to be based on a high level idea of reusing knowledge similar to our initial pattern-approach.

In addition to the study of the CBR methodology, the previous literature study was updated with more recently published articles on for example OL. This was partly an ongoing process throughout the research. Also more specific details were studied, such as methods for bridging the abstraction gap between patterns and extracted terms, and existing methods for ontology search and ranking, with the intention to use such methods for improving the pattern selection step.

Conceptual framework development

The conceptual framework resulting from the first iteration was slightly modified, to incorporate more aspects, for example both the abstraction and granularity of the ontology patterns, and additional phases were added to

the proposed ontology construction process. The development of the pattern classification framework was based on theoretical analyses of the pattern categories and numerous discussions with both pattern experts and ontology researchers. The idea was to develop a comprehensive framework that would be easy to understand and help ontology engineers communicate around patterns, and to cover the complete development process of the ontology and any envisioned future problems. This resulted in merging of two of the initially proposed abstraction levels, and the addition of several granularity levels on each level of abstraction. The result was a more comprehensive framework where the levels were divided based on a single matter of concern, i.e. either abstraction or granularity.

The semi-automatic ontology construction method was extended to become a complete framework for pattern-based ontology construction and also to cover the life cycle of the patterns. An evaluation phase was deemed necessary at an early stage, whereby this was added to the previous two steps of the method, and additionally the extraction of patterns was incorporated into the method. When reading about CBR in the literature it was discovered that this almost perfectly corresponded to the four phases of the CBR cycle, on a high level. The techniques used might not have been the same, but to view patterns as encoded experience and the use of patterns as reusing past solutions was an easy connection to make. By making that connection, the method suddenly had four phases, with predefined steps and tasks that already approximately corresponded to what was proposed in the first iteration, see Figure 4.2. The method was at this stage named *OntoCase*, referring to ontologies and the connection to the CBR methodology. We decided however to only include the elaboration of the first two phases in the scope of this thesis.

Architecture development

The architecture of the first two phases of *OntoCase* was then extended and modified to improve the shortcomings found in the first iteration, and new requirements were developed. The pattern catalogue was now seen as a 'pattern base' that might contain several different catalogues, and that needed a structure in itself. The steps of the two phases were subsequently detailed more as a collection of possible methods that could be selected by the user of such a method, rather than as a linear 'pipeline' process. The basic steps remained the same, including pattern matching and selection, and pattern composition when constructing the initial ontology.

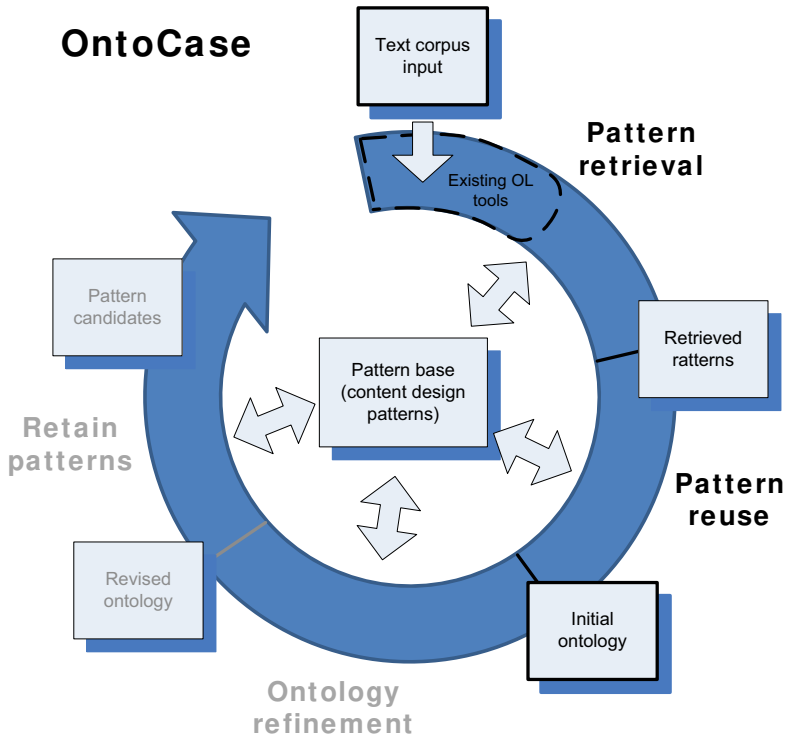


Figure 4.2: The OntoCase phases in focus.

Analysis and design

In the analysis and design step both improvement of previously designed components as well as designing new parts were included. For example, two more algorithms were added to the concept matching in the pattern ranking, resulting in the ranking being based on both string matching, a head heuristic, and the linguistic information of WordNet. These additions were made based on the study of relevant existing research on ontology search and ranking, but were also considerably modified in order to fit the specific case of ontology design patterns.

System building

The considered parts of the method were then implemented as running research prototype software. This time the approach was implemented as a

whole, and not only as an integration of existing software. Existing software was still used for the input processing, since it was beyond the scope of this thesis to develop new OL methods for element extraction. The implementation was done based on the Jena ontology API, in order to conform to the general policy on interoperability of software within the research group. In addition, two different string matching libraries were tested, and some more tests were run with respect to the choice of OL tool to integrate. Thereby the second implementation was more thoroughly prepared, and done in less of an ad-hoc fashion. Nevertheless, it should be considered as an initial research prototype, e.g. lacking proper documentation and installation support. Neither maintainability nor efficiency were priorities when developing the system. Each part of the prototype was tested and evaluated on small example cases during the implementation, in order to discover problems and fine-tune the method.

Evaluation

The method was finally tested through a set of experiments, both a re-run of the previous experiments from the first iteration and new experiments, and method characteristics were assessed. The SEMCO evaluation case was re-run in order to be able to identify differences and improvements compared to the results of the first iteration. It was not possible to completely re-run the evaluation however, since the experts at the project partner site were no longer available. In addition to this experiment, an ontology was constructed for the Business School at Jönköping University, based on internal documents. This was an easy to access case, where domain experts were readily available to assist the evaluation. Nevertheless, it was not deemed too 'internal' since our research was conducted at the School of Engineering, although at the same university.

The method was also tested on a set of ontologies that were not really enterprise ontologies, but rather more general domain ontologies. The reason for doing this was the lack of a proper case involving an enterprise ontology. For these cases some of the patterns were not suitable, but otherwise the OntoCase approach is not, as a general framework, tailored to only enterprise ontology construction. These experiments were considered as very valuable in order to provide more evidence of the characteristics of the proposed method, and to also show the range of applications that the method might be used for.

Finalization

During both the first and second iteration, peer-reviewed research papers were published at several conferences to communicate and validate the relevance of the research with other researchers in the community, see section 1.5. The results were presented at both internal and external research seminars, and at international conferences. Finally, this thesis was written based on the results presented in the papers and the latest evaluation results, in order to communicate the final results of the research to the ontology engineering research community.

Chapter 5

Ontology patterns

As noted in chapter 3 patterns are used in many different areas of computer science, both as construction templates and as indications of repeating events or structures within a set of objects. The term ontology pattern can be intuitively interpreted in several ways, either it brings to mind recurring structures within ontologies, templates for constructing ontologies, or even patterns of how several ontologies are connected or used.

This thesis focuses on patterns for engineering ontologies, i.e. patterns that may assist an ontology engineer in the process of constructing an ontology. Such patterns may be recurring structures in actual ontologies or engineered templates, but the intention is to assist the ontology engineering process. Focus is also on patterns related to the construction of the actual ontology, and not, for example on patterns for ontology engineering work processes or documentation of ontologies. A range of ontology engineering patterns are discussed and characterised in this chapter, with the intention of structuring the area of ontology engineering patterns, hereinafter denoted ontology patterns, and providing a comprehensive typology of patterns to be used as a vocabulary for ontology engineers and researchers. After presenting the complete typology, we focus more in depth on ontology design patterns, and specifically content patterns solving specific design problems in ontologies.

5.1 Characteristics

Reuse has been applied in knowledge engineering for several decades, but it is not until recently the notion of patterns has been adopted on a broader scale. We generally define the notion of *ontology pattern* as:

Definition 3. *An ontology pattern is a set of ontological elements, structures or construction principles that intend to solve a specific engineering problem and that recurs, either exactly replicated or in an adapted form, within some set of ontologies, or is envisioned to recur within some future set of ontologies.*

Based on this definition a set of elements, structures or construction principles have to exhibit the following in order to be a pattern:

- Provide a solution to a specific problem or category of problems (problem focus).
- Recur in existing solutions or envisioned solutions (reuse focus).

These are considered as necessary and sufficient conditions, whereas every recurring solution to a problem may be denoted a pattern. One can also take into account the notion of a 'good' solution, a solution based on best practices, as is common in the field of software engineering patterns, but this is not a necessary condition for something to be a pattern. A pattern that does not follow such best practices, but instead exhibits a common problem or mistake is usually denoted an *anti-pattern*. An anti-pattern exhibits a common pattern, which may both provide a possible solution to a specific problem and may recur, but is a pattern that exhibits a 'bad example' of how not to solve a problem. The notion of anti-patterns will not be treated further in this thesis.

The definition above encompasses all kinds of patterns, as described further below, and captures the notion of a pattern being something recurring, whether it has already been observed several times or is constructed so that it will be reusable in future situations, and will thereby be recurring. The definition also captures the abstract nature of a pattern, which implies that a pattern may recur in many different forms; it may be adapted and changed, but it is still an incarnation of the same pattern. This abstract notion of 'adaptation' makes the pattern definition not very usable as a formal definition in practise, but it can be seen as a general description of the intuition behind patterns. Adapting a pattern to solve the modelling

problem at hand is denoted *pattern instantiation*, or in the case of ontology design patterns also *pattern reuse*, since in this case we are actually reusing pattern concepts and properties in the ontology. More specifically content pattern reuse is commonly achieved through *pattern specialisation*. Pattern instantiation might be a signature morphism, as we shall see later for content design patterns, but might also include other types of adaptations.

The notion of ontology element was mentioned above. In this thesis we will use the terms ontological element and ontological primitive interchangeably to denote the 'building blocks' of ontologies, i.e. concepts, relations, and axioms, as described in section 2.2.4. In this thesis we do not restrict ourselves to one specific ontology representation language, although the OWL entity types can be regarded as ontological elements or primitives in our sense, but we broaden this definition to include all language axioms in the notion of ontological primitives. OWL entity types are, for example data types, properties, individuals and classes. When combined with all OWL axioms, this constitutes a list of the basic language building blocks, which is what we denote with the terms ontological element and ontological primitive.

This notion is also connected to the distinction between a model and its representation. A model can exist in its own right, but in order to be communicated, stored, or used it has to be represented using some representation language. The language may be formal or informal, graphical or textual, but all models existing outside the mind of a human being are represented through some kind of representation language or notation. In the rest of this thesis we will distinguish between ontology modelling languages, which are abstract *logical* languages, in turn possible to represent with a graphical or textual syntax, for modelling ontologies. Such modelling languages are usually based on a foundation in logics, such as description logics (DL) or F-logic, but the important distinction is that such languages can then be syntactically represented in many different ways using a computer processable representation language. For OWL such a formal computer processable syntax is the OWL XML serialisation, and for RDF it can be a triple notation. This distinction between a modelling language and its computer processable representations is not further formalised here, it depends on the perspective of the observer. The above description aim to provide the intuition behind the notions, helping the reader to understand the pattern definitions and descriptions provided later.

The general definition of ontology pattern given at the beginning of this chapter accommodates many different kinds of patterns. Other patterns

concerning ontologies may be imaginable, but in this thesis we limit the scope to patterns for ontology engineering, even more specifically the actual ontology construction. Ontology patterns are described and characterised below, with respect to three pairs of aspects:

- Extraction and purpose
- Structure and content
- Abstraction and granularity

5.1.1 Extraction and purpose

Patterns may be seen from at least two perspectives regarding their origin and usage. Either patterns are purely experience-based, the *pattern mining and recognition* perspective, or they are carefully designed and constructed, the *patterns as templates* perspective. The two perspectives were mentioned briefly in section 3.2. Sometimes the distinction between the perspectives is not clear and sometimes overlapping, but the perspective implies some important things concerning the nature of the patterns. The perspectives have not been treated in detail in previous literature on ontology patterns.

From the experience-based perspective, patterns are recurring structures that can be found in some set of solutions, for example recurring sets of concepts and relations in ontologies. In this sense, there are no requirements on what a pattern should contain, its level of abstraction, or how it is structured or retrieved; it is simply a solution that solves some problem and that occurs sufficiently often based on studying previous solutions. Whether or not it is appropriate for ontology engineering, i.e. if it is a best practice or an anti-pattern or something in-between, cannot be determined simply by discovering it.

Patterns can thereby be mined and recognised in ontologies as well as in all other types of structures, according to the experience-based perspective. The process of mining patterns in ontologies would require a set of ontologies to be available, criteria for what kind of patterns should be recognised, and a threshold for a sufficient number of occurrences. Input to the process is the set of ontologies, modifiers are the pattern criteria and the threshold, and the output is the sets of recurring ontological primitives that constitute the patterns. Depending on why this process is performed different criteria may be set, for example we might only be looking for a 'connected' set of ontological primitives, where the ontology is viewed as a graph and the

primitives constituting a pattern candidate need to form a connected graph, in order to find a small area of the ontologies that recur in the ontologies of the set. Additional criteria might be that the areas exhibit a 'strong connection' as defined in graph theory, see Cormen et al. [40] for details. A harder problem would be to discover not only a set of elements, but more general, abstract, construction principles or structures. Discovered, and possibly generalised, patterns stored previously can be used to recognise new occurrences of the patterns, hence the connection to pattern recognition.

In contrast, from the design perspective, i.e. the patterns as templates perspective, patterns are carefully engineered templates that represent a consensus how to best solve a specific problem. The templates have to be sufficiently general and abstract in order to be reusable in many cases and they usually represent some notion of best-practices, i.e. a 'good' way to solve a problem and not just any way to solve it. Or being an anti-pattern the pattern can instead show the opposite, an example of how not to solve a problem. From this perspective a pattern can in theory be constructed entirely without evidence from actual solutions, only based on the opinions of some community. Patterns are nevertheless usually based on previous experience of the community involved, although perhaps not based on actual solutions, and the belief that the pattern will be (re-)usable for constructing future solutions.

From a hybrid perspective, patterns can be discovered, instead of completely manually engineered, generalised and modified manually, and then used for engineering. Especially when automating an engineering task, the hybrid perspective is commonly applied. Then patterns can be produced semi-automatically, and subsequently matched and applied automatically. The manual step is most often the generalisation of discovered patterns, and sometimes additionally the pattern matching, although in this thesis we will later propose methods for automating pattern matching and specialisation.

Patterns are usually represented and presented differently depending on how they are extracted and used. Patterns automatically extracted and intended for automatic pattern matching can be represented in the same representation language as the ontologies themselves, thereby being ontologies, logical vocabularies or XML-statements for instance. Whereas other more abstract patterns might be represented using a description language, modelling language, more suited to the level of abstraction of the pattern. The usage of the patterns plays a crucial role for their representation. Patterns intended for manual use can in some cases be sufficiently described using an informal template and natural language texts, in order to inspire de-

velopers when producing new solutions. Patterns to be used automatically however, have to be formally expressed in a machine processable notation and template-based natural language descriptions may not be required at all.

In ontology engineering both perspectives, including the hybrid approach, have been applied and described in literature. The pattern mining and recognition perspective has been applied, for example for ontology search and matching, when two or more ontologies are compared in order to find overlapping areas. The hybrid perspective has been used in OL, when discovering linguistic patterns for extracting ontological elements from text and then proceeding to extract elements using the learnt patterns. The patterns as templates view is more common in manual ontology engineering, where ontological design patterns have been used for guiding ontology design processes, teaching novice ontology engineers good design principles, and for facilitating communication between developers.

5.1.2 Structure and content

When discussing ontology patterns, whether extracted or manually constructed as mentioned above, there are two main kinds of patterns presented in literature:

- Structural patterns
- Content patterns

For more background on these types of patterns and existing references to them in literature, see section 3.2.3.

Structural patterns deal with the logical structure of the ontological elements, but not with the actual ontology represented by these. A structural pattern is a logical vocabulary with an empty signature, i.e. no actual concepts, relations or other axioms are actually present. In contrast content patterns deal exactly with such ontological modelling solutions, and can even be domain-specific, depending on their level of ontological abstraction and intended use. Content patterns are thereby a specialisation of structural patterns, since content patterns both constrain the logical structure of how the solution to the problem should be modelled, and in addition also set requirements on the ontological content. Content patterns are instantiations, and possibly combinations, of structural patterns where the signature is no longer empty.

Structural patterns can be instantiated by adding a signature, e.g. adding actual concepts, to the pattern. A structural pattern can be instantiated with any signature, although not all signatures would produce a 'correct' ontology with respect to the domain at hand. To assist in solving such problems, i.e. to produce a reasonable ontology with respect to a domain, there are some additional requirements on the instantiation of content patterns. The instantiation of content patterns only allow morphisms that preserve downward taxonomical ordering. The intuition behind this is that since content patterns present a solution valid in some domain, although perhaps very broad or general, this solution will also be valid in sub-domains, but will not necessarily be generalizable to a broader domain or transferable to another domain.

5.1.3 Abstraction and granularity

Additional important aspects when discussing ontology patterns are the level of abstraction and granularity of the patterns, which has not so far been treated in related literature. Granularity is concerned with the scope of the pattern, e.g. treating some small part of an ontology or a complete ontology. Whatever level of granularity is chosen, patterns need to be focused in order to solve a specific problem, on a specific level of granularity. Granularity can intuitively be seen as the level of detail, or the scope, of what is treated. Either we may be interested in the details of some individual elements of an ontology, a small part of an ontology or the structure and content of the complete ontology. If we are presenting a pattern concerned with the complete ontology we will probably not specify as many details as when presenting a pattern concerned with only a single element of an ontology.

The level of granularity is closely related to the notion of abstraction level of an ontology pattern. Abstraction is about hiding the details of some structure in order to describe another aspect of the structure in a more convenient manner, thereby 'abstracting' from the details. An abstraction is more than a change of granularity, because a change in abstraction implies not only changing the scope and detail of the 'view' as described above, but also changing the perspective to view completely different aspects, possibly also in a new form of representation. When moving up or down in terms of abstraction the basic building blocks change, and the description language may thereby be different to utilise different possibilities to describe features and characteristics in order to emphasise other relevant aspects.

5.2 Typology of ontology patterns

Our research defines four levels of abstraction, including applicable levels of granularity, which are described below. For an initial overview of the granularity levels see previous discussion by Blomqvist [22].

5.2.1 Ontology application patterns

In section 2.3 four sets of questions were stated in order to summarise considerations during the ontology engineering process. The first set of questions was:

- What is the purpose of the ontology? How is it to be applied in a software system?

The level of application patterns is intended to address exactly these types of problems, concerned with the overall scope and purpose of the ontology. Such patterns specify, for example what tasks the ontology, or ontologies, should be able to perform and the nature of the interface(s).

Albertsen and Blomqvist [7] have defined an ontology *application pattern* as follows:

Definition 4. *An ontology application pattern is a software architecture pattern describing a software system that utilises ontologies to create some of its functionality. The pattern also describes properties of the ontology or ontologies in the system, and the connection between the ontology and the rest of the system.*

In this case the abstraction level is the one of ontologies as components in software systems, and the complete ontology is the smallest unit considered, with respect to ontology engineering. Consequently there is only one level of granularity possible, viewing the complete ontology as a unit. The focus of this type of pattern is on how to use the ontology and the functionality it is intended to provide. These patterns provide the connection to software patterns, mainly software architecture patterns and reference architectures, and thereby to the system that uses the ontology.

There are no restrictions on ontology application patterns when it comes to how they are extracted or designed, or to structural or content issues. Although quite uncommon in software architectures, such patterns could, in theory, be automatically derived from actual systems, thereby conforming to the pattern mining perspective. More commonly, software architecture

patterns are quite abstract and carefully designed architecture styles that provide a well-proven structure for a certain type of system, which conforms with the patterns as templates perspective. Whether such patterns focus on structure or content is a matter of how the ontologies are described within the pattern. For the pattern to be structural there cannot be any restriction on the content of the ontology, only the logical structure of its interface. In contrast, if the ontology is required to expose an interface able to provide specific information, possibly even connected to a domain, then the pattern would have the character of a content pattern.

Ontology application patterns generally aim to describe generic ways of using implemented ontologies, e.g. in terms of purpose, context, and interfaces. The pattern prescribes the general requirements of the ontology within the system, which can then be used by ontology engineers when constructing it. This idea of abstracting the best practices of applying and using an ontology, or several ontologies, within some context or application, is an important issue. Many models and common practises exist, but are not formalised as patterns. A generalisation of existing models could result in initial application patterns for ontologies. Application level patterns are dependent on the ontology usage area, and can also be domain dependent. For a more detailed discussion about such patterns, and notes on how to represent them, see the discussion by Albertsen and Blomqvist [7]. Ontology application patterns are not treated further in this thesis.

5.2.2 Ontology architecture patterns

Again we return to section 2.3 where four sets of questions were stated in order to summarise considerations during the ontology engineering process. The second set of questions was:

- What parts are to form the ontology? How should the architecture of the ontology be formed?

The level of architecture patterns is intended to address exactly this type of problem, concerned with the overall organisation and possibly even content of the ontology.

An ontology *architecture pattern* is defined as follows:

Definition 5. *An ontology architecture pattern is a pattern describing the overall structure of the ontology, possibly restricting the design patterns that may be used to implement the ontology.*

This means that an architecture pattern can be an abstract composition of several structural and content patterns on the design level, but the main intention is the description of an overall structure of the complete ontology by restricting how it is composed, or defining what it should contain and how. Note that on this level of abstraction, the details of the ontology are hidden, the relations and concepts that are to be used are not considered, nor how to model those in some language. At this level, the ontology can, for example be considered as composed of ontology modules, or display other architectural styles, but remains on an abstraction level above the logical structure and detailed content of the ontology. At this level, the smallest units can be modules or layers, for instance.

The intuition behind ontology architecture patterns is to describe a generic way to design the overall structure of an ontology in order to fulfil the goal of the ontology in question. Important questions when designing ontologies are whether to divide the ontology into modules, views or layers or use other construction principles. Note that such restrictions should most likely be described in a designated ontology architecture description language, which distinguishes patterns on the architecture level from ontology design patterns treating the complete ontology, see the following sections. Ontology architecture patterns can be of two different levels of granularity, either treating the complete ontology or only some smaller part, i.e. perhaps describing the architecture of one or several modules of the ontology.

An architecture pattern can be a structural architecture pattern, only restricting the structure, while an architecture pattern specifying certain modules with specific content is a content pattern, also addressing the content of the ontology. Concrete architectures could, in theory, be discovered automatically from existing ontologies. These can then be generalised into architecture patterns, or patterns may be engineered manually as templates and guidelines. To discover architecture patterns automatically, methods for analysing the overall structure of an ontology would be needed. Such methods are not very common today but do exist, for example for discovering modules within a certain ontology. Automatically discovering the architecture of an ontology is an interesting future research area, although challenging. Manually designed ontology architectures exist in many semantic applications, but to the best of our knowledge these have not been generalised to ontology architecture patterns.

We define the notion of ontology reference architecture as follows:

Definition 6. *An ontology reference architecture is a domain-specific ontology architecture pattern containing restrictions and requirements on both the logical structure and the content of a complete ontology.*

So far no such ontology reference architectures have been defined, but this is considered an interesting future area of research, as well as studying different possible architecture description languages for ontologies. Ontology architecture patterns, and more specifically ontology reference architectures, will be mentioned briefly in the context of the OntoCase approach in this thesis, but only to point out the possible benefits of using such descriptions, such patterns will not be treated in detail and are considered future research topics.

5.2.3 Ontology design patterns

From section 2.3 we retrieve one of the four sets of questions that were stated in order to summarise the ontology engineering process. The third set of questions was:

- What should the ontology contain? What concepts, relations, and axioms?

The level of design patterns is intended to address the type of problems related to the actual content of the ontology and how it should be modelled. Whether the complete ontology is considered all at once or if it is divided into modules modelled independently is not the concern of this type of patterns. However, how these parts are realised using ontological elements such as concepts, relations and axioms is the focus of design patterns.

Ontology design patterns aim to describe a generic recurring construct in ontologies that solves some specific modelling problem. We define an ontology *design pattern* as follows:

Definition 7. *An ontology design pattern is a set of ontological elements, structures or construction principles that solve a clearly defined particular modelling problem.*

Ontology design patterns describe solutions on the abstraction level of logical ontology modelling languages, whether graphical or formalised in mathematical logics.

Such patterns exist on three levels of granularity, i.e. treating individual logical elements, such as the definition of one concept, relation or axiom, treating a smaller part of the ontology, such as one module or part of one module combining several primitive elements, or treating the complete ontology, such as restricting the overall structure of the ontology on the logical level. The latter is not to be confused with the architecture patterns previously described. On the abstraction level of architecture patterns the overall structure, and possibly content, is in focus without discussing the logical realisation of that structure on the modelling level. An ontology architecture pattern can also be described using an abstract ontology architecture description language, while on the design level the focus is on the logical constructs of the ontology, although the patterns on the highest level of granularity also do concern the complete ontology.

What is sometimes denoted a semantic pattern in literature, see chapter 3.2.3, is from our perspective a commonly used ontological element but independent of the representation language, and thereby an ontology design pattern on the lowest level of granularity. The pieces that together build a larger ontology design pattern, on a higher level of granularity, are usually smaller ontology design patterns on a lower level of granularity. We can think of an ontology design pattern on the lowest level of granularity as a primitive, a building block, for constructing ontologies that cannot intuitively be further divided into sub-parts. These kinds of patterns have been commonly used in graphical interfaces of ontology engineering tools where the users do not have to care about the realisation of the pattern in a representation language but only choose to graphically construct concepts, relations and axioms.

Design patterns can, in turn, be divided into structural design patterns and content design patterns, depending on the restrictions they impose. A *structural design pattern* is defined as follows:

Definition 8. *A structural ontology design pattern is a logical vocabulary, with an empty signature, that solves a specific modelling problem.*

A *content design pattern* is defined as follows:

Definition 9. *An ontology content design pattern is an instantiation of one or several logical design patterns proposing a set of ontological elements, with a non-empty signature, combined to solve a specific modelling problem.*

Structural patterns can be discovered automatically but without considering the content aspects it may be very hard to restrict and make use of

such patterns, many nonsense patterns are discovered, as noted by Thörn et al. [201]. In contrast, content patterns might be very useful to discover automatically, a kind of 'component mining' for ontology reuse. Manually engineered structural and content patterns on the granularity level of defining small parts of the ontology, solving a specific modelling issue, are the ontology patterns most similar to software engineering design patterns. Such pattern already exist today to some extent, as was described in chapter 3.2.3, and content design patterns will be further explained in section 5.3. The ontology content design patterns on the granularity level of smaller ontology parts are the main focus of the rest of this thesis. The patterns are manually designed or re-engineered based on other constructs, such as data model patterns, as we shall see later.

5.2.4 Syntactic ontology patterns

The final question listed in section 2.3 for summarising considerations during the ontology engineering process was:

- How should the ontology be represented syntactically?

The level of syntactic patterns is intended to address this type of problem, i.e. how the content of the ontology is to be represented. This can concern, for example naming of concepts and properties, or the ontology itself, or how the ontology elements and axioms are represented in some machine processable language in order to be usable by some reasoning engine and in a software system.

A syntactic pattern, strictly speaking, is a recurring combination of representation symbols. This could, in theory, include all previously described patterns on all levels of granularity and abstraction, depending on what is read into the term 'representation symbols', but the focus here is on syntactic representation in a specific language notation mainly for automatic processing by a computer. The distinction between what is a syntactic representation and what is a more abstract modelling language, on the design level for instance, might not always be completely clear since all languages have a syntax.

In this thesis we treat natural language realisations of ontological elements and ontology representation languages intended for use by a computer, e.g. for automatic reasoning, as syntactic representations. This means that a syntactic pattern describes the realisation of, for example a structural or content design pattern in a specific language notation, such

as the XML serialisation of OWL or RDF. We thereby define a syntactic pattern as follows:

Definition 10. *A syntactic ontology pattern is the realisation of other ontology patterns, or parts or combinations of ontology patterns, in a specific representation language.*

This definition adds an additional lowest level of granularity possible for this type of pattern, where single strings and character combinations are the core of the pattern.

Most ontology representation languages have their own common constructs, or in software engineering terms, their own language idioms. Such idioms are syntactic patterns of that language. Other languages can be possible, since ontologies can be translated into other representations, such as natural language. Natural language is a very imprecise way to express ontological elements, but the whole field of OL has focused on extracting ontological elements from natural language texts. Lexico-syntactic patterns, such as Hearst patterns described in chapter 2, are used for extracting ontological features from sentences. These are examples of patterns from natural language that correspond to the realisation of some structural or content design pattern for ontologies. In this sense all such lexico-syntactic patterns can be considered syntactic patterns for ontologies.

Syntactic patterns can be extracted or learnt automatically as in many machine learning approaches that are currently applied to OL, or they may be manually engineered. They can be used both automatically, for example in OL, or manually as descriptions or templates, describing how to model using a specific language. Patterns on this level are most often structural patterns, only describing the structure of some feature without including the actual concepts and relations in question, but in theory there can also be content patterns. An example is a specialised Hearst pattern extracts subclasses to a specific top-level concept. The syntactic ontology pattern level also includes naming conventions, conventions for naming ontological elements like concepts and XML namespaces. Such patterns can be considered as structural patterns, and are on the syntactic abstraction level.

5.2.5 Summary of typology levels

The abstraction levels above, and corresponding levels of granularity, of ontology engineering patterns provide a framework for classifying and describing ontology engineering patterns; a pattern typology. The levels can

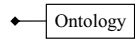
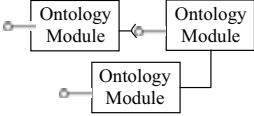
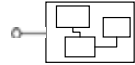

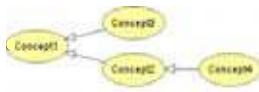


Abstraction	Granularity	Illustration of level
Application pattern	Complete ontology	 <p>Ontology requirements and interface, possibly described in a software architecture description language.</p>
Architecture pattern	Complete ontology	 <p>Overall ontology organisation and parts, possibly described in 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> <...></pre> <p>Representation of parts of an ontology in an ontology representation language.</p>
	Elements of the ontology	<pre><owl:Class rdf:about=""> <rdfo: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 5.1: Levels of abstraction and granularity of ontology patterns.

be summarised and illustrated as in Figure 5.1. Note that the illustrations are informal and are included to give an intuitive idea of what a pattern may contain. Several of the levels have yet no formal language for describing patterns.

On the application pattern level only one level of granularity is possible, the ontology, or ontologies, considered as one unit. The patterns on this level focus on the interface of the ontology and the tasks it is intended to perform, thereby closely related to the requirements of the ontology, but also related to the software architecture where the ontology, or ontologies, are to be used. On the architecture level, there are two levels of granularity possible. Either the scope is the complete ontology and the smallest units are, for example layers or modules of the ontology, or the scope is one such part. Architecture patterns deal with overall structure, either of the complete ontology or smaller parts, but the focus is on general organisation. An architecture is the translation from requirements on the application level to the parts that will realise those requirements, and how the parts will be interconnected.

The design pattern level of abstraction has three possible levels of granularity, i.e. either the complete ontology is considered, the smaller parts solving some specific problems, or each individual element. Design patterns deal with the design of the ontology, i.e. the realisation of the requirements into an ontological model. As far as concerns the complete ontology, this may be a restriction on the used relations, for example specifying the exclusive use of 'subclass'-relations would mean constructing a taxonomy. Patterns showing how smaller parts of the ontology can be realised are restricted by a focus on some specific problem, and the granularity level of single elements introduce a way of expressing ontological elements in a manner independent of syntactical representation. Finally, syntactic patterns are present on all levels of granularity, and focus on representation specific idioms. The rest of this thesis will deal with patterns on the ontology design pattern level. The proposed OntoCase method mainly uses ontology content design patterns on the granularity level where small pieces of an ontology, for solving specific modelling problems, are described.

5.3 Ontology content design patterns

Within this thesis the main focus is on ontology content design patterns, and the intention is to use such patterns for semi-automatic ontology construction. Of course, in order for such a method to be usable, sufficient patterns must be present. Only very small catalogues of patterns actually exist today and these patterns are quite abstract, in terms of the ontological concepts within the patterns. This means that they are hard to use in an automatic method. The existing patterns are also domain independent, hence they

only introduce very general design solutions. Domain-specific pattern can in addition provide some general reusable background knowledge valid in the domain. Therefore we have extended this existing catalogue of patterns with more domain-dependent patterns. Nevertheless, the patterns are still reusable in many different cases. Below our initial efforts to construct such patterns are described.

5.3.1 Pattern representation

We can imagine several different ways of representing and describing content design patterns. As mentioned in chapter 3 patterns in software engineering are commonly described in natural language, as a guideline in a book, i.e. catalogue, of similar guidelines. A template is commonly used to describe such patterns. Templates are 'tables' with fixed headings where information about the pattern is collected, such as name, addressed problem, proposed solution, examples, consequences of usage. Such templates have also been proposed for ontology patterns, as noted in section 3.2.3, and are usually used for patterns presented to a community, e.g. on the web*.

However, patterns used automatically, e.g. in ontology learning, are rarely described in this way since they are intended mainly for use by a software system. It is possible to develop a template suitable for these kinds of patterns, but in many cases it is simply not necessary because the patterns are not treated by humans. Instead focus could in such cases be on computer processable meta-data about the patterns, but primary focus is on the formal representation of the patterns themselves. Such patterns are formally represented to be directly usable within an OL system, for instance.

In our case we have taken the latter approach, even though the type of patterns used, i.e. content design patterns, is commonly used manually. Presenting the patterns with a template intended for human users is beyond the scope of this thesis. The patterns developed and used here are intended for automatic matching and thereby the most important thing is their formal representation. In the first iteration of our research, as described in section 4.3, the patterns used were represented as F-logic ontologies, while in the second iteration they were translated into OWL, since the available patterns that had emerged during this time were all in OWL. The patterns are thereby expressed as small autonomous ontologies, with their own

*Examples of a template can be seen at the ontology design pattern portal at <http://www.ontologydesignpatterns.org>

namespaces. None of the patterns constructed during this research have any formal dependencies, however the patterns used in the final evaluations as described in section 8.1 sometimes import other patterns, and are thereby less independent than the ones developed here.

5.3.2 Constructing patterns

Considering the current situation, when ontology content design patterns are only now starting to appear but no sufficiently large catalogues are available, there are at least two different approaches for creating such patterns. The first approach is to take a set of existing ontologies and derive patterns from them, i.e. the pattern mining perspective, either done manually or automatically. The other possibility is to develop criteria of 'good design' and construct patterns that reflect these principles.

Unfortunately, deriving criteria for how to design ontologies is a difficult task and requires deep knowledge and long experience of the field. This indicates that such an approach is more suitable as a community approach, where a community of experienced people try to agree on best practices. Such an effort is underway, and a web portal for ontology content design patterns has been opened[†], where patterns can be proposed, reviewed and finally accepted and added to the catalogue after scrutiny by the community. The initial pattern proposals that are at this moment proposed in the portal were in most cases extracted from a top level ontology, which was used as a 'gold standard' of good design. This option would in theory be possible also in our case. However, our intent was to construct domain-specific patterns, tailored to the kind of ontologies we focused on in projects such as SEMCO, and to the best of our knowledge there is no commonly accepted top-level ontology dealing with product development companies and software development.

The first approach, which attempts to extract patterns from existing ontologies has so far not been tested on a large scale. Some minor experiments were conducted in the context of our research by Thörn et al. [201], when graph pattern algorithms were applied in an attempt to discover patterns in ontologies based on a triple representation. The experiments resulted in the conclusion that further research is necessary in order to tune such algorithms if useful patterns are to be discovered. The main problem is that graph algorithms do not generally take into account any kind of semantics of the graph elements, thereby proposing a huge number of nonsense patterns.

[†]<http://www.ontologydesignpatterns.org>

Although interesting patterns may be found among these, the noise ratio is too large. Additionally, finding such ontologies pose a problem for our specific focus on enterprise ontologies. Enterprise ontologies are so far quite scarce, but more important most of them are proprietary to the enterprises they describe.

Due to these issues we tried to find additional possibilities for pattern construction. A natural idea, since we were conducting research on knowledge reuse, was to reuse knowledge from other areas for pattern construction. Knowledge already accumulated in other areas can act as inspiration or may even be directly transformed into ontology design patterns useful for enterprise ontology construction. Many patterns in other areas of computer science also describe some kind of knowledge about enterprises, although they might have a different purpose than constructing ontologies. Our approach was to study patterns from other areas in order to develop initial candidates for ontology patterns. For our first experiments in the area, some sources of pattern-like constructs were chosen, among others sources from data modelling, software engineering, knowledge and software reuse, for example reuse of problem-solving methods.

One of the areas most similar to ontology engineering is the database domain. Database schemas are modelled to describe a domain, with the intent of storing data efficiently, but schemas share many properties with ontologies. Schemas can even be considered to be ontologies, from which to extend data instances. Such schemas may be particularly suitable when, as in our case, the ontologies in focus are ontologies describing information within enterprises. One major application area of databases is storing enterprise data. Based on this reasoning we intended to benefit from reusing this knowledge, and our first attempt was to 'transform' the knowledge stored in data model patterns into ontology content design patterns. Some parts of the data model patterns were left out for this initial experiment, such as cardinality constraints. The aspects that were taken into consideration and their chosen mappings to ontology elements are shown in table 5.1. A frame-based logic was the formalism used, at that time, for our ontology design patterns, which is based on quite similar foundations as the entity relationship (ER) model underlying the data model patterns.

To illustrate this mapping Figure 5.2 shows a very simple data model pattern that describes organisations of different types. The data model pattern was proposed by Silverston [180]. The notation in the figure is the one used by Silverston [180], where graphical inclusion is intended to denote a subtype-supertype relation. This pattern was then translated into the on-

Table 5.1: Mappings between data models and ontologies.

Data Model	Ontology
Entity	Concept
Attribute	Relation to 'attribute'-concept or literal
Subtypes/Supertypes	Subsumption hierarchy
Relationships	Relations
Mutually exclusive sets	Disjoint concepts

tology content design pattern depicted in Figure 5.3, using the visualisation capabilities of the ontology engineering workbench KAON[‡].

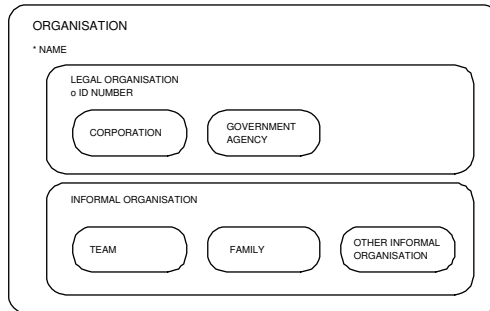


Figure 5.2: Data model pattern describing organisations. [180]

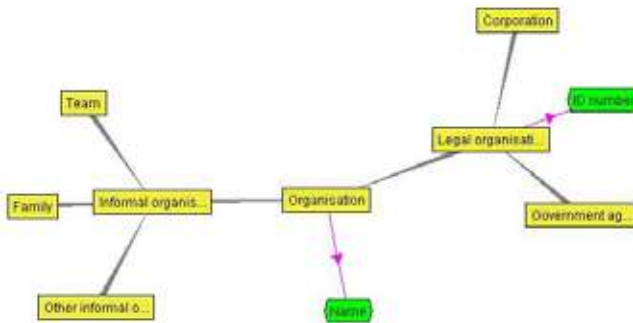


Figure 5.3: Resulting ontology content design pattern.

[‡]<http://kaon.semanticweb.org/>

Table 5.2: Chosen mappings between goal structures and ontologies.

Goal Structure	Ontology
Node (goal or method)	Concept
AND-relations	Part-of relations
OR-relations	Choice relation (related to a new concept representing the category of possible choices)
Iterations	Part-of relations

Another source of patterns was the goal structures of the object system models, presented by Sutcliffe [199]. These constructs can be considered as representing processes within a company, consisting of a set of goals and choices for how to accomplish them. Table 5.2 shows the mappings chosen to adapt a goal structure for use as an ontology design pattern.

Since the naming of the concepts in the source areas might not be adequate for ontologies, and ontologies aim at describing concepts and not mainly terms, the resulting ontology design patterns were enriched with label synonyms. The synonyms used were manually selected from WordNet synsets, as described in section 2.2.3. This gives the patterns a certain level of abstraction, since the concepts are not represented by one single lexical term, but actually a set of terms, as described in section 2.3 with respect to ontology learning. At this stage only basic parts of the ontologies were considered, for example only simple axioms were incorporated into the patterns, such as disjointness of concepts. In future experiments more complex axioms can be developed for the patterns, in order to make the resulting ontology design patterns more useful and precise.

5.3.3 Pattern catalogue

The set of patterns constructed consisted of 26 patterns. In Table 5.3 the patterns are listed, together with their sources. A list of the patterns with more details of their content can be found in appendix B. The set of patterns was constructed with a specific domain in mind, namely product development companies. Some sources were general, but many of the patterns are concerned with concepts such as product development, requirements analysis and parties and processes of an enterprise. This makes the initial catalogue somewhat domain-dependent, but not with respect to industry domain, merely with respect to the type of enterprise.

Table 5.3: Patterns and their original sources.

Pattern name	Source
Actions	Analysis pattern [69]
Analysis and modelling	Goal structure [199]
Communication event	Data model [180]
DOLCE Descriptions and Situations	Top-level ontology [80]
Employee and department	Data model [180]
Engineering change	Data model [179]
Information acquisition	Goal structure [199]
Organisation	Data model [180]
Parts	Data model [179]
Party	Data model [180]
Party relationships	Data model [180]
Party roles	Data model [180]
Person	Data model [180]
Planning and scheduling	Goal structure [199]
Positions	Data model [180]
Product	Analysis pattern [69]
Product associations	Data model [180]
Product categories	Data model [180]
Product features	Data model [180]
Requirements	Data model [180]
Requirements analysis	Goal structure [199]
System	Cognitive pattern taxonomy [82]
System analysis	Cognitive pattern taxonomy [82]
System synthesis	Cognitive pattern taxonomy [82]
Validate and test	Goal structure [199]
Work effort	Data model [180]

A majority of the patterns were constructed on the basis of the belief that they would truly constitute 'good' ontology design patterns, but a few were included although it was beforehand doubtful they would constitute very good patterns. This was in order not to presuppose any conclusions of their usefulness, and instead later be able to validate or falsify this intuition through experiments. It was considered unwise to restrict the pattern sources to just one or two at this early stage. The source believed to be most unreliable was the patterns adapted from software analysis patterns, as described by Fowler [69], since these had a firm focus on software and not on any kind of knowledge or information.

Later in this research, when the evaluations of the OntoCase method were conducted this catalogue was extended with existing patterns from the ODP portal [§]. The reason for not including those patterns in the initial phase was that this portal did not exist at the time. The extended pattern catalogue is described in section 8.1. Some example patterns can be viewed in in appendix B.

[§]<http://www.ontologydesignpatterns.org>

5.3.4 Are patterns really useful?

The question whether or not patterns are truly useful and provide all the benefits that were proposed in section 3.2 is not easily answered. Even in software engineering there were only a few attempts to experimentally prove the benefits of patterns. Later in this thesis the content design patterns, as described above, will be used semi-automatically and shown to provide some benefits compared to existing OL methods. But what about content patterns in general? Although not the main focus of this thesis some selected results of a small study conducted within the context of the NeOn project[¶] will be presented here. The intention was to provide at least some initial evidence that ontology content design patterns are really worthwhile, in this case used manually. More important for the scope of this thesis however, the results point at some findings indicating that automatic support for pattern retrieval and reuse is really needed. These results act as an additional motivation for proposing a semi-automatic method, on top of the theoretical motivation in section 1.2.4, where the research questions were formulated. The complete study was presented in the report of Dzbor et al. [55]. In this thesis only a small subset of the collected results are presented, i.e. mainly those that are relevant to indicate the requirements for automatic pattern-support.

Motivation and hypotheses

Before proposing a typology as the one presented above in this chapter the first question any researcher should ask is, do we really need such patterns? In chapter 3 three proposed benefits of patterns were taken from related literature:

- Reuse
- Guidance
- Communication

Reuse benefits are described as the possibility to bootstrap solutions and provide increased quality. Guidance benefits are concerned with patterns acting as guidelines and inspiration, and patterns pointing out common problems. Communication benefits mean that users can more easily refer to specific problems and solutions by referring to patterns, and patterns can

[¶]<http://www.neon-project.org>

be part of documentation. Guidance and communication benefits are quite obvious and intuitive. It is not more complex than the fact that proper training and using a common language has benefits when designing any kind of artefact. The reuse benefit is more controversial, even in software engineering there is still an ongoing dispute whether or not systems are really better, and if processes are really more efficient, when patterns are used. At least the aspect of reducing development time has to the best of our knowledge not yet been proven, and it is unclear if this aspect actually exists.

Ontology design patterns have a different intent and usage than software patterns. Since these patterns are often represented as reusable solutions, rather than guidelines in a book, this would indicate that perhaps reuse benefits can be achieved in the case of ontology patterns. On the other hand to reuse pieces of solutions rather than reading about good practices in a book also puts added requirements on tool support for doing so. This was part of the focus of the study, and the main focus of the parts presented in this thesis.

With this in mind the experiment conducted intended to show that from a user perspective content design patterns are perceived to be beneficial. In this case the 'users' are users of the patterns, i.e. people designing ontologies. If possible we also intended to provide some evidence for an increase in quality of the produced ontologies, and investigate other characteristics of pattern-based design. The study was based on a set of hypotheses, among these the following two hypotheses are treated here:

1. Ontology content design patterns are perceived as useful by ontology designers when constructing ontologies.
2. The quality of the constructed ontologies is improved when using ontology content design patterns.

In addition, and more important for this thesis, problems were also collected and suggestions for support that would be beneficial for ontology engineers when reusing patterns were suggested.

Experiment setup

Ideally the design of such an experiment should have had one group using patterns and a control group doing exactly the same tasks not using any patterns. In reality this setup turned out to be too resource demanding, and subsequently a reduced setting had to be adopted. In this setting the

main idea was for the subjects to perform two modelling tasks, the first one aiming to establish the level of modelling ability of the subjects before introducing them to the notion of ontology patterns and the second one, set in a different domain but containing similar modelling issues, aiming to record the ability of the subjects with the assistance of the patterns. Throughout the experiment the subjective opinions of the subjects were collected through questionnaires, and the ontologies produced were stored for later analysis.

The experiment was conducted in three separate settings, with a different amount of training of the participants in each setting. In total, the experiment involved 45 subjects in two locations, divided over the three settings. Before the experiment, all the subjects had to fill out a background questionnaire about their previous knowledge and prior experience in ontology modelling.

The first setting was a PhD course conducted over four full days at the University of Bologna, where ontology engineering and design was the major topic. The subjects were mainly PhD students and junior researchers, with limited background in ontology design, but reasonable knowledge of modelling and with a high general level of motivation. In this setting the first day was devoted to lectures and a simple exercise to introduce the subjects to the OWL language and the modelling tool to be used, i.e. Top-Braid Composer. The second day the subjects had more specific lectures related to modelling in OWL, but without mentioning patterns, and in the afternoon the first modelling task was solved during 2 hours. After completing the task the subjects filled out a first questionnaire recording their experiences during the session.

The following day was spent teaching ontology design patterns, and in order to practise the subjects had the opportunity to redo the task from the day before, now using patterns from a catalogue^{||}. After this day a questionnaire captured their first impression of ontology content design patterns. Finally, on the fourth day they were given a second task but the same pattern catalogue as the day before. The new modelling problem was designed to contain similar modelling issues as the second day, only set in another domain, and they had the same time, 2 hours, to solve it. This final modelling task was also concluded with a questionnaire. Ontologies from all the tasks were collected and stored.

^{||}<http://www.ontologydesignpatterns.org>

The second setting was a special purpose setting for this experiment, not set during any course or other teaching event. The subjects were researchers and PhD students from two research groups at the School of Engineering, Jönköping University. In this setting the previous experience of ontologies and ontology design was slightly higher, but the incentive to learn was on the other hand lower, since this was not a course selected by the subjects as in Bologna. This setting was condensed into less than one day, consisting of an initial lecture on the essentials of modelling with OWL, then the first modelling task, which took only 1 hour, and subsequent questionnaire, a new lecture on ontology content design patterns and then directly afterwards the last modelling task, which also took 1 hour without the opportunity to first practise the use of patterns, and the corresponding questionnaire.

The third setting was within a master's course on Information Logistics at Jönköping University. This setting was in terms of training and tasks directly comparable to the first setting in Bologna, the only difference being the experience level of the subjects and the fact that the lectures and tasks were spread out over one month and not condensed into one week. The backgrounds of the subjects were more diverse this time, and the experience level quite low. Most were master students who had only taken one basic course in information modelling. The motivation to assist in the experiment by answering questionnaires and producing high quality ontologies was considerably lower than in the other two settings.

The modelling tasks were quite small, intended to be solvable within less than two hours by a person reasonably familiar with modelling in OWL. The domain of the first task was the music industry, containing concepts such as musicians, bands, records, songs, albums and album reviews. The domain of the second task was hospitals as a workplace, containing concepts such as hospitals, nurses, unions, representatives, and locations. In both tasks the subjects needed to model notions like time intervals and points in time, n -ary relations connecting several concepts, and roles held by people at a certain point in time.

The questionnaire results were summarised and collected into statistics about subject backgrounds, reflections on the tasks, which indicated how the subjects perceived the tasks and the patterns, and a collection of suggestions and common problems noted by the subjects. The ontologies were then analysed based on the notions of coverage, usability, common problems, and pattern usage. Since the tasks were quite small they could be divided into more or less 'atomic' problems, solvable in different ways, but that needed to be covered by the solutions. The coverage of the ontologies over the problem

definition was then assessed by looking for solutions for each atomic problem in each ontology. Usability in this case meant the understandability and reusability of the ontologies, and was assessed by studying how well good practices such as naming conventions, labelling and commenting, providing formal definitions and inverse relations, were followed. Common problems were identified based on collecting issues with respect to modelling choices, that may create certain problems when the ontology is used, within the ontologies. Pattern usage was registered by looking for instances of patterns in the solutions, along with 'unintentional' usage, i.e. when a pattern was used in the first task even though patterns had not yet been presented. In addition a set of common mistakes, and issues not considered to be errors but rather 'bad practises' were collected. In this thesis we only present a selection of the results, for a complete account of the results with examples and collected experiences please refer to the report by Dzbor et al. [55].

The first questionnaire enquired about the background of the subjects, some differences between the groups were noted. In summary, the subjects were mostly master students, PhD students and junior researchers. Most had had some contact with the notion of ontologies, but only, for example through a course they had taken, or by hearing about ontologies in a research project. Only about 4 out of 45 subjects were more experienced in ontology construction, i.e. they had been directly involved in ontology design and research on ontologies. For the rest of the subjects the experience level was more on the level of modelling in general, e.g. UML and ER-modelling, and possibly having constructed some small light weight example ontology. 10 of the subjects had some previous knowledge of the tool used in the experiment, and a majority had tried some ontology modelling tool at some other time. Only one of the subjects had ever used ontology patterns before this experiment, but none had used the kind of patterns tested in this study.

The two tasks were presented slightly differently, in order to ensure that the second one would not be easier in itself, and thereby introduce a bias in the study. Another reason was to compensate for the increased general training in modelling ontologies that the subjects got during the course of the experiment. Therefore, the first task was presented as a text, but already divided into short sentences that were convenient to model one by one. The text consisted of several sentences and each modelling problem was expressed very clearly and explicitly. In the second task, the complete modelling problem was expressed in just one sentence, only implicitly containing some of the modelling problems stated more straightforwardly in the first exercise. No help for dividing the text into manageable pieces for

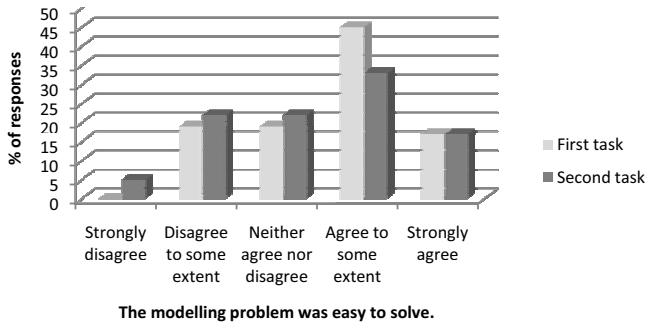


Figure 5.4: The perceived level of difficulty of the modelling problem.

modelling was given. According to the subjects' own perception, the first task might have been slightly easier to solve, see Figure 5.4. Although the reasons for perceiving the problem as harder to solve may be different, we felt that the attempt to leverage any bias in this way was successful.

Experiment result summary

During the course of the whole experiment most subjects stated that the tasks were clear and easy to understand, the problem was relatively small in size, the domain was familiar and the tool was reasonably easy to use. This provided a good background for studying the mistakes that were nonetheless made, how the tasks were solved and how the tasks were perceived by the subjects. Even though the tasks were found to be clear and easy to understand the general opinion was that the tasks were not so easy to solve, and that for all the tasks problems were discovered, during modelling, that resulted in the remodelling of some parts.

The patterns in themselves were received with mixed feelings by many of the subjects. Table 5.5 shows two questions in the same diagram. The first one was if the subjects felt that some of the patterns were obvious and trivial, and we can see that this was the case for some patterns. The second question was if some patterns provided useful solutions that the subject had not thought of before looking at the pattern. In this case, too, we can see that many of the subjects agreed. These results are a manifestation of the constant pattern trade-off, between being very specific and useful or being very general and highly reusable. A small and very general pattern would probably be perceived as trivial, while a specific and complex pattern

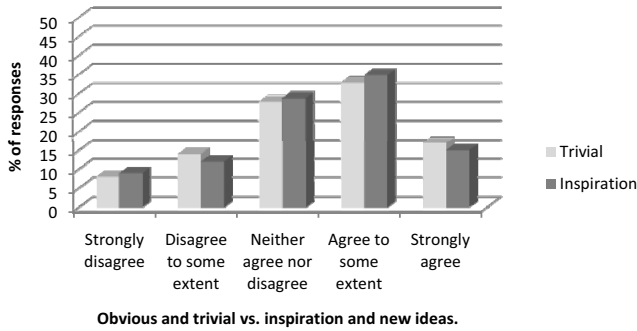


Figure 5.5: Trivial and inspiring patterns.

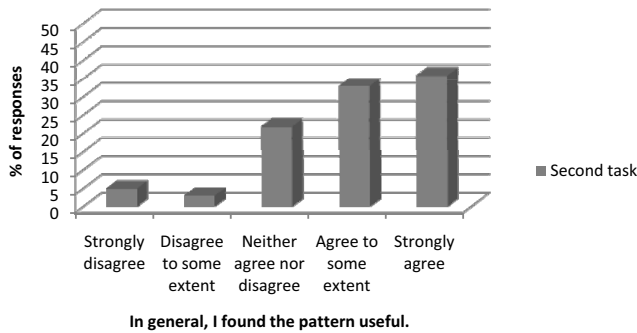


Figure 5.6: Overall perceived usefulness.

would be perceived useful if it solved a particularly difficult task the ontology engineer had in mind, but it would not be widely reusable. When asked directly most of the subjects agreed that the patterns were in fact useful, as can be seen in Figure 5.6.

The next thing we wanted to determine was how and why the patterns were useful. Three questions were then put to the subjects regarding in what way the patterns had helped them. For the subjects doing the pattern training exercise, redoing the first task, this question was posed after the exercise. It was answered by the others after the second task. We can see in the diagram in Figure 5.7 that there is a tendency towards the opinion that tasks were easier and that they produced a better result when using patterns, but there was no support for the task being completed faster. Another question was posed, asking if they were satisfied with their results

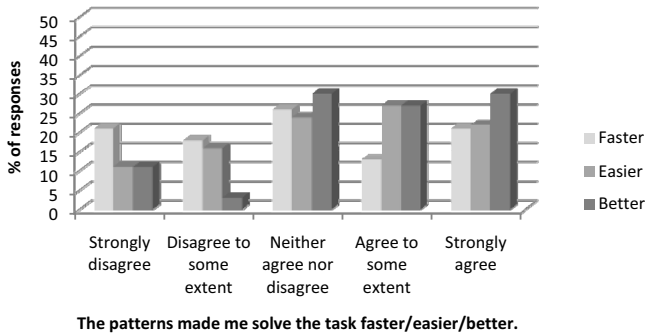


Figure 5.7: Using patterns, was the task solved faster, easier and better?

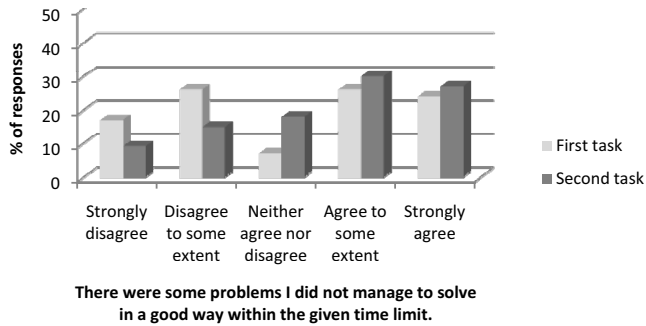


Figure 5.8: The extent of managing to solve problems within time limit.

after the maximum time given for the tasks, as seen in Figure 5.8. This also supported the impression that in fact the subjects perceived themselves as slower when using patterns.

To further study how the subjects perceived the whole process of being introduced to patterns a question was included which asked them to compare the different tasks they had solved. The two groups that did the complete set of exercises and tasks were asked to compare all four exercises, where the two main tasks of this experiment were number two and number four. The group doing only the experiment tasks, without the practise sessions, was asked to compare their two tasks. The questions were formulated so that they had to first select the task they had perceived as the easiest to solve, and then pick the task where they believed that they had solved the modelling problem most successfully, produced the 'best' model. Fig-

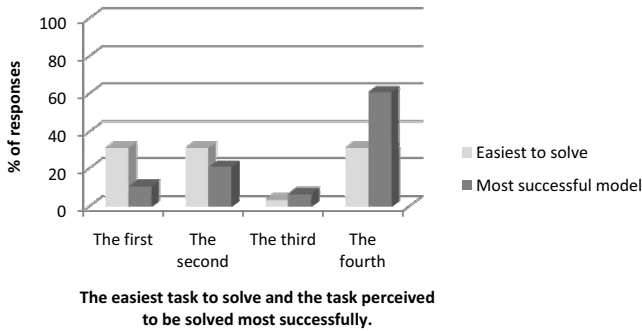


Figure 5.9: The easiest task and the 'best' ontology.

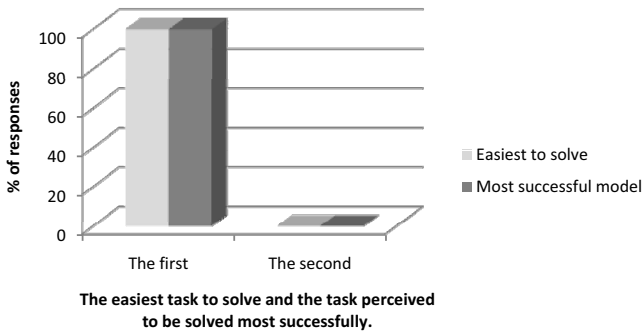


Figure 5.10: The easiest task and the 'best' ontology.

Figure 5.9 and Figure 5.10 show the answers. We can conclude that there is a big difference between the two groups. All of the subjects that received minimal pattern training and went straight for the final task, without exercising patterns, felt that the first task was easier and that they produced a better result the first time. While the ones that received more training to a high extent agreed that the best ontology was produced in the final task. Nevertheless, the two groups did not perceive the same task as easiest, but we could conclude from their answers that the hardest task was most likely the one where they exercised the patterns for the first time. A conclusion that can be drawn is that patterns are quite difficult to understand and to use, and patterns are not immediately useful, not without proper training or proper tool support.

A set of common problems were also stated, concerning how to use the

patterns. Some of those problems were:

- Understanding the patterns.
 - Too brief descriptions of some patterns.
 - Missing examples.
- Difficult to match between the specific domain and the general patterns.
- How to choose between patterns.
- How to use only parts of a pattern, and how to extend patterns.
- How to combine patterns.
- Missing patterns.

Next, the ontologies were analysed, to achieve a more objective assessment of the usefulness of the patterns and to identify remaining problems. The coverage was generally high for most ontologies, which indicates that the subjects had managed to solve most parts of the problems within the time limit. Concerning usability, all of the ontologies resulting from the first task lacked comments, and many also lacked concept and property labels, as well as disjointness axioms. Usually some formal definitions of concepts were included and some inverses of selected properties. The ontologies resulting from the second task, in which they all included some concepts and labels, for instance, showed considerable improvement in all respects.

Common problems in the ontologies from the first task were how to model n-ary relations, how to distinguish between persons and their roles and between an abstract information object and its concrete physical realisation, i.e. songs and tracks in this case. Naive solutions were provided, for example modelling roles as subclasses of persons, which in turn resulted in the problem that a person cannot have different roles in different settings or during different time periods. In the ontologies that solved the tasks correctly, most of the solutions were more or less identical to the ones suggested by a corresponding content pattern, the so called 'information realization' pattern, something that the subjects at the time had not yet seen.

Many of these issues were in the ontologies of the second task in fact solved by the means of directly reusing content design patterns. Most subjects used several patterns in their ontology, and most of the common problems from the first task did not occur at all in the set of ontologies from the second task. Unfortunately, some patterns seemed difficult to reuse, such as the situation pattern, intended to solve the n-ary relations problem, and the collection pattern, for modelling collectives such as labour unions. Therefore, misused patterns could be observed, mainly patterns that were specialised and composed in an incorrect way but also patterns that were used in a context where they were not really applicable. From this we can conclude that patterns are sometimes both hard to match and select, as well as reuse in a correct way. There is no direct tool support for any of these tasks present today. Patterns have to be selected manually, imported into the ontology by means of the owl:import functionality, and then specialised by the ontology engineer. However, patterns did solve most of the modelling issues, and prevented common mistakes, when understood and used in a correct way, whereby these results still confirm the usefulness of patterns.

Analysis and conclusions

Some general conclusions can be drawn from the results above. First let us revisit the hypotheses stated at the beginning of this section. The first hypothesis proposed that ontology content design patterns would be perceived as useful by ontology engineers. The subjects of this study were mostly novice ontology designers, which means that no general conclusion can be drawn, but for such inexperienced ontology engineers there is clear support for the hypothesis. Most of the subjects, 69%, perceived the patterns as either very useful or to some extent useful, while only 8% did not find them very useful. Additional support for this hypothesis is that a majority of the subjects who received proper pattern training, 61%, perceived that they solved the final task better than any other task during the experiment. We were also able to note several improvements in the ontologies themselves that confirm this hypothesis. We cannot, however, find any support for saving time by using patterns. This may be because most subjects were inexperienced, it may also be due to the small amount of training, but we cannot rule out that using patterns can actually increase development time instead of reducing it. Despite this, we conclude that experimental results support the first hypothesis; content design patterns are useful.

Nevertheless, using patterns is no universal solution that will make novice ontology engineers construct wonderful ontologies. The opinions of the subjects, as presented above, confirm that they built 'better' ontologies when using patterns. At the same time, 72% of the subjects concluded that they in the final task still had to, in some way, remodel parts of the ontology during the process, because of issues they did not foresee when they began the modelling. The results of analysing the ontologies themselves, however, strongly support the second hypothesis, i.e. the quality improved both with respect to fewer remaining problems and better usability. We conclude that also the second hypothesis is supported.

However, neither hypothesis was supported without some perceived problems. This issue is also related to the fact that many problems the subjects experienced when using patterns were not supported by any available tool or guideline. Issues related to understanding patterns could be resolved through better training, providing more detailed information about patterns and providing examples connected to the patterns. The issue of providing support for finding, matching, selecting, adapting, extending, and composing patterns is more difficult. As we have noted in previous chapters, there are some basic support for ontology search and retrieval, matching and merging, but these approaches are not tailored to small and general ontologies such as content design patterns. Search can be conducted also on the natural language descriptions of the patterns, or any kind of metadata, but this in turn puts us back in the scope of the first problems noted, that pattern descriptions are currently insufficient and sometimes unclear. Additional tool support for, semi-automatic, pattern selection and reuse is definitely needed.

To summarise this discussion, patterns are in fact useful and increase the quality of the produced ontologies, when used correctly. Nevertheless, there are many unresolved issues regarding the use of patterns, and there is currently very little tool support to meet these issues. We see this as an additional motivation for proposing a semi-automatic ontology construction method that exploits patterns. The method proposed in the following chapters, called *OntoCase*, attempts to exploit patterns by automatically matching them to ontological elements learnt from text or present within an initial ontology, then selecting a set of patterns, adapting and pruning them, and finally composing them into a draft ontology. In addition to improving current methods in ontology learning, *OntoCase* becomes one step in the direction of providing support for exactly the problems perceived by the subjects of this study.

Chapter 6

Initial method, industry evaluation and experiences

This chapter describes the first phase of this research, which involved developing and implementing an initial method for pattern-based semi-automatic ontology construction. The method was applied in an industry case, which gave valuable insights to the requirements and open issues of this problem, as well as valuable experience comparing manual and semi-automatic ontology construction. First, the initial version of the method is described, then the experiments performed within the SEMCO project are presented, and finally the results are analysed and some conclusions are drawn.

6.1 Initial method

Based on the initial literature study and a survey of existing tools, we concluded that many approaches already exist for the different steps of element extraction within ontology learning from text. The approaches include algorithms tailored to extract specific elements, e.g. terms, relations, or specific axioms, for ontology construction. However, so far very few of these approaches continue beyond the extraction of single terms or relations, the results are generally large and diverse, and the approaches rely on manual validation in order to construct the actual ontology from these extracted elements. Our initial idea was to study the feasibility of applying ontology patterns on top of these results, to produce a 'better' ontology that is easier for the ontology engineer process further.

The method that was initially developed can be seen in Figure 6.1. We assumed that a state of the art OL system, providing a set of elements extraction algorithms, would as a minimum produce a set of terms and relations. The idea was to take extracted terms and unnamed binary relations, match them against the patterns and depending on the result use parts of the patterns to build the ontology. As a pre-processing step a text corpus was analysed by a term and relation extraction software, which was assumed to render a list of possibly relevant terms and relations. These lists of terms and relations were assumed to be provided as input for our method.

6.1.1 Pattern matching and selection

Based on the input of terms and relations, the first step was then to match the list of terms against all the patterns in the library. The precise method used for this matching was not fixed, at this stage of our method development, but was restricted to lexical matching. There exist many matching approaches for lexical matching, and for each method a decision has to be made on what grounds to register an accepted match. Later we show one possible realisation of this matching process, using a string matching tool, that was used for the initial experiments.

The first step resulted in two things; a score for each pattern, representing the amount of terms in the pattern that matched the term-list, and a list of the terms in the extracted term-list that were considered to match the pattern at hand. The list of correctly matched terms was used, for each pattern, to extract possible relations in the pattern also present in the text corpus. This relation extraction was outside the scope of our research, since tools for this have already been proposed. The output of this was a list of concept pairs. These connected concepts were then compared to the current pattern and a score was computed based on the amount of relations in the pattern identified among the extracted relations. The exact nature of the matching was not fixed at this stage, an example procedure used for the initial experiments is presented later.

Next, the two scores obtained, matched concepts and matched relations, were weighted and added to form a total 'matching score' for each pattern. Then, a decision was made, according to some threshold value, which patterns to keep and included in the resulting ontology and which to discard.

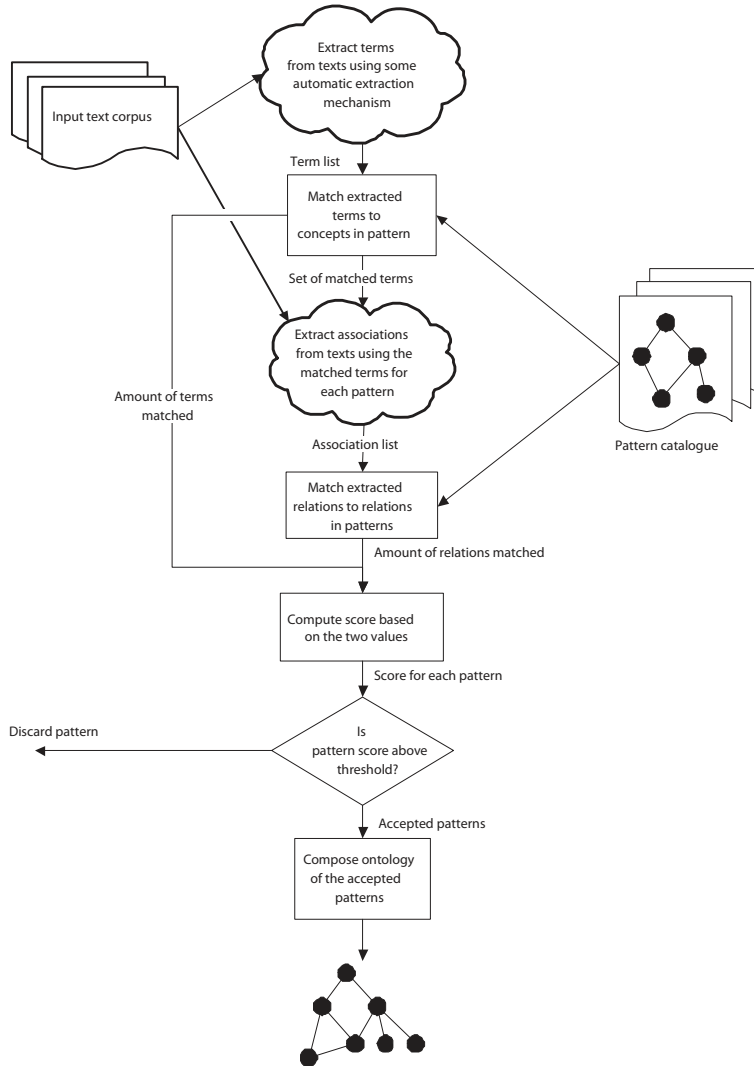


Figure 6.1: The basic steps of the initial method.

6.1.2 Ontology composition

The following step was to build an ontology from the accepted patterns. This construction was based on the matching lists generated, i.e. the lists of matched concepts and relations. The method for building the ontology, by considering one pattern at a time, is depicted in Figure 6.2.

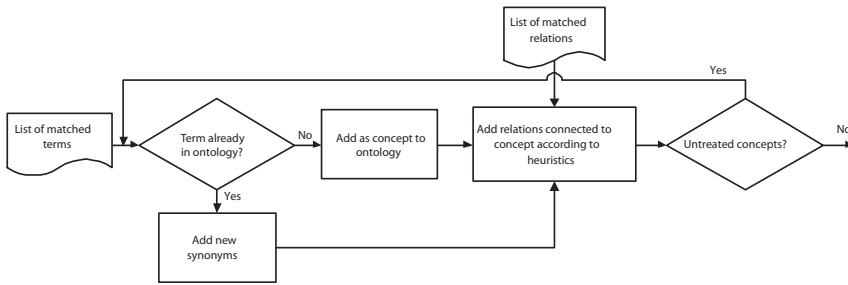


Figure 6.2: Steps of the ontology building, for each accepted pattern.

For each pattern the lists of matches were used as input. Then there were an iterative step which considered all concepts and relations of the pattern. If the term at hand was already in the ontology, no conflicting synonyms was assumed as sufficient evidence for equality, the concept in the ontology was only enriched with all new synonyms of the added term. If the term was not in the ontology already, it was added as a concept along with all matched synonyms of the concept label.

Relations between concepts were added according to a set of heuristics. Relations to and from concepts already present in the ontology, and listen in the matching process previously, were added. Other heuristics, such as adding hierarchical relations directly between concepts which in the pattern are separated only by intermediate concepts, or adding a node if more than a certain number of relations lead to it could be included, but were not used in the initial experiments. The iterative process continued until there were no more concepts or relations of the pattern to consider.

At this point, the focus was to create a process very close to a fully automatic method, thereby drawbacks such as that the patterns might not perfectly reflect what the enterprise actually means by the terms they use were ignored. A fact that reduce the effect of this problem was that if a concept had been matched to the wrong sense intended by the enterprise, then other parts of the pattern would probably not be matched, and

thereby pruned later in the process. It should also be kept in mind that semi-automatic methods should be seen as a complement, an aid, to the manual ontology construction process, whereby there will certainly be a post-processing step performed by ontology engineers before the ontology is taken into use.

6.2 Experiment - SEMCO

In order to validate the approach introduced above we performed an experiment, which was part of the research project SEMCO, as described in section 1.1.2. The scope of the experiment was to construct a selected part of the enterprise ontology for one of the SEMCO industry partners. The purpose of the ontology was to support capturing relations between development processes, organisation structures, product structures and artefacts within the development process. The ontology was limited to describing the requirements engineering process, i.e. requirements and specifications with connections to products and parts, organisational concepts and project artifacts. As a crucial part of this study the same ontology was constructed using two different methods, the semi-automatic method presented in this thesis and a manual method, as presented by Öhgren and Sandkuhl [151]. The construction was performed in parallel with access to the same material and background knowledge, but without any contact or interaction between the development processes. The structure of the experiment is illustrated in Figure 6.3.

6.2.1 Semi-automatic ontology construction

For the semi-automatic construction 26 ontology patterns were developed based on the approach presented in section 5.3.2, and are presented in Table 5.3. The patterns were represented as small ontologies, additionally enriched with concept label synonyms manually selected from WordNet*. The text corpus used as input consisted of software development plans, software development process descriptions, and other similar documents internal to the enterprise in question. The extraction of relevant terms and relations from these texts were performed using the OL tool Text-To-Onto† [127] within the KAON tool-suite‡. This choice was made mainly out of

*<http://wordnet.princeton.edu/>

†<http://sourceforge.net/projects/texttoonto>

‡<http://kaon.semanticweb.org/>

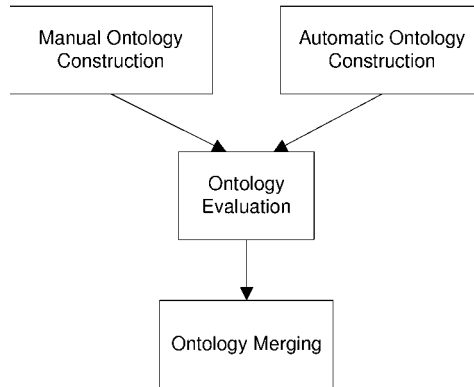


Figure 6.3: The structure of the experiment.

convenience, since this was one of the most elaborate tools and it was freely available on the KAON website. Any evaluation of the methodology would be conducted with the same prerequisite extraction method, so this was not considered a significant bias at this early stage of method development.

In order to keep the experiment on a reasonable scale, to be able to validate the accuracy and efficiency of the method manually, the concepts and patterns were restricted to a relatively low number. An absolute frequency threshold of 10 was set for the term extraction and used during the experiment. This rendered 190 terms as the initial input of the construction process.

The matching of the pattern-concepts and their label synonyms against the extracted terms was done using a lexical matching tool. In order to be able to test different matching algorithms and measures, an existing string matching comparison tool proposed by Cohen et al. [39] was selected for this task, Secondstring[§]. A Jaccard string similarity measure was selected and the threshold for a match was set to a similarity level of 0.5. When this matching was completed for each pattern, a score was computed according to the number of matched concepts. There was also a list of matched terms, matching score above the threshold, which were then used with the Text-To-Onto tool in order to extract relations between those terms.

When matching relations, all relations were assumed to be transitive,

[§]<http://secondstring.sourceforge.net/>

since this simplified the matching task when an intermediate concept did not exist among the matched terms. The score representing the number of matched relations was weighted and added to the score of matched concepts resulting in a total score for each pattern. Since the relations were deemed more important than the lexical matching of concept names, the relation scores were in this experiment given a higher weight than the concept-matching scores.

Most patterns received a quite low score. This was mainly due to the difficulty of extracting relations and thereby also matching relations to the patterns. A quite low threshold score was set, after some consideration and manual evaluation of the relevance of the patterns, and also because by accepting quite a few patterns the properties of the pruning algorithm could be studied more thoroughly. This resulted in 14 accepted pattern out of the original 26.

The 14 accepted patterns were then composed into an ontology using the method specified previously. Each pattern was treated separately, one at a time. For each pattern each of its concepts was considered. If a matched term was not already in the ontology it was included, together with all its matched synonyms. Then all relations leading to and from the concept were considered. Using a set of heuristics some of the relations were added to the resulting ontology.

Descriptions of the heuristics used:

- Include all relations between added concepts, even if they were not matched.
- Use the transitive property of hierarchical taxonomic relations, if an intermediate concept is missing add the child directly at the level of the missing concept.
- An associative relation which originally relates two concepts is added even if one of the concepts is missing, if and only if there is a direct subconcept of the missing concept present in the ontology.

The resulting ontology contains 35 concepts directly beneath the 'root' concept and in total 85 concepts. The ontology is not really divided into subject areas, as can be seen in Figure 6.4, a screenshot from the visualisation tool in KAON. The final choice of ontology implementation language was still to be made at this time, but the internal representation of the KAON tool based on F-logic conformed to the ontology definition used.

The figure shows only a small part of the ontology and some details are hidden to increase readability. Despite this, we can note that products, parts and requirements play a central role in this ontology. Also roles, work and parties appear in the ontology. Already in this small illustration the high number of relations can be noted. For example in the figure the relations tell us that a product will be produced in response to a work requirement, the product is asked for via a set of product requirements, the product has a set of features which can be either available or selected and the product is described in some document.

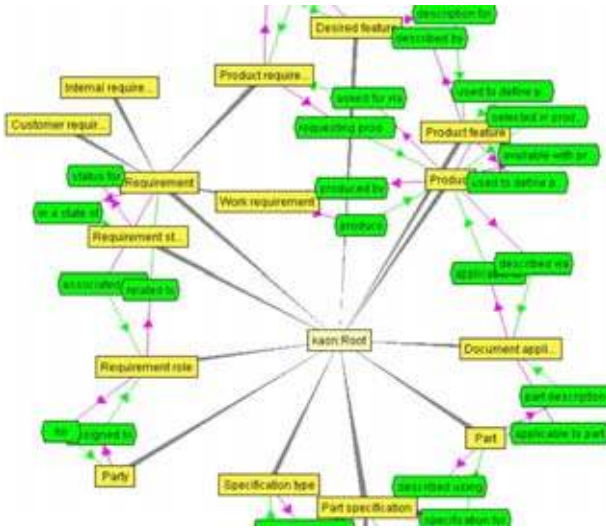


Figure 6.4: Top-level concepts of the aut. constructed ontology.

Figure 6.5 shows an additional part of the resulting ontology. This shows for example concepts concerning products, their features, and their connection to the product requirements. When analysed individually the coverage of the ontology with respect to the extracted terms turned out to only be about 34%, which is a relatively low number. This, together with other characteristics of the ontology, is analysed further in later sections.

6.2.2 Manual ontology construction

The manual construction followed the phases described by Öhgren and Sandkuhl [151]. A user requirements document was produced, which in-

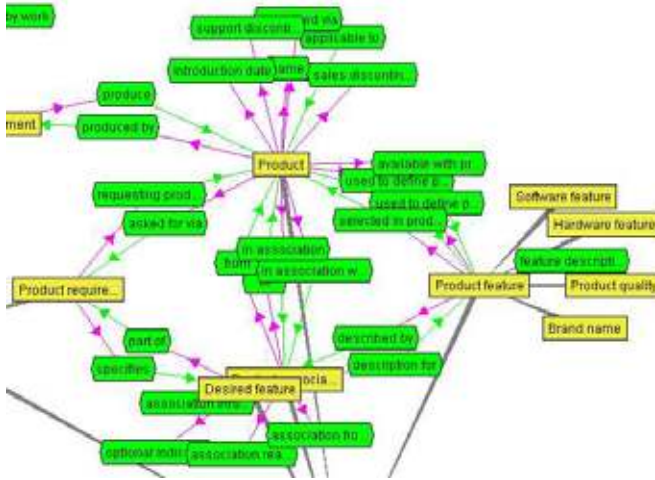


Figure 6.5: A part of the resulting ontology.

cluded identification of existing knowledge sources, defining usage scenarios, and the possibility to find other ontologies to integrate was considered, but no ontologies that were considered relevant were found. The available project documents were used in the first iteration of the building phase. A simple concept hierarchy was built, but natural language descriptions for each concept were deemed unnecessary at this point. It was quite hard to derive relations and axioms from the documents so after document analysis, focus was switched to the other knowledge sources: interviews with selected employees at the company.

Interviews with company employees were performed in two sessions. At the first session the interviewees discussed the top-level concepts, then went further down the subsumption hierarchy, discussing each concept. Feedback was given in the form of suggestions, such as "Restructure this" or "This concept is not that important". After the first interview session, the suggestions were considered and some were implemented. The second interview session was carried out similarly, resulting in minor corrections of the ontology. Implementation of the ontology was integrated into the building phase, the ontology was quite light weight and no language expressivity problems occurred. The last phase of the manual methodology, evaluation and maintenance, was partly integrated into the building phase, where the interviewees reviewed the ontology. Other parts of the evaluation are de-

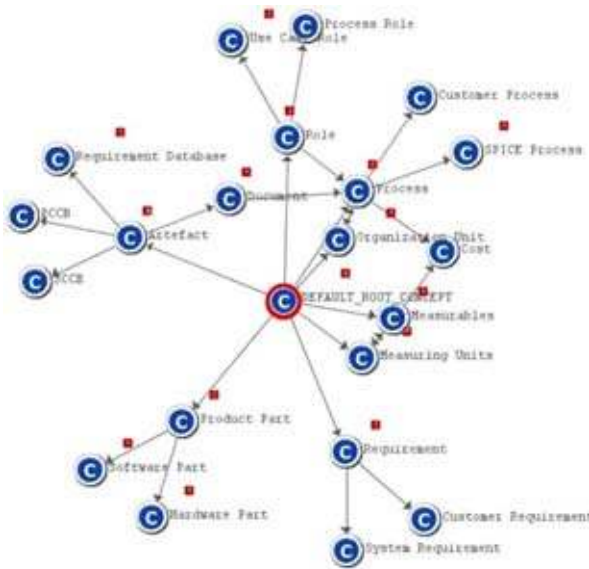


Figure 6.6: Top-level concepts of the man. constructed ontology.

scribed later in this chapter, and maintenance was not performed within the scope of the project.

The resulting ontology has 8 concepts directly beneath the 'root' of the subsumption hierarchy, and 224 concepts in total. Some of the most general concepts are illustrated in Figure 6.6 through a screen-shot from *OntoEdit*, a previous version of *OntoStudio* from *Ontoprise GmbH*[¶]. The ontology representation language was not considered at this point, since the application intended to use the ontology was still in its planning stage at that time. The choice of tool was based on convenience, since the tool was available and the ontology engineer constructing the ontology was familiar with that specific tool.

The resulting manually constructed ontology contains a few major parts, as seen in Figure 6.6. The figure shows only a small part of the ontology and some details are hidden to increase readability. Despite this, the division of the ontology into subject areas can be noted. Directly related to the focus of the ontology are the parts dealing with product parts and requirements. In addition one part deals with artefacts, which denotes dif-

[¶]<http://www.ontoprise.de>

ferent things produced during the product development process, such as documents and requirements. Another important part is the organisation units and the roles present in the organisation, which in turn participate in the processes of the organisation. This was included for connecting roles to both the request and realisation of different requirements and product parts, in different process steps. Finally some supporting areas like quantities and measuring units were included in order to assist when describing requirements and product parts.

6.2.3 Evaluation

This section presents the choice of evaluation methods for the resulting ontologies, and a description of the evaluation and its results.

Evaluation setup

A decision was made to use several evaluation approaches, both intended for ontology expert and domain expert evaluation, to get a broader view of the ontologies and indirectly also the construction methods. A detailed description of the measures and the rationale behind the selection was presented in section 4.2.2.

A general comparison of the ontologies was needed to get an idea of differences and similarities. This comparison was done based on basic structural measures, such as number of concepts, average number of attributes per concept, average number of subclasses per concept, and average number of associations per concept, as for example suggested by Gangemi et al. [77]. Also, the cohesion metrics proposed by Yao et al. [222] were used, since we felt that they complement the other measures well. These metrics are: number of root classes, number of leaf classes and average depth of inheritance tree.

Second, an evaluation was performed by internal ontology experts using the two most well-known approaches for structural evaluation of the taxonomy, presented by Gómez-Pérez [84] and Guarino and Welty [91]. Internal ontology experts were used for these evaluations, mainly because of their previous knowledge of the evaluation methods. Since we were evaluating both the ontologies and, indirectly, the methods for constructing them, the initial idea was that the errors discovered could give valuable indications on advantages and disadvantages of each construction method.

Finally, to evaluate the content of the ontologies at a functional level,

i.e. their fit to the intended scope, a subset of the OntoMetric framework by Lozano-Tello and Gómez-Pérez [122] was used. For our purpose only the dimension *content* was deemed interesting, and only one level of characteristics for each factor. Some characteristics were not applicable to both ontologies and since this was mainly a comparison, these were not considered. Domain experts from the company in question formed the evaluation team, but internal ontology experts prepared the material, assisted through the evaluation, and collected the results.

A desirable method of evaluation would of course have been to apply the ontologies in their intended application context. This was not possible however, since the resulting application of the SEMCO-project was planned to be developed and deployed in a second project that had not yet started at that time.

General characteristics

First the structural characteristics of the ontologies were collected. In Table 6.1 these are presented for the two ontologies, the ontology constructed using the semi-automatic method is denoted 'Aut' and the manually constructed ontology is denoted 'Man'. The results showed that the automatically constructed ontology has a large number of root concepts, it lacks some abstract general notions to keep the concepts together in groups or subject areas. It is also quite shallow and many concepts lack subconcepts altogether. Despite this, the concepts are more strongly related through non-taxonomic relations and have more attributes than in the manually constructed ontology ontology.

The manually constructed ontology, on the other hand, contains a larger number of concepts. It also contains a top-level abstraction dividing the ontology into 'intuitive' subject areas. There are few attributes and relations, this might be due to that many attributes are actually represented by other specific concepts, they were just not connected by an appropriate relation. A lesson learnt was that relations seem to be harder to elicit from interviews than the concepts themselves.

Evaluation by ontology engineers

Two structural evaluation methods were used by ontology experts, general taxonomic evaluation criteria and the OntoClean framework. The ontologies were first evaluated by ontology engineers according to the criteria presented by Gómez-Pérez [84]. The criteria were the following:

Table 6.1: Comparison of structural characteristics.

Characteristic	Man	Aut
Number of concepts	224	85
Number of root concepts	8	35
Number of leaf concepts	180	64
Avg depth of inheritance	2,52	1,95
Avg number of rel. concepts	0,13	0,79
Avg number of attributes	0,01	0,46
Avg number of subclasses	1,00	0,57

- Inconsistency: circularity, partition and semantic errors.
- Incompleteness: incomplete concept classification and partition errors.
- Redundancy: grammatical redundancy, identical formal definitions of concepts or instances.

There were no circularity errors in the automatically constructed ontology since there was no multiple inheritance present, this also prevented most errors belonging to the inconsistency partition errors group. Multiple inheritance in the manually constructed ontology occurred only in a few cases, and no circularity errors were discovered among these. This also reduced the possibility of partition errors, as mentioned previously. There are no exhaustive decompositions or partitions specified in either ontology so this eliminates the possibility of finding any other kind of partition errors.

Semantic inconsistency errors were more subtle to discover. This was a question of identifying wrong classifications. In the automatically constructed ontology there existed two concepts which could be thought of as wrongly classified since they made no sense in the context of this ontology, they were simply 'junk' that happened to enter the ontology due to the uncertainty of the ontology construction process. Semantic inconsistencies could also occur when two overlapping patterns are both included in the ontology, but this seemed not to be the case in the ontology at hand. Concerning the manually constructed ontology these errors could only be assumed to have been discovered in the interview sessions with the domain experts, because at this stage no more such errors were discovered.

Next, the incompleteness criteria were examined. Incomplete concept classifications could exist in the semi-automatically constructed ontology due to concepts missing in the patterns or not explicitly mentioned in the text corpus used to develop the ontology, and thereby not extracted. Since

this was an application ontology, not the complete domain needed to be modelled, only the parts required for this specific application. When comparing the two ontologies however, the semi-automatically constructed ontology seemed to lack more specific concepts, such as document names and company specific terms. This was definitely a problem originating from the semi-automatic process in itself, since it did not at this time contain any way to determine subsumption relations between extracted elements and patterns, only direct overlap. Even for the manually constructed ontology it was difficult to determine the incompleteness criteria until the ontology was to be used in its intended context, but in comparison to the semi-automatically constructed ontology it was more complete on the lower levels of abstraction.

Several occurrences of partition errors were found in the semi-automatically constructed ontology, especially lack of disjointness definitions. This could be included in the patterns in order for it to propagate into the constructed ontologies, or more recent methods, as proposed by Völker et al. [210], could be used to try to discover disjointness directly from the text input. Also some cases of believed exhaustive knowledge omission were found, but on the other hand that knowledge might not be needed for this specific application. In the manual construction process disjointness and exhaustive partitions were not discussed before building the ontology, so it is at this point not certain that there is a need for defining this. Deciding this ought to be part of the construction methodology, i.e. an important addition to the manual methodology.

Finally, there were no concepts with identical formal definitions but different names or redundant subclass relations in either ontology. Redundant subclass relations are not present in the patterns used in the automatic approach and no overlapping patterns had introduced it in this case. However, it is worth studying when considering overlap between patterns, and methods to resolve such issues might be needed in the semi-automatic ontology construction process.

Next, another expert evaluation was performed, this time by using the OntoClean methodology. Every concept in the ontology was annotated with the properties rigidity, identity and unity. This resulted in a backbone taxonomy containing 25 concepts in the semi-automatically constructed ontology. Here two violations of the unity and anti-unity rule were found and one violation of the incompatible identity rule. When analysed the unity problems arose because in this company 'work' was seen as a 'product', but 'work' is generally not a whole. The identity conflict had the same cause,

Table 6.2: Result of the OntoClean evaluation.

OntoClean rule	Man	Aut
Incompatible identity	No	1
Incompatible unity criteria	No	No
Unity/anti-unity conflict	1	2
Rigidity/anti-rigidity conflict	No	No

the issue of 'work' being defined as subsumed by 'product', physical products were identified by a id-number while work was not. This is a quite serious problem which required some consideration to be solved, so that the solution still reflected the reality of the enterprise but did not introduce undesirable properties of the ontology when used in an application.

For the manually constructed ontology, the backbone taxonomy contained 178 concepts. One violation of the unity and anti-unity rule was found, and none of the other kinds of errors. The violation arose between the concepts 'function' and 'code', while a function was a clearly defined unit the concept of code was more abstract and could not generally be seen as a homogeneous unit. This violation existed mainly due to that the abstraction level differed too much among the concepts on the same level of the taxonomic hierarchy of the ontology. The fact that no other violations were found was perhaps due to the simple structure of the ontology, it was very much like a simple taxonomy of terms. A summary of the results is presented in Table 6.2, where the manually constructed ontology is denoted 'Man' and the semi-automatically constructed ontology is denoted by 'Aut'.

Evaluation by domain experts

The third step of the evaluation process included a functional evaluation performed by domain experts from the company in question. The evaluation was done based on a part of the OntoMetric framework proposed by Lozano-Tello and Gómez-Pérez [122] as mentioned in the evaluation setup previously. Only the dimension 'content' was considered and also no final score was computed, since the assessed characteristics were quite few and could individually tell us much about the nature of the ontologies. The evaluation was performed by a team of domain experts, working at the company in question, but the process was guided by an experienced ontology engineer in order to explain the notions to be evaluated to the evaluation team. For the evaluation a standard ontology editor user interface was used, where the

ontologies were visible as graph structures. The ontology engineer assisted the team in understanding the semantics of the ontology primitives and the functionality of the user interface but was not allowed to explain the content of the ontologies, i.e. concepts and relations.

The dimension 'content' contains four factors: concepts, relations, taxonomy and axioms. For each of these factors characteristics applicable in this case were chosen. The scale suggested by Lozano-Tello and Gómez-Pérez [122] ranging from 'very low' to 'very high' in five steps was used for assessment, i.e. for each evaluated issue the evaluation team members had to agree on a subjective score between 'very low' and 'very high'. The characteristics used and the resulting scores for each ontology are presented in Table 6.3, where 'Man' denotes the manually constructed ontology and 'Aut' the semi-automatically constructed ontology.

As presented in Table 6.3, both ontologies seemed to contain an appropriate number of concepts, neither was considered too small or too large and both seemed to cover the intended scope, but the concepts in the manually constructed ontology were deemed more essential. This is most likely due to that the concepts were more specific. The semi-automatically constructed ontology also lacked some general abstract concepts to give it a comprehensible structure, which sometimes confused the evaluation team since the ontology had no intuitive division into subject areas. On the other hand, the semi-automatically constructed ontology contained more attributes and relations, a higher density of each concept, which helped to describe and define the concepts and reduced the need for natural language descriptions of each concept. This fact was found very useful by the domain expert evaluators, and is a valuable finding with respect to how people interpret concepts when no formal definition is present. The evaluators were faster to grasp the meaning of these concepts than some of the concepts in the manually constructed ontology that were only described through their placement in the taxonomy, without additional properties connected to the concepts.

The semi-automatically constructed ontology contained more non-taxonomic relations than the manually constructed one, even such relations that the company might not have thought of itself but which were still valid. This was explicitly noted by the evaluation team. The manually constructed ontology mostly contained relations explicitly stated by the company and easily expressed in words, e.g. either in documents or orally during the interviews. It was the non-taxonomic relations that gave structure and comprehensiveness to the semi-automatically constructed ontology while the manually constructed ontology relied on specificity of concepts and precise

Table 6.3: Results of the domain expert evaluation.

CHARACTERISTIC	SCORE				
	Very low	Low	Medium	High	Very high
CONCEPTS					
Essential concepts in superior levels	Aut			Man	
Essential concepts		Aut			Man
Formal spec. coincides with naming			Aut	Man	
Attributes describe concepts	Man			Aut	
Number of concepts				Man	Aut
RELATIONS					
Essential relations		Man		Aut	
Relations relate appropriate concepts			Man	Aut	
Formal spec. of relation coincides with naming			Aut	Man	
Formal properties of relations	Man			Aut	
Number of relations		Man			Aut
TAXONOMY					
Several perspectives			Man	Aut	
Maximum depth			Man	Aut	
Average number of subclasses			Aut	Man	
AXIOMS					
Axioms solve queries		Man	Aut		
Number of axioms	Man	Aut			

naming, i.e. company specific terms.

The semi-automatically constructed ontology also had a taxonomic structure, even though it lacked both some abstract top-level and the most specific levels, compared to the manually constructed one. Despite this, it was perceived by the evaluation team as having quite a large depth, most likely due to the detailed division of the intermediate levels. This detailed division of the taxonomy was due to the detailed taxonomies present in the patterns used as a basis for the ontology. The manually constructed ontology had a larger number of subclasses per concept since a high number of very specific concepts existed. At a higher level of abstraction the average number of subclasses per concept was comparable between the ontologies.

The number of general axioms was low in both ontologies, and the ones present were simple. More advanced 'business rules' was something that the company might need if the implemented application using the ontology were to function efficiently, especially if extended to handle more advanced use-cases than simple structuring and retrieval of documents, and information within documents. For the manual method the question is how to elicit such rules using interviews, which is not a well-specified task so far. In the semi-automatic method these should be included in the patterns but then needs to be appropriately matched to the extracted elements. So far we are not aware of any method for matching and comparing axioms, this is

probably a suitable focus of future research.

At the end of the evaluation, in addition to the evaluation of the characteristics, an unstructured interview was conducted with the evaluation team members in order to see what parts might be completely missing. Natural language descriptions of concepts was one such item of discussion. For the task to be performed by the implemented ontology the interviewed domain experts thought this was not needed, since it was quite clear from the naming and context how a certain concept would be used. In a longer perspective though, for evolution and maintenance of the ontology, this would still be desirable, since concepts and their meanings can also evolve and change during an application's lifetime.

6.2.4 Analysis and practical consequences in the project

When compared to the extracted elements, the automatically constructed ontology covered only 34% of the terms extracted from the documents. The reason was partly a small pattern catalogue, but when compared to the manually constructed ontology it was mainly quite specific terms that were missing. A pattern selection process is not enough to cover the scope, since patterns are too abstract compared to a text corpus. Some abstract information was also missing since this is not explicit in the texts. The semi-automatically constructed ontology also had some nice features when compared to the manually constructed ontology, e.g. a larger number of relations connecting the concepts and to some extent a better structure. This analysis lead us to believe that the pattern selection and combination approach tested in this initial experiment was not enough to substantially improve on current OL approaches. Ways to introduce a general structure and abstract concepts were needed, as well as ways to incorporate the most specific terms within the context and the structure of the patterns, i.e. to close the abstraction gap between learnt elements and abstract patterns.

Merging of the ontologies

Based on the evaluation results presented previously and the analysis of the benefits and drawbacks of each ontology, it was in the context of SEMCO decided that the two ontologies should be combined to produce the final version of the SEMCO ontology.

Since both of the constructed ontologies were built for the same case, and were partly constructed using the same knowledge sources it was as-

Table 6.4: General characteristics

Characteristic	M	A	C
Number of concepts	224	85	379
Number of root concepts	8	35	5
Number of leaf concepts	180	64	273
Avg depth of inheritance	2,5	1,9	3,5
Avg number of rel. concepts	0,1	0,8	1,3
Avg number of attributes	0,0	0,5	0,1
Avg number of subclasses	1,0	0,6	1,0

sumed that they used approximately the same terminology. Methodologies commonly used for ontology merging do not take into account such special features, but rather solve problems related to distinctions in definitions and conflicts between terms. A major function of existing matching and merging tools is that they are able to give suggestions on candidates that should be merged, or have a relationship. In our case it was fairly obvious what terms should be connected and what terms that were possible to merge into one concept. Based on this, the merging process was performed manually and the KAON tool-suite was used for the implementation. The reason for this was still convenience, since the more complex relations and some axioms of the automatically constructed ontology were already implemented using this tool. The process started with an 'empty' ontology, where parts from each ontology were entered in turn, starting from the top of the subsumption hierarchy.

First, the top-level concepts of the manually constructed ontology were added, together with all relations between them. The former top-level concepts of the automatically constructed ontology were thereby grouped as subconcepts of this structure. This step also resulted in some slight re-organisation of the top-level concepts, and the addition of some intermediate concepts, to make the two ontologies fit together and to achieve a more intuitive structure. It was also considered important that all the siblings of a concept were on the same level of generality.

Second, the most specific concepts from the manually constructed ontology were inserted into the ontology, at the bottom level of the subsumption hierarchy. The fit between the two ontologies was not always perfect, therefore some new intermediate concepts were introduced. This was due to the same reasons as the re-organisation of the top-level, to achieve an intuitive structure and to have all siblings on the same level of generality. All re-

lations and attributes from the manually constructed ontology, which were not already in the automatically constructed ontology, were also included.

This process resulted in an ontology with 379 concepts, where only 5 of them were placed directly beneath the root of the subsumption hierarchy. A summary of some general characteristics of the ontology is presented in Table 6.4, in the table the combined ontology is denoted by C , together with the values of each ontology before the combination, the manually constructed one denoted by M and the automatically constructed one denoted by A . The average measures represented the average over all concepts in the ontology. The intermediate 'glue'-concepts that were added during the combination process amount to 18% of the total number of concepts.

During the merging process, care was taken to make sure that no new errors, i.e. of the type described in 6.2.3, were introduced. The manual merging process made sure that there were no language level syntax mismatches. Language expressivity problems were avoided partly due to the fact that neither of the ontologies were very complex and the general axioms that did exist were fairly simple and partly because both ontologies were expressed in F-logic based languages. Since both ontologies were constructed for the same enterprise and task it was also possible to avoid ontology level mismatches.

The most interesting evaluation is however not yet performed, since the applications where the ontology can be used are only partially developed and not yet deployed. To apply the ontology is the only way to test how well it actually performs its tasks. The ontology was implemented in the KAON tool-suite. The top-level concepts of the resulting ontology can be viewed in Fig. 6.7. In the figure some parts of the ontology are excluded due to readability reasons. The final ontology contained many of the same subject areas as the manually constructed ontology described previously at the top level, such as roles, processes and parties. In the combined ontology two new top categories were introduced, further grouping the concepts concerned with products and their features, namely 'work product' and 'feature'. A work product is anything that is produced by a process, e.g. requirements and specifications as well as the product itself. The feature concept grouped all features of products and processes.

When the application scenarios in the following section were developed in detail, a decision was made to export the ontology to Protégé^{||}, but this was conducted through the tools' import and export capabilities, i.e.

^{||}<http://protege.stanford.edu/>

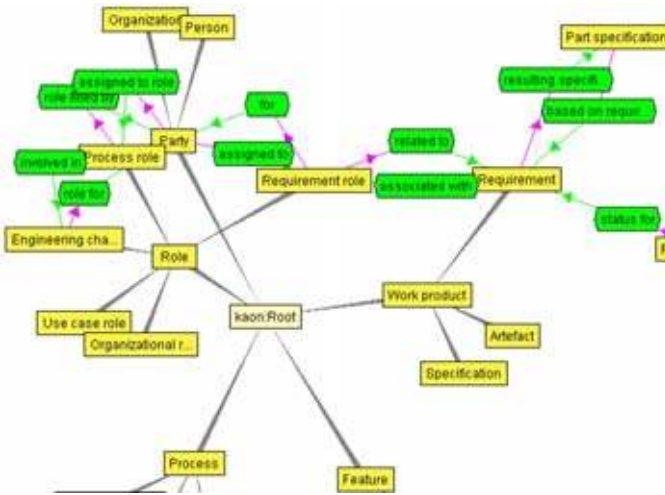


Figure 6.7: The top-level concepts and relations of the resulting ontology.

still without introducing any mismatches since Protégé conforms to a similar frame-based ontology representation formalism as KAON. In addition an experiment was made to transform the ontology into the standard web ontology language OWL, but this time some problems arose. The most frequently encountered problem was the issue of classes as property values, which is not easily expressible in OWL, as described by the W3C [3]. The conclusion was that for our application scenarios the Protégé internal ontology formalism would be used.

SEMCO applications

There were several applications of the ontology envisioned in the SEMCO-project, but so far only one of these have been implemented, however not yet deployed. This scenario was the ontology-based artefact management, supporting reuse and comparison of artefacts between projects, as a part of the more general framework of realising a domain repository for the development processes. The artefact management tool was constructed as a plug-in for the ontology development environment Protégé and is currently called ArtifactManager, as described by Billig and Sandkuhl [19]. The main idea of the artefact management tool was to use the enterprise ontology to define and store metadata and attributes of an artefact, as well as a link to the artefact itself. The enterprise ontology provided the attributes and the

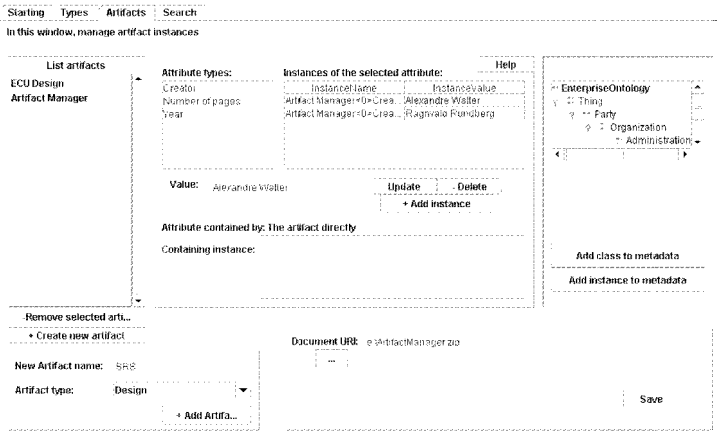


Figure 6.8: The Protégé ArtifactManager plug-in. [19]

metadata, and artefacts were attached to it as instances, and connected to instantiated attributes. In this way the actual artefact can be connected to a part of the enterprise ontology and have attribute values corresponding to instances of enterprise ontology concepts.

When artefacts have been stored they can be searched, retrieved, and compared using their connection to the enterprise ontology. The search is divided into attribute search and ontology search, where attribute search focuses on common keyword-based search of artefact attributes. The search possibility involving the ontology is based on searching for concept paths similar to the artefact metadata. The user could also create his own ontology, as a query, and compare it to the enterprise ontology, instead of selecting parts of the enterprise ontology itself. In that case the search was performed as ontology matching. The artefact management tab of the ArtifactManager plug-in is shown in Figure 6.8.

The ArtifactManager plug-in was used by experienced engineers involved in the project, but not evaluated together with the intended users at the enterprise. Hence the final evaluation of the ontology in this scenario is yet to be performed. However, it can be noted that the final merged ontology covers all the major parts concerning common artefacts in the requirements engineering phase, which leads us to believe that it can be used directly with the ArtifactManager without any further configuration or modification.

A second scenario proposed for future extensions of the SEMCO project was the integration of feature models and enterprise ontologies, by supporting the feature metamodel in the enterprise ontology as proposed by Thörn et al. [202], with the aim of identifying similar requirements and product features in future projects. The features might be related also to the organisational elements in order to track responsibilities and expertise. The aim of the application is to generate internal requirements directly from detected features in the source documents, i.e. customer requirements, and the enterprise ontology and its feature model, based on semantic similarities between the source documents and stored requirements of previous projects.

Requirements for improved method

Based on the evaluations, the analysis of their results and the consequences of this in the context of the SEMCO project, some open issues were listed that needed improvement. These open issues led to the development of the complete OntoCase framework presented in the following chapter. The main open issues identified, with respect to the subproblems of semi-automatic ontology construction, in the initial method were:

- Problem analysis
 1. Lack of possibilities to include specific requirements and to focus the construction process.
 2. Lack of assistance for finding the 'right' input.
 3. Lack of assistance for finding the right pattern catalogue, determining its appropriateness and extending it if needed.
- Input processing
 1. Lack of support for importing additional file formats.
 2. Lack of possibility to start from already existing models and ontologies instead of plain text.
- Element extraction
 1. Incorporate additional existing algorithms.
 2. Lack of possibility to extract complex axioms.
- Ontology composition

1. Lack of possibility to bridge the abstraction gap between patterns and extracted elements during pattern matching and composition.
 2. Lack of possibility to add abstract general concepts as subject areas of the taxonomy.
 3. Lack of guidance for the pattern composition.
 4. Lack of conflict resolution, between patterns and between extracted elements.
 5. Lack of ways to assess relevance of elements in a way that conforms to human intuition.
 6. Lack of possibility to adjust level of abstraction.
 7. Lack of possibility to add additional background knowledge.
- Evaluation
 1. Lack of automatic and semi-automatic evaluations.
 2. Lack of semi-automatic refinement methods based on evaluation results.
 - Post-processing
 1. Lack of connection to manual approaches.
 2. More clearly define iteration possibilities of method.
 - User interface
 1. Not clear what is really helpful to the ontology engineer.
 2. No graphical user interface present.

Concerning problem analysis we noted that some way to focus the process was needed. Currently the ontology would cover everything mentioned in the input text documents. In the SEMCO experiment mentioned previously, this issue appeared very clearly since some of the documents used were not exactly within the right focus, although present in the texts, which generated a set of terms in the ontology that were not within the intended scope. There was then no way of excluding them from the ontology, other than manually intervening and removing them. A more reasonable approach would be to state the requirements of the intended ontology in some way, e.g. through a set of competency questions, and then match those to

the patterns in order to get an ontology specific for a certain application case and not representing the complete set of text documents. This issue is closely related to the issue of finding the 'right' input, since if the previously discussed method to focus the construction process is missing or very weak then guidelines for finding the correct input are instead the only way to focus the scope of the ontology. Nevertheless, even with a method to additionally specify requirements and focus, assistance needs to be provided regarding what input might be suitable and what is not suitable. The question arises how to find the input needed, even if it is determined what can be suitable.

The third and final issue regarding the problem analysis is a slightly more general issue, concerned also with patterns in general. In order to use a pattern-based method a pattern catalogue must be present. Patterns are probably not stable entities, but evolving and changing, new patterns emerge. Patterns can additionally be more or less domain dependent, as discussed in chapter 5. The issues noted in the SEMCO experiment was that the pattern catalogue was too small to really cover the complete scope intended, and that some of the patterns were not really suited for this domain. This indicates that it would be desirable to describe the patterns with some metadata in order to be able to do a first rough selection based on things such as domain of the pattern. Some ways of evaluating and evolving the patterns, and extending the pattern catalogues, are definitely needed.

Concerning the input processing issues a considerable amount of time in the SEMCO experiment was spent on converting documents into the correct input format. A major general shortcoming is also that the method so far only is concerned with text corpus input, many other kinds of input could be imagined. In the SEMCO case for example there existed several databases at the company in question, it would have been desirable to also be able to include for example data models or other more structured information, such as taxonomies and thesauri if present. When processing the input current OL systems only extract certain kinds of elements, but as discussed in previous chapters there is still need for more approaches concerned with for example concept formation, relation hierarchy extraction and general axioms. The ontology produced in the SEMCO case was light-weight, in terms of logical complexity, and this was the focus throughout this thesis, but in the future it would be desirable to also produce more complex ontologies.

The ontology composition step was the main focus of the initial method described in this chapter, whereby most of the analysis and open issues

are also found within this area. The first step of the process involved the matching between patterns and extracted elements, and as noted previously in this chapter only considering direct overlap is not enough to find all connections. In fact, the inherent difference in term abstraction level between patterns and extracted terms indicate that a method such as the one employed for the initial SEMCO experiment will only 'accidentally' identify a pattern, but in most cases appropriate patterns will not be found due to this abstraction gap.

The general structure of the ontology produced in the SEMCO experiment was not very easily comprehensible. For example, no intuitive top structure was present to make the ontology easier to understand by humans and to divide it into subject areas, neither was the ontology divided into any kind of modules or views. For the SEMCO experiment this was not crucial, but for a larger more complete enterprise ontology some structure to guide the pattern composition would definitely be needed, so that the ontology gets a more intuitive overall architecture. For the more detailed steps in pattern composition also some naive assumptions were made in the initial method, such as assuming that patterns will not have conflicting definitions and that parts of the pattern that were not matched are not relevant to include in the ontology. To detect and resolve conflicts between patterns is partly concerned with the addition of pattern metadata, as proposed above, but also to evaluation methods that can be used during construction in order to detect conflicts and problems when they arise. What to keep and what to discard when instantiating a pattern also needs more research. Since one of the intentions when using patterns is to include more background knowledge, and include such information that is not explicit in the text corpus, it is not desirable to naively discard everything that did not have a match. Additionally some information might need to be added, in order to for example adjust the level of abstraction of the different parts of the ontology, or to add additional information to improve the ontology.

Evaluation of the ontology should be provided as part of the framework. As seen in the SEMCO experiment evaluating the ontologies is not trivial and usually requires some adaptation of the evaluation methods, and evaluations are usually performed manually. Nevertheless, evaluation is a crucial step in ontology construction as mentioned previously, evaluation is for example needed already during the ontology composition process. Additionally, to be able to evaluate the initial ontology and provide suggestions for improvements, and to optionally iterate the complete process with additional input would be desirable scenarios. However, the process will never be

completely automatic, thereby a crucial issue is to really integrate methods like this with manual ontology construction methodologies and frameworks, and to eventually provide intuitive user interfaces for supporting the manual process.

Chapter 7

Semi-automatic pattern-based ontology construction

Based on the experiences gained in the SEMCO experiment described in chapter 6 and the general open issues discussed in chapter 2 a more complete framework for pattern-based ontology construction was developed. In this chapter the current version of OntoCase is described in detail. Examples and initial evaluations of some of the methods are also discussed, although the complete evaluation of the implemented parts of OntoCase is presented separately in chapter 8.

7.1 OntoCase overview

OntoCase is a general framework for iterative and experience-based semi-automatic ontology construction, based on the notion of ontology patterns. To incorporate the idea to learn from previous experience, and to assist in solving the pattern construction problem, the framework is inspired by the methodology of case-based reasoning (CBR). CBR proposes a cycle of learning from experience and using the learnt information to solve new problems. Intuitively this applies perfectly to the pattern-based ontology construction problem, since patterns are in fact encoded experiences, that are used to solve new problems. In Figure 7.1 an overview of the OntoCase approach is presented, including the tasks that are performed in each of the phases. Below we first discuss the inspiration from CBR and then the details of the OntoCase framework are discussed.

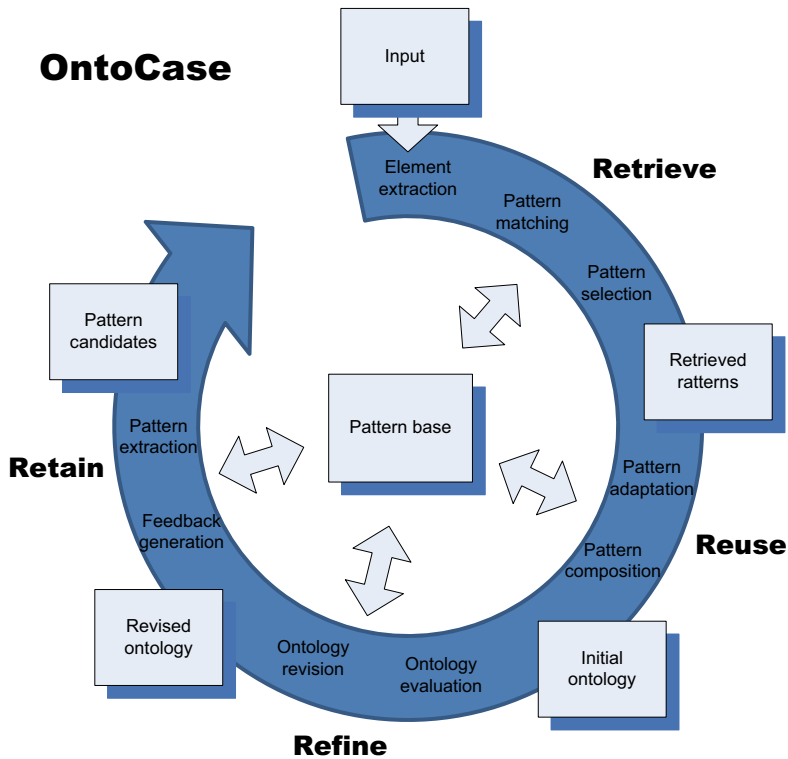


Figure 7.1: The OntoCase framework.

7.1.1 Semi-automatic ontology construction and CBR

A summary concerning some of the benefits and drawbacks of using case-based reasoning was presented in section 3.3. Some of the benefits listed were to reduced the load on knowledge acquisition tasks, learning from the past, reasoning with incomplete, imprecise or insufficient information, and reflecting human reasoning and means of explanation. Intuitively the reader can reflect on that these are also some of the general problems of ontology engineering. The guidelines were expressed as 5 questions, as presented in 3.3:

1. Does the domain have an underlying model?
2. Are there exceptions and novel cases?

3. Do cases recur?
4. Is there significant benefit in adapting past solutions?
5. Are relevant previous cases obtainable?

With starting point in these questions we can now motivate the suitability of CBR for pattern-based semi-automatic ontology construction. The first question concerns the domain, and ontology construction is certainly a very tough problem to model completely, not all underlying mechanisms are fully understood. When constructing an enterprise ontology the problem and the requirements can never be completely unambiguously defined, and whether the resulting ontology is completely correct and fulfils all those requirements is usually also a matter of human judgement. Currently there is no reasonable possibility to model the problem so that a complete and correct ontology can be generated completely automatically.

There are certainly exceptions and novel cases, depending on the problem at hand. No two enterprise ontologies will ever be exactly the same, even an enterprise ontology for the same enterprise with the same scope and intended use will be different if constructed at a different point in time. Ontologies are highly evolutionary structures that reflect the current situation. It is usually not possible to construct ontologies, except very simple and basic ones, that apply to several enterprises, because even though enterprises may look similar on the surface there are most often exceptions and differences present on the detailed level. However, some general principles apply for different kinds of ontologies. When discussing enterprise ontologies, including the organisation structure in the ontology will probably be very common. Even though the variability is almost infinite on the detailed level, some general principles apply for enterprises and thereby also for their ontologies. General principles of modelling also most certainly apply when modelling ontologies in logical languages.

The fourth question is harder to answer, is there really any benefit in adapting an old solution rather than constructing a new one? There are two main benefits that are generally proposed when considering knowledge reuse; to construct better models and to make the modelling easier and faster. This has unfortunately not yet been shown to hold empirically in the general case, although parts of an initial experiment supporting the conclusion that ontologies are of better quality when based on patterns were presented in section 5.3.4. Intuitively it seems plausible that at least the first proposition should hold also in a more general setting, by using a

'template' or an example solution, i.e. a pattern, an engineer, or a semi-automatic system, should be able to avoid making some common mistakes. Whether the process is really easier and faster is a harder question, since we are adding an additional structure as input to the problem, namely the patterns. It is likely that the patterns add some overhead, for example the time it takes to understand and correctly apply a pattern.

Finally, answering the fifth question is also not straight forward, are past cases really present when discussing enterprise ontologies? If complete ontologies are considered this is probably not true. Enterprises will not provide their ontologies and make them public, and to only use past cases from the enterprise in questions introduces a 'chicken and egg' problem of how to build the first ontologies of the enterprise. If instead smaller parts, such as ontology patterns, are considered this is however a reasonable assumption. Patterns do not give away the details of the enterprise, but encode valuable experiences and can be stored in catalogues for future reuse.

OntoCase and CBR

When comparing our initial ideas for OntoCase, as described in chapter 6, with the general idea of the CBR methodology, these are quite similar. The arrival of a new case would in terms of our initial method mean the arrival of a new text corpus with the intention of constructing an ontology. The problem can be expressed as finding the ontology that best represents the domain of the input text corpus, and in addition the problem could be further described by a set of optional competency questions or similar means of representing ontology requirements. The retrieval phase corresponds to extracting ontological elements from the text corpus, retrieving a set of possibly relevant patterns, i.e. the partial past cases, and evaluating that set for relevance. The final task in the retrieval phase is to choose what patterns to reuse.

The following phase of CBR is reuse, in our initial method this corresponds to reusing the patterns to propose an initial ontology. This phase contains the specialisation and adaptation of the patterns and the composition of the adapted patterns into an ontology. The revise phase of CBR corresponds to evaluation and revision of the ontology, which was not present in our initial method. The retain step is connected to the construction and refinement of the patterns. Pattern extraction from solutions is still future work and in the initial method only manual construction of patterns was considered.

7.1.2 The OntoCase framework

One of the core elements of a CBR approach is the case base and its content. Figure 7.1 above presented an overview of OntoCase and its tasks, additionally Figure 7.2 shows a tentative instantiation of the OntoCase cycle. The general framework is presented at the centre of the figure and an example of how these phases can be realised in actual tasks with input and output data is surrounding the core framework. In the OntoCase approach, illustrated at the centre of Figure 7.2, the case base corresponds to a pattern catalogue, denoted pattern base, containing both ontology design patterns and tentatively architecture patterns, although no such patterns were constructed yet. Patterns are here seen in a broad sense, as recurring reusable modules, but not necessarily proposed and certified by any 'pattern authority', as discussed in chapter 5. The patterns on the design level are constructed for automatic use and are therefore small self-contained ontologies described in some ontology representation language, examples presented in chapter 5 and appendix B.

The first OntoCase phase corresponds to case retrieval and constitutes the process of analysing the input, i.e. the text corpus, extracting ontological elements and matching them to the pattern base, and then selecting appropriate patterns. The second phase, case reuse, constitutes the process of reusing retrieved patterns and constructing a first version of the ontology, by adapting, i.e. specialising, and composing the patterns. The third phase concerns revision of the ontology, improving the fit to the intended scope and task and improving the ontology quality. The final phase includes the discovery of new patterns as well as storing pattern feedback.

In Figure 7.2 the OntoCase process is described at the centre of the figure, and surrounding it an example instantiation of the complete process is illustrated. The process starts from the top of the figure, with the input provided by the user, and proceeds clockwise in the illustration. Certainly the process is not limited to being applied in a linear fashion, the intention is to be able to apply each phase independently, or iterate the complete process if necessary, depending on available input data. Each phase in itself may also contain iterations, but for the sake of simplicity we will here describe the process step by step.

In the OntoCase approach there is an uncertainty inherent in all the described steps. Each primitive found in the analysis of the input, mainly terms and relations, can have a certain degree of confidence associated with it, and so can the pattern primitives. The pattern confidences are based on

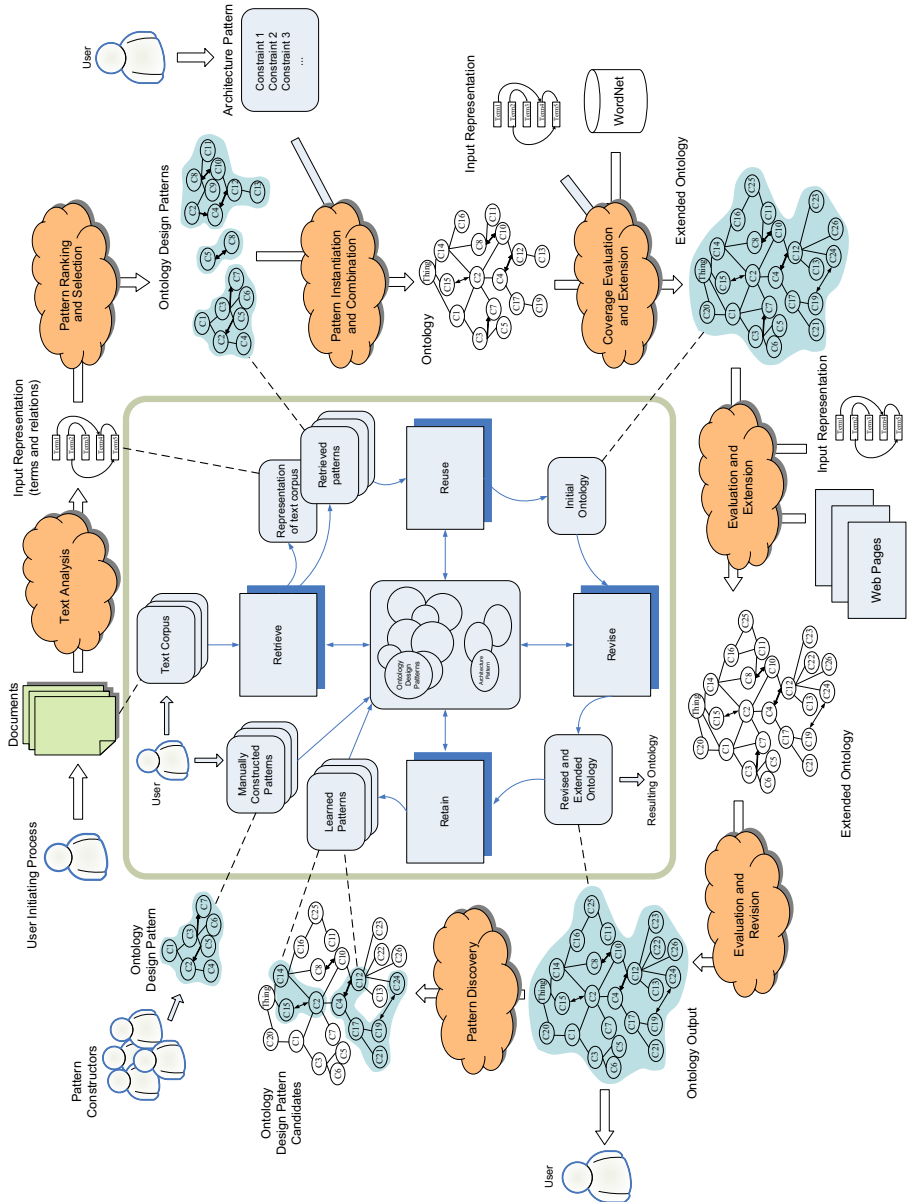


Figure 7.2: The OntoCase approach.

a set of factors, where one factor is the 'nature' of the pattern, i.e. whether it is a consensual ontology design pattern that was manually constructed representing some best practice or if it is a pattern retained from one or several solutions, i.e. a pattern candidate. The pattern primitives match extracted elements only to a certain extent, and the levels of confidence are transferred onto the elements of the constructed ontology when it is built. To store the ontology using a standard ontology representation formalism, such as OWL, thresholds can be set for acceptable confidence levels or the ontology primitives can be validated by a user.

When comparing the complete OntoCase cycle to the initial approach used in the experiment described in chapter 6, the experiment covered two of the four phases. The analysis of the input text corpus and selection of patterns were in the experiment realised using existing text processing and string matching algorithms, and the combination used a naïve implementation mainly transforming input and output into correct formats. The reuse phase, composed patterns based on heuristics and used the extracted elements only for tailoring labels and synonyms. Compared to the conducted experiment the phases have been considerably improved in the current OntoCase approach. The description of the retrieval and reuse phase below shows actual solutions together with future work, while the discussion of the revise and retain phases describes theoretical views and future work, since OntoCase as a whole is still ongoing research. However, first we provide a more detailed discussion about the pattern base for storing design and architecture patterns.

7.1.3 Pattern base

The pattern base would tentatively contain two distinct kinds of patterns, patterns on the design and on the architecture level. Among the patterns on design level the focus is on design patterns representing solutions to restricted partial problems, i.e. solutions to modelling problems providing small pieces of the complete solution. These are represented as small ontologies, as described in chapter 5. Intuitively, ontology design patterns need a certain level of abstraction to be reusable in several cases, just as patterns in other areas. However, ontology design patterns for automatic use need to be specific enough to be matched against the extracted elements and be suitable for providing missing general domain knowledge not explicit in the input text corpus, and they need to be formally represented.

All the patterns on design level are thereby small ontologies, with the



Figure 7.3: A pattern covering communication event concepts.

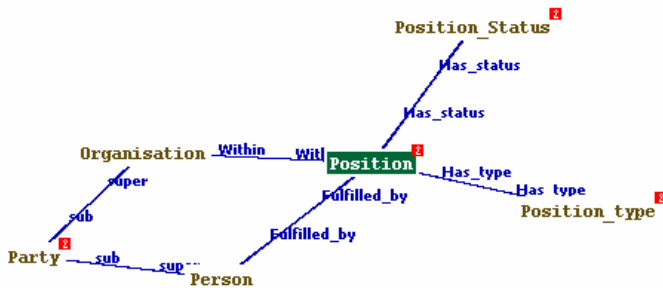


Figure 7.4: A pattern covering concepts related to positions.

restriction that the graph representation of the pattern must be connected. At this point we do not restrict our approach to only 'certified' or commonly patterns, but also accept any kind of proposed pattern candidates. The quality of patterns certainly affect the quality of the output ontologies, but with the scarce pattern catalogues existing today we have to lower the standard in order to have a sufficient number of patterns to start with. Examples of slightly simplified versions of two patterns intended for enterprise application ontology construction that were used in the initial construction experiment described in chapter 6 are illustrated in Figure 7.3 and 7.4. The positions pattern clearly qualify as an abstract ontology design pattern, the communication event pattern is much more domain specific, and might be treated as a recurring solution rather than an ontology design pattern, since consensual acceptance and validation of the pattern by the community is at this point uncertain.

The first pattern, illustrated in Figure 7.3, contains concepts connected to communication within and between companies. A communication event

has a certain purpose in the organisation and a specific role for a specific party involved in the communication. This is an example of a relatively detailed pattern, encoding domain knowledge of the business domain, but not necessarily any specific business domain. Both patterns presented as examples originate in data model patterns for enterprise database construction, but were slightly modified, replacing relational model specifics with ontology modelling best practices as described in chapter 5. The transformation was made manually by an ontology engineer. The second pattern example, illustrated in Figure 7.4, represents a more abstract pattern, dealing with positions within an organisation and their fulfilment by specific persons. This is a simplified version of a pattern similarly inspired by data model patterns.

In the OntoCase method, ontology design patterns can be constructed manually although candidate patterns can also be discovered from produced solutions. The manual pattern construction process involves reaching a consensus and encoding best practices in ontology modelling. The OntoCase method does not focus on this manual process, but assumes the existence of a pattern base. In case there is a lack of best practices, or consensus, also the possibility of adapting modelling patterns from other areas, such as database construction and problem solving methods, could be used as previously described in chapter 5. In addition to the ontology design patterns in the pattern base, possibly useful variants and specialisations of these design patterns can be stored. In this way the distinction between mined patterns, as reoccurring solutions retained from constructed ontologies, and consensual ontology design patterns will not be strict in the OntoCase approach, as a matter of fact in the implemented automatic construction process no distinction is made at all. The automatic discovery of pattern candidates from solutions is still future work, as described later.

The notion of architecture patterns would tentatively be used to ensure an appropriate overall structure of the constructed ontology, and to impose constraints on the ontology construction process. In an abstract sense, an ontology architecture pattern can, for example, describe a division of the ontology into modules, layers, and subject areas. Such a general structure could utilise a top level enterprise ontology similar to what is described by Uschold et al. [207]. In the OntoCase approach an architecture pattern is represented as a set of constraints, both used when composing the ontology from the selected ontology content design patterns and also as a specification or restriction when searching for patterns in the pattern base. So far no architecture patterns have been constructed in practice, mainly due to the

lack of available enterprise ontologies, but the theoretical notion of architecture patterns will still be considered as part of the OntoCase framework. More specifically we envision that future versions of the implemented OntoCase framework will exploit *ontology reference architectures* for guiding the ontology composition process in the reuse phase.

The pattern base is currently realised in the form of a database, only considering the storage of content design patterns in the form of OWL building blocks. The pattern base contains metadata of each pattern, pointers to the actual pattern, and connections between patterns, tentatively including connections between possible architecture patterns and content design patterns. The metadata is part of the pattern base index and is currently based on the labels of the core concepts of each pattern, a set of manually entered keywords (optional), the domain of the pattern (optional), and the name of the pattern. Connections to other patterns can currently be of two kinds, either the pattern is a 'variant of' another pattern or it is 'related to' another pattern. The variant relation indicates changed patterns, i.e. patterns with a significant overlap, and can also be used to connect the consensual design patterns to patterns that are retained solutions. When patterns are retrieved for matching and ranking, based on a database query, also variants and related patterns will be retrieved.

7.2 Retrieval

This section discusses the detailed methods of the retrieval phase, in its current version.

7.2.1 Text processing for ontology learning

A small evaluation was performed comparing one of the recent OL systems freely available, to more basic text analysis techniques such as tokenisation, morphological normalisation, and stop word removal. This was done to exemplify why pattern-based methods, e.g. OntoCase, need to rely on recent OL systems, rather than standard text processing tools. For this example the collection of concept extraction algorithms in the research prototype called Text2Onto, as described by Cimiano [34], was chosen. The choice was based on that Text2Onto incorporates a large number of different algorithms, it is freely available for download, and it is considered by the community as one of the more elaborate systems existing today.

The example is not intended as a quality measure of the particular approach, or approaches such as the one proposed by Cimiano [34] in general, it only aims to point out that certain features such as multi-word term extraction, through algorithms as proposed by Frantzi et al. [71] also used in Text2Onto, are needed for OL, and that multi-strategy approaches seem to give more useful results than classical text processing. This is also our motivation why we build on existing OL systems, even though they are mostly research prototypes, for element extraction instead of using standard text processing components.

In the example a small text corpus was first manually analysed by an ontology engineer, who picked the terms from the text he most likely would use as labels of concepts, and label synonyms, when building an ontology to represent the texts. This manually constructed term list was used as a 'gold standard', when comparing the result of Text2Onto to different combinations of basic techniques, i.e. tokenisation, weak stemming, and stop-word removal, in a standard implementation, i.e. the GATE ANNIE framework, as described by Cunningham et al. [42].

Some example results are shown in Table 7.1. The experiment covered several additional combinations, but the ones presented below are sufficient to illustrate the general issues. The low level of recall using standard components, see experiment 1-3, is mainly due to the fact that the classic techniques do not consider multi-word terms, while this can be done through more elaborate algorithms in tools like Text2Onto. It is also obvious that the OL approach, see experiment 4, performs a much more effective and tailored filtering to also increase the precision. This is mainly done through specific term relevance measures and lexico-syntactic patterns applied on the parsed text.

In general, existing OL approaches do not only transform terms and linguistic annotations, determined through classic natural language processing techniques, to an ontology representation, but additionally use algorithms specifically tailored towards discovering ontology primitives and filtering out irrelevant information. A problem with using existing OL systems, many of which are still experimental prototypes, as a basis for OntoCase could be to introduce a bias, and possibly filter out relevant information. Although our small experiment indicates that the extracted primitives of the OL system better agrees with human intuition, the decisions of a human ontology engineer, than simply presenting every word from a text, we recognise this risk since the recall is still below 50% compared to manual extraction. However, using existing OL algorithms clearly improve a lot on using classical

Table 7.1: Precision and recall of text analysis for concept discovery.

Experiment	Method	Precision	Recall
1	ANNIE tokenisation	14%	33%
2	ANNIE token. + weak stemming	16%	37%
3	ANNIE token. + weak stemming + stop words rem.	18%	37%
4	Text2Onto	52%	48%

text analysis techniques, which is the only alternative in terms of automatic processing, for our specific case. As we shall see later we have used the Text2Onto approach for performing the text analysis, when an existing initial ontology is not already available.

7.2.2 Retrieval steps

Below the steps of the retrieval phase are described in detail.

Element extraction

The first step of the retrieval phase, as illustrated in Figure 7.2 previously, involves extracting a representation of the input text corpus, i.e. extract ontological elements from the texts. The input could in theory be of different kinds, such as HTML pages or database schemas, but so far we assume plain text files. The extraction involves analysing the input text corpus to extract as much 'evidence' relevant to ontological primitives as possible. This analysis is the main focus of most existing OL approaches, as explained previously, and studied through the experiment in the last section. Therefore developing new solutions for this is not the focus of our research, merely to select existing approaches that fit to the OntoCase approach.

As an alternative to this step OntoCase can also start from an existing ontology, instead of extracting the elements from a text corpus. This initial ontology might have been extracted by some other OL tool, it might be the output of OntoCase itself if used iteratively, or it may be any 'seed ontology' that the user provides. Whether the elements are extracted from a text corpus or present in an ontology, OntoCase stores them as an ontology representing the input, i.e. the input representation in terms of CBR. The extracted elements are assumed to be associated with normalised confidence values, representing the confidence with which the elements are believed to be correctly extracted and appropriate for the ontology. If the input is

instead an ontology, as mentioned above, its elements can also be associated with confidence values, e.g. if the ontology is the result of a previous run of OntoCase, but if not, the further application of the OntoCase method can be performed also without such values, although the result will be more uncertain.

Pattern matching, ranking and selection

The second step of the retrieval phase involves comparing the input representation, i.e the lists of terms and relations or the seed ontology, with associated confidence values, if present, to the pattern base. When viewing the matching on an abstract level it is similar to ontology matching, with the addition that both 'ontologies', the input representation and the pattern, are uncertain, as explained above. On a more detailed level there are also some specific characteristics of the ontologies being matched, i.e. the patterns are relatively small, well structured, and abstract, while the input representation can be large, and is usually quite diverse and sparsely connected. The matching results is a special kind of ontology alignment, used in the reuse phase for composing the ontology. It is also used in this step, as a set of correspondences, each associated with a type and a value, between primitives of the input representation and the patterns, for computing the total ranking value. The matching method developed is heavily inspired by recent ontology ranking schemes, such as the one described by Alani and Brewster [6], although our approach draws on the richer structure of the input extracted from the text corpus.

We propose a ranking scheme based on four different factors; concept coverage, relation coverage, density and semantic proximity of matched concepts in the pattern. Each pattern is evaluated against these factors and a single value representing the rank of the pattern can be computed. The pattern selection is made based on choosing patterns in the ranking order and computing the resulting total coverage over the input representation. Patterns are chosen until a threshold is reached, either a threshold on the ranking value itself, or a sufficient coverage has been obtained, or the increase in coverage is no longer significant by selecting more patterns.

To illustrate this process through an example we use the *positions* pattern presented in Figure 7.4 and a pattern describing *work effort* and its connection to the realisation of requirements, as illustrated in Figure 7.5 below. The example input representation is based on a set of randomly selected short passages extracted from the text corpus used in the initial

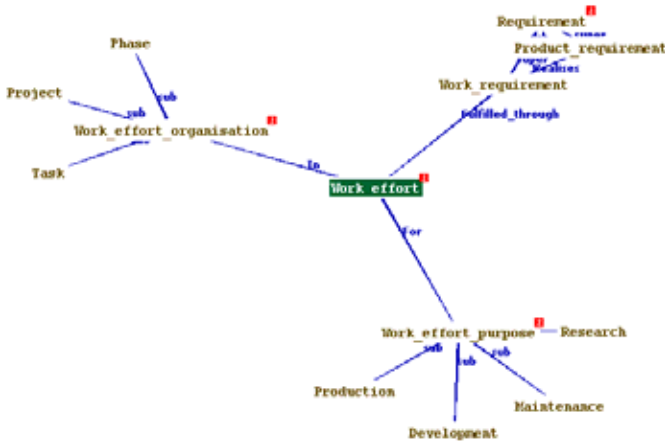


Figure 7.5: A pattern covering work efforts and requirements.

meeting (0.81)	team (0.61)	project (0.5)
overview (0.69)	work (0.61)	requirement (0.5)
sw verification (0.69)	integration test (0.56)	software (0.5)
document (0.61)	plan (0.56)	sw (0.5)
engineer (0.61)	project management (0.56)	sw development (0.5)
engineering (0.61)	project manager (0.56)	sw project (0.5)
information (0.61)	design (0.5)	testing (0.5)
phase (0.61)	designer (0.5)	
product (0.61)	developer (0.5)	
production (0.61)	development (0.5)	
sw project management (0.61)	management (0.5)	
sw project manager (0.61)	manager (0.5)	
system (0.61)	process (0.5)	

Figure 7.6: List of extracted terms with associated confidence values.

experiment of Section 6.2.3, i.e. project plans and process descriptions of a company in the automotive industry. The details of the term extraction are not relevant here, but it results in a term list of 33 terms as shown in Figure 7.6, which will be used throughout this example.

For this example we only compute the concept coverage and use this as the pattern rank. Concept coverage is based on a direct and an indirect part, the direct part being term and label similarity, determined through string matching, and the indirect part being subsumption coverage, determined for example through the WordNet dictionary and a head heuristic. We apply the Jaccard string similarity measure, details explained by for example Cohen et al. [39], with a threshold of 0.5 on the term list and the two

Table 7.2: String matching scores.

Pattern concept label	Extracted term	Similarity
requirement	requirement	1.0
project	project	1.0
phase	phase	1.0
development	development	1.0
production	production	1.0
product requirement	product	0.5
product requirement	requirement	0.5
work requirement	work	0.5
work requirement	requirement	0.5
work effort	work	0.5
project	project management	0.5
project	project manager	0.5
project	sw project	0.5
development	sw development	0.5

patterns. The concept labels of the *positions* pattern in Figure 7.4 have no matching terms in the term list. On the other hand, the *work effort* pattern from Figure 7.5 displays several matches, see Table 7.2.

Indirect concept coverage measures are applied to discover, with assistance of WordNet and heuristics as mentioned above, that for example the terms *designer*, *engineer*, *developer* and *manager* are concerned with people, thereby indicating that the concept *person* in the *positions* pattern can 'indirectly' cover these terms. Similarly, some terms can be found that are possibly covered by the concept of *organisation*. In total this gives the *positions* pattern the aggregated concept coverage value of 0.27. For the *work effort* pattern also some indirectly covered terms can be found and together with the direct coverage, the string matching results, this results in a value of 0.35. In these numbers the uncertainty of the extracted terms is incorporated, as well as the uncertainty of the matches and the fraction of the pattern that is covered. In the complete method the ranking scheme would proceed to calculate the relation coverage, the density, and the proximity of the matched concepts, before aggregating these into a rank value.

The important benefits of this approach is the ability to rank abstract patterns, such as the *positions* pattern with no directly overlapping terms, since the ranking is not only based on simple string matching. The complete ranking algorithm takes into account extracted and matched relations, and the utility of the patterns in terms of their structure in relation to the

matched concepts. For pattern selection the total coverage of the pattern(s) over the input representation should be computed. Selection can then be made based on what the user defines as sufficient coverage, as well as the coverage gain in selecting more patterns. The details of the ranking scheme are presented later in section 7.5.

7.3 Reuse

The reuse phase is concerned with instantiating, i.e. adapting and specialising, and combining the patterns into an initial solution ontology. The combination process tentatively uses the architecture pattern chosen by the user, if present, to guide the composition, while iterating over all selected patterns. The default rule is to only include those parts of a pattern that had some match in the input representation. This includes selecting only matched concepts, selecting appropriate labels, and their synonyms, as well as selecting relations. The process is enhanced by a set of heuristics, intended to create a more well-structured ontology.

In the example presented in the last section, for the *positions* pattern the default rule would imply to only include the concepts *person* and *organisation*, which covered some input primitives, but additionally a heuristic can be used to include the superconcept *party* if it is desirable to keep the instantiated pattern 'connected'. Another such heuristic is to use the transitive property of taxonomic relations, including more taxonomic relations to keep the ontology connected even if not all intermediate concepts were matched and included. The heuristics currently proposed include the heuristics of the initial method, as presented in section 6.2, and two additional heuristic that were added in the revision of OntoCase:

1. Include all relations between added concepts/terms, even if they were not matched.
2. Use the transitive property of hierarchical taxonomic relations, if an intermediate concept is missing add the child directly at the level of the missing concept.
3. Add all extracted subconcepts (terms) of concepts (terms) that were matched to a pattern concept and included.
4. An associative relation which originally relates two concepts (or terms) is added even if one of the concepts (or terms) is missing, if and only

if there is a direct subconcept of the missing concept (or term) present in the ontology.

5. Add all superconcepts of pattern concepts that were matched to extracted elements.

The first heuristic is included since relations are a lot harder to extract from a text corpus than terms, hence there will be a number of relations missing. If two pattern concepts are matched to extracted terms, and the concepts are connected through a relation in the pattern, which is not matched, the relation can still be included in order to ensure that the pattern structure is preserved, as illustrated in Figure 7.7.

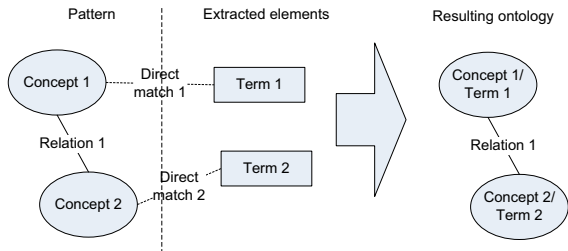


Figure 7.7: Heuristic for adding unmatched relations.

The second and fourth heuristics are included to preserve the structure even if there are 'gaps' in the matched parts. Gaps in the taxonomy are closed by using the transitive property of the subclass relation. Adding relations that could be added based on the first heuristic even if a concept in the taxonomy is missing, is the focus of the fourth heuristic. Examples of the heuristics are illustrated in Figure 7.8 and in Figure 7.9 respectively.

The third and fifth heuristics are included to preserve taxonomies present in the extracted elements and in the patterns. From patterns we pick the general top structure, and from the extracted elements we pick the specific taxonomies of terms extracted from texts. Examples of the heuristics are illustrated in Figure 7.10 and in Figure 7.11 respectively.

Pattern composition is another task during the ontology building of the reuse phase. During pattern composition, pattern overlap needs to be resolved. For some patterns explicit 'variant of' relations exist in the pattern base, then these can be used to assume an overlap between patterns. Overlap can additionally be handled using heuristics, for example assuming that two concepts represent the same concept if they have the same set of

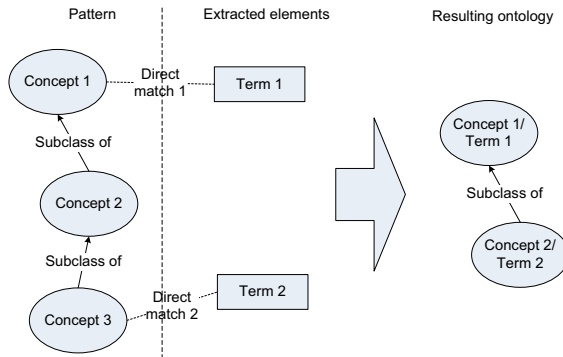


Figure 7.8: Heuristic for taxonomic relations.

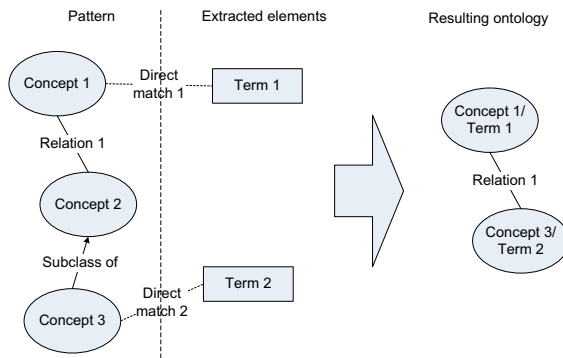


Figure 7.9: Heuristic for downward propagation of relations.

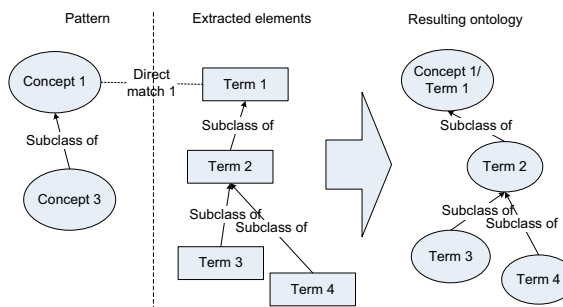


Figure 7.10: Heuristic for preserving extracted structure.

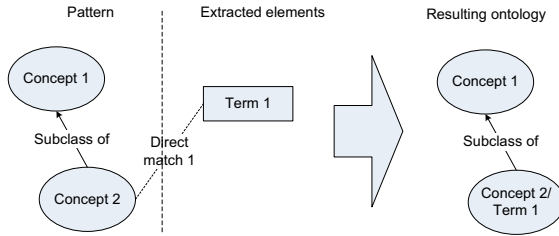


Figure 7.11: Heuristic for preserving pattern top levels.

synonyms and no conflicting relations, or other axioms. This is an initial approach, also used for the first version of OntoCase, that needs further refinement in future work, and should conform to more advanced methods of ontology integration and merging. However, this has not been further developed in this version of the OntoCase method. The confidence values of the pattern primitives and input representation primitives are used together with the matching values to compute new confidence values for the primitives of the resulting ontology.

7.4 Future work - revise and retain

For the revision phase, one objective is to compensate the missing background information of input texts, i.e. to really cover the complete domain intended. Some missing parts have already been added with the help of design patterns, but still we can see from the initial experiment that this will probably not be enough. In this case we have to use external sources of information to try and attach extracted primitives to the ontology, and extend it, in a structured way. Our idea for improvement involves using selected (focused) parts of the web to gather more general information. In addition a second step of the revision phase will focus on reduction of redundancy and resolving inconsistencies in the ontology. This step would be based on user intervention, hence an appropriate graphical user interface is essential.

The final phase, retaining patterns, is mainly inspired by approaches to ontology modularisation and algorithms for finding strongly connected graph components. Finding coherent parts of the ontology that might constitute suggestions for new patterns can be done by traversing the taxonomy, and through heuristics the candidates can be restricted in their size

and structure. We envision some user involvement in this step, validating and possibly generalising the candidates before inclusion in the pattern base. The feedback process for existing patterns, involves the recalculation of confidence values present in the applied patterns, as well as supporting the user in making pattern changes or updates. Purely manual pattern construction methods are considered outside the scope of OntoCase, but manually constructed patterns can be stored and used in the approach.

7.5 Pattern ranking

The problem of pattern ranking focuses on describing how well a pattern fits to the input representation. The patterns are in general more abstract than the primitives found in the input hence a straight forward matching is not always possible, in contrast to the selection approaches that exist. Instead the matching must be viewed as a process of gathering clues as to whether a pattern might be suitable or not. In this thesis we focus on a ranking scheme that utilises a multi-strategy approach.

First, matching of terms against concept labels is applied, similarly to the approaches existing for ontology search. Inexact matching is used but still only a few similarities will be found, mainly due to the abstraction level of patterns. Additionally an indirect matching is introduced, 'abstracting' from terms to concepts. Also, the extracted relations will be matched against the pattern relations. Term to relation label matching is applied in some related approaches, but in our case we additionally utilise the fact that we have already distinguished between terms representing possible concepts and terms representing relation labels in the input representation. As a third step quality measures similar to the ones used for ontology search are applied, to give clues on the usefulness of the pattern, but the measures are selected and adapted based on the specific characteristics of ontology patterns.

The ranking scheme is inspired by recent ontology ranking and selection approaches as mentioned above, but also has some major differences. First, the input to an ontology search engine is commonly assumed to be a small set of keywords. In our case the input resembles a diverse ontology, in the form of a term list and possible relations between those terms. This richer structure should be utilised, hence the addition of a relation matching part which is not present in any of the related approaches for ontology search. Since patterns are constructed to be reusable components they can be as-

sumed to be more abstract than most terms used in a text. This assumption leads to the additional introduction of two subsumption coverage measures. The task is also slightly different from that of ontology search and selection in that the input in our case is intended to restrict the scope of the ontology, while a keyword search usually only hints to some general topical concepts. It follows that the common approach in ontology search engines is to study how well *one single ontology* covers the query, but in our case the task is to study how well a pattern fits the input in order to find the best *set of patterns*.

When compared to ontology matching, we can also find similarities. We use many of the basic techniques also used in ontology matching. However, there are some major difference, since we are already beforehand aware of the abstraction gap between the extracted elements and the patterns. Patterns and extracted elements rarely have a large overlap, if any. It is also hard to rely on structural matching techniques since the structure around the extracted elements is often quite sparse.

7.5.1 Concept coverage

To determine the direct coverage of extracted terms over the concepts of a pattern, string matching of concept labels is used. The direct coverage is computed based on the fraction of the pattern concepts c_i that matches a term t_i in the extracted term list T . Each extracted term t_i is associated with a confidence value $conf(t_i)$ as mentioned previously. The string matching between terms and concept labels produces a similarity value $sim(t_i, c_j)$ representing the degree of similarity between two strings, any common normalised string matching measure could be used. These values are then composed into a weighted matching value for each discovered partial match, $match(t_i, c_j) = conf(t_i) \cdot sim(t_i, c_j)$. The intuition is to give higher relevance to matches involving terms with a higher extraction confidence. The confidence values and string matching scores are normalised, so the matching score ($match()$) will also assume values between 0 and 1. For each concept the direct coverage ($dc()$), the maximum weighted matching value, is computed as:

$$dc(c_i) = \max_{t \in T} (match(t, c_i)) \quad (7.1)$$

Since most extracted terms are quite specific, there is a need for additional ways to find clues of connections between a pattern and extracted terms. We have chosen to initially focus on possible subsumption relations,

i.e. hypernym relations between extracted terms and concept labels. Two approaches are used, looking for connecting hypernym chains in WordNet and subsequently using the 'head heuristic' explained previously. The intuition of subsumption coverage is that if a term is subsumed by a concept of a pattern, then that is an additional clue that the pattern fits to the input.

The dictionary approach starts with a term $t_i \in T$. This term is matched against the WordNet dictionary, through exact string matching, its corresponding WordNet term, if any, denoted by $wn(t_i)$. The matching of a pattern concept c_j to the WordNet dictionary can be done 'offline' and the match, denoted $wn(c_j)$, stored together with the pattern. At runtime the hypernyms of each $wn(t_i)$ are searched for some $wn(c_j)$ of a concept in the currently evaluated pattern p . Since $wn(t_i)$ can have several senses, $sen(wn(t_i))$ denoting the number of senses, there can be several paths from $wn(t_i)$, not all leading to $wn(c_j)$ matching the concept c_j . If one or more hypernym chains are found between the terms, as illustrated in Figure 7.12, the number of chains connecting $wn(c_j)$ and $wn(t_i)$ is denoted by $|paths(t_i, c_j)|$ and the shortest chain by $path_{min}(t_i, c_j)$. The subsumption coverage ($sub()$) of the concept $c_j \in p$ can be computed as:

$$sub(c_j) = \sum_{t \in T} \frac{|paths(t, c_j)|}{length(path_{min}(t, c_j)) \cdot sen(wn(t))} \cdot conf(t) \quad (7.2)$$

To directly find some related concept c_j in the pattern, for a multiword term t_i , the so called 'head heuristic' is applied. For each match the number of modifiers, additional words preceding the pattern term, denoted by $mod(t_i, c_j)$, are treated analogously to a step in the hypernym chains above. Since we have no information about senses of the terms, this is disregarded. The formula then takes on this simpler form, for computing the head heuristic relation coverage ($vr()$) of a concept c_j :

$$vr(c_j) = \sum_{t \in T} \frac{1}{mod(t, c_j)} \cdot conf(t) \quad (7.3)$$

The coverage of each concept is computed as the sum of the three scores, for each concept of the pattern, with a maximum score for each concept set to 1. The total number of concepts of the current pattern p is n and the number of concepts that had any match is m . Then the total concept coverage ($CC()$) of the term list T can be computed as shown below.

$$cover_c(c_k) = \min((dc(c_k) + sub(c_k) + vr(c_k)), 1) \quad (7.4)$$

$$CC(p, T) = \frac{1}{n} \sum_{k=1}^m cover_c(c_k) \quad c_k \in p \quad (7.5)$$

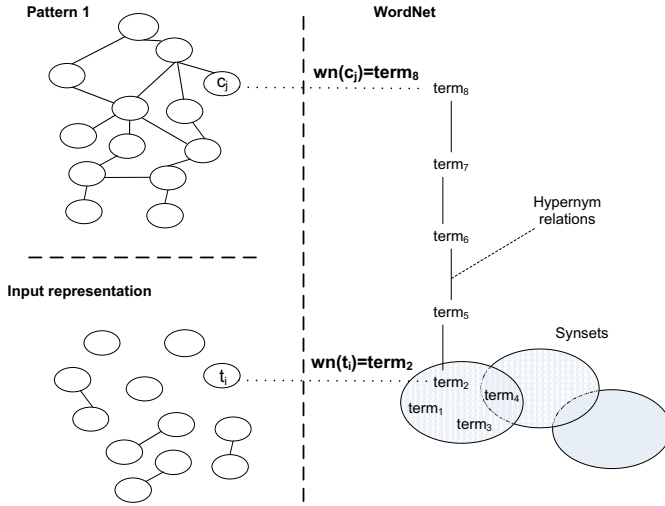


Figure 7.12: The use of WordNet hypernym chains.

7.5.2 Relation coverage

Relation to relation matching can be done both based on relation labels and on the term to concept matches found in the direct concept coverage calculation. If t_1 is considered to match c_1 and t_2 is considered to match c_2 , then a relation between t_1 and t_2 might match a relation between c_1 and c_2 with some degree of confidence. It is not certain that it is the same relation, if there are no other clues, but it can be added as a hypothesis with a certain level of confidence. If the extracted relation is a named relation then the label is also matched, using the same kind of string matching as for the term to concept case described previously.

For each relation pr_i from concept c_d to concept c_r in the current pattern p , the best match (if any) is selected from the extracted relations, as illustrated in Figure 7.13, r_j from term t_d to t_r . The fitness score of a match is calculated based on the individual matching scores of the direct concept matches ($dc()$) and the extraction confidence of r_j , denoted $conf(r_j)$. When matching relation labels we let $sim(pr_i, r_j)$ denote the string similarity between the relation labels. The value of the fitness score of the pattern relation pr_i to any extracted relation $r_j \in R$, where R is the set of all extracted relations, can be computed as $rel(pr_i, r_j) = \frac{1}{2}(match(t_d, c_d) \cdot match(t_r, c_r) \cdot conf(r_j) + sim(pr_i, r_j))$.

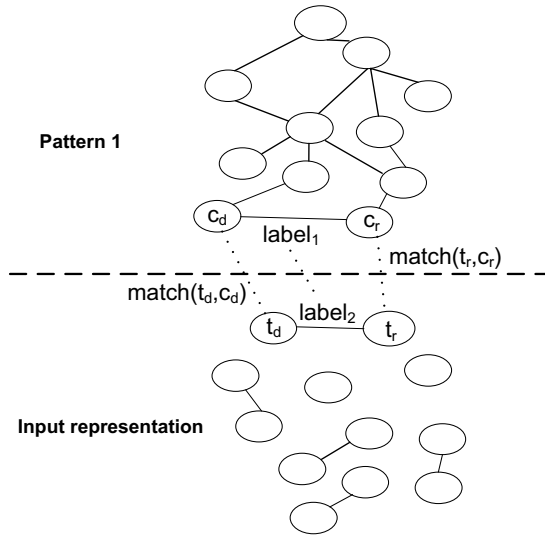


Figure 7.13: The relation matching.

The total relation coverage of one individual pattern relation is:

$$cover_r(pr_i) = \max_{r \in R} rel(pr_i, r) \quad (7.6)$$

and the pattern coverage is then computed from these individual matches. The total number of relations in pattern p is denoted n and the number of matched relations is denoted m . The total relation coverage is:

$$RC(p, R) = \frac{1}{n} \sum_{k=1}^m cover_r(pr_k) \quad pr_k \in p \quad (7.7)$$

7.5.3 Utility measures

When evaluating ontology fitness, measures like centrality, density and semantic similarity are commonly used. These mainly aim to assess the structure of a larger ontology but with some modifications similar measures can also be used for pattern ranking. Among the measures mentioned we choose to use the notion of density and a variant of semantic similarity that we call proximity. Measures such as structure and connectedness, as proposed by [28], only consider the ontology (or pattern) as a whole and are thereby too crude to give suitable clues on the usefulness of a pattern. Centrality is not

used since patterns are small, usually at most one or two levels of taxonomic specialisation and in such a setting centrality does not make sense.

The density measure chosen is a simplified version of the measure proposed by Alani and Brewster [6], considering taxonomically related concepts, taxonomical siblings, and concepts directly related through general relations. The reason for not including indirect relations is pure simplification, and instances are left out because we are dealing with patterns that are not expected to contain any instances. Both for this measure and the proximity, all relations are considered disregarding any 'direction' of the relation. Hence, a relations from a to b is counted as a relation for both a and b .

The number of all concepts of a pattern p is denoted by n . The number of concepts that had any match in the concept coverage evaluation previously is denoted m , below we assume that c_i belongs to this set. The number of concepts directly connected to c_i through taxonomic relations is denoted $tax(c_i)$, and the number of siblings is denoted $sib(c_i)$. The number of concepts directly connected to c_i through general relations, non-taxonomic relations, is denoted $re(c_i)$. Combined, we compute for each concept:

$$den(c_i) = \min\left(\frac{tax(c_i) + sib(c_i) + re(c_i)}{n}, 1\right) \quad (7.8)$$

For each concept considered, a c_i in the set C containing m concepts, we finally weight the density value with the coverage value $cover_c(c_i)$ it received when matched in the concept coverage stage. The sum of all weights $\alpha = \sum_{j=1}^m cover_c(c_j)$ is used for normalisation. The complete density of pattern p is expressed as:

$$DE(p, C) = \frac{1}{\alpha} \sum_{j=1}^m cover_c(c_j) \cdot den(c_j) \quad (7.9)$$

The proximity measure considers those concepts that had some match through concept coverage, the set C above. The distance $dist(c_i, c_j)$ between two concepts c_i and c_j in a pattern is computed as the length of the shortest path between the concepts, taking into account all relations, except paths passing an 'artificial' root node, present in some ontology languages. The length of a path from a concept to itself is 0. The maximum distance (as defined above) between any two concepts in p , matched or not, is denoted $diam(p)$. The proximity value of c_i and c_j can then be expressed as $prox(c_i, c_j) = 1 - \frac{dist(c_i, c_j)}{diam(p)} \cdot (1 - \frac{m}{n})$ if $c_i \neq c_j$ and $prox(c_i, c_j) = 0$ if no connecting path exists. The total proximity value of pattern p is normalised

through the sum $\beta = \sum_{k=1}^m (m - k)$ and computed as:

$$PR(p, C) = \frac{1}{\beta} \sum_{i=1}^{n-1} \sum_{j=i+1}^n prox(c_i, c_j) \quad (7.10)$$

7.5.4 Rank calculation

In order to make the ranks of the specific patterns comparable between different instantiations of the process all normalisations are based on each pattern itself, for example on the pattern size. This gives the benefit of comparability, since a pattern will get the same rank even if more patterns are added, as long as the input is the same.

The four measures need finally to be aggregated into one value. Although more advanced combinations could be imaginable, for simplicity reasons a linear combination is used with equal weight on all four measures. Let $score(p) = \{v_1, v_2, v_3, v_4\}$ where $v_1 = CC(p, T)$, $v_2 = RC(p, R)$, $v_3 = DE(p, C)$ and $v_4 = PR(p, C)$. The combined ranking value of a pattern p can be expressed as:

$$Rank(p) = \frac{1}{4} \sum_{i=1}^4 score(p)[i] \quad (7.11)$$

7.5.5 Pattern selection

The simplest approach for selection is to let the user set a threshold on the ranking value, but this is not very flexible. A more elaborate approach is to study the total coverage of the patterns over the input representation in order to dynamically set the selection threshold. A third option is to additionally study the increase in coverage that each step in the ranking order produces. Initially we have used a combination of the first two approaches, setting a threshold both on a suitable coverage and on the pattern rank itself.

The coverage of selected patterns can be computed using the matching data from the ranking process. Since patterns may have overlaps, at this stage two primitives of the same type with equal labels are considered as equal, all primitives have to be considered independently. For example if two patterns each cover 50% of the input, this does not necessarily mean that they together cover 100% of the input. In the worst case the matched parts are identical, selecting both patterns then still yield only a total coverage of 50%.

The coverage is also 'uncertain', just as during the ranking process. The coverage of a set of patterns P over the set of input primitives $I = T \cap R$, extracted terms T and relations R , can be expressed as:

$$Coverage(P, I) = \frac{1}{\beta} \sum_{prim_I} \max_{prim_P} (cover(prim_I)) \cdot conf(prim_I) \quad (7.12)$$

Where the value is normalised through $\beta = \sum conf(prim_I)$, where $conf(prim_I)$ is the original confidence values of the extracted primitives, and where $\max_{prim_P}(cover(prim_I))$ denotes the maximum partial coverage of any pattern primitive over a specific input primitive. Whether $cover$ denotes a concept coverage, $cover_c()$ in equation 7.4, or a relation coverage, $cover_r()$ in equation 7.6, depends on the primitive, if it is a relation or a term.

7.5.6 Ranking experiment

An initial experiment was performed using the above approach. In the experiment the focus was on contrasting the specific features of OntoCase ranking compared to similar approaches, such as ontology ranking and keyword-based search, but it is not to be considered a complete evaluation.

Experiment overview

From the catalogue of patterns used in previous research experiments, see chapter 6, 10 patterns were randomly selected. Due to space limitations it is not feasible to show the details of each pattern, but in Table 7.3 the general topic and number of concepts of each pattern are presented. It is worth noting that all patterns are 'connected', in a graph sense. The general domain of the experiment for which the patterns were originally used was product development enterprises, with focus on requirements engineering, but the patterns are not domain specific with regard to industry domain, only the type of enterprise ontology.

A subset of the texts used for the same experiment as mentioned above was selected and the texts were reduced to a small corpus of six short documents. The origins of the texts are process descriptions and project plans from a single enterprise. Based on these texts the patterns were first manually ordered, through a card sorting-like method letting an ontology engineer order every possible pair of patterns, based on the perceived fit. This process was conducted twice, once with the instruction to order patterns based on topical content and once to order them regarding the structure

Table 7.3: Example patterns

	Name	Conc.	Topic description
a)	Product categories	11	Classification of product category concepts.
b)	Work effort	24	Describing work effort connected to requirements, organisations and work effort categories.
c)	Validation	19	Steps and techniques in validation and testing.
d)	Actions	8	Types of actions and their connections to plans.
e)	Requirements analysis	4	General tasks of requirements analysis.
f)	Organisations	8	Taxonomy of organisation concepts.
g)	Parties	14	Classification of different parties.
h)	Product associations	10	Categories of associations between products.
i)	Positions	7	Types of positions and their fulfilment by a party.
j)	Product features	24	Categories of product features and their attributes.

and perceived utility. The combination of these two orderings, from three ontology engineers, resulted in the manual ranking of patterns that is used for comparison against the automatically computed ranking orders below. It is not to be perceived as the completely 'correct' order of patterns, only as an indication of how human ontology engineers perceived the patterns. To create a true 'gold standard' a larger group of subjects would be needed.

The text corpus used for the manual evaluation was processed by an existing OL system, proposed by Maedche [123], resulting in 48 terms, at a frequency threshold of 10, and 60 relations, each associated to a confidence value. This extraction was not considered as a main factor of this experiment since all automatic ranking was based on the same input, but compared to the manual ranking this constitutes a bias.

First, keyword search, both exact and inexact matching, was used, ranking the patterns according to how many keywords were present in the pattern. The inexact matching was based on string inclusion, how large part of one string was covered by the other. In the experiment the threshold for considering the terms as 'matching' was set to 0.5. The rank was then computed as the fraction of pattern concepts matched.

Next, ranking was performed using the *AktiveRank* ontology ranking scheme as described by Alani and Brewster [6], since this was considered to be the most similar ranking scheme among the related work. A slight simplification was made concerning the density measure, removing the indirect relations count. The experiment used a re-implementation of the ranking scheme, based on the description presented by Alani and Brewster [6], where ambiguities existed the formulas and not the textual explanations were used.

Finally, the *OntoCase* pattern ranking approach presented in this paper was applied. As an inexact string matching measure the same string inclu-

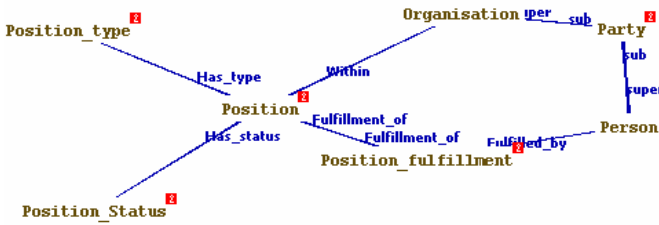


Figure 7.14: An ontology design pattern including the positions concept.

prototype(1.0)	information(0.61)
ascm(0.81)	module(0.61)
meeting(0.81)	phase(0.61)
acsm(0.69)	product(0.61)
mini(0.69)	production(0.61)
overview(0.69)	sw project manager(0.61)
quality(0.69)	sw project management(0.61)
sub(0.69)	system(0.61)
sw verification(0.69)	team(0.61)
description(0.61)	verification(0.61)
document(0.61)	work(0.61)
engineer(0.61)	...
engineering(0.61)	

Figure 7.15: An extract from the terms list used.

sion as in the previous approaches was used. The comparison should not be viewed as a full evaluation of the proposed research, merely as an indication of its usefulness. More thorough evaluations of the complete OntoCase approach, including the ranking of patterns, are presented in chapter 8.

Example ranking computation

To illustrate the actual process of computing the ranking score of one specific pattern, we look at one example pattern and present the steps in detail. To increase the understandability of this example one of the smallest patterns were chosen, the *positions* pattern, pattern i in Table 7.3. This pattern can be illustrated as in Figure 7.14 and a part of the extracted terms, the term list T , used for matching can be seen in Figure 7.15.

In the case of this pattern, no term matches any concept label so $dc(c_i) = 0$ for all c_i in the *positions* pattern. On the other hand, both the concept

labels *person* and *organisation* covers a set of terms through WordNet hypernym chains. Four terms, *engineer*, *designer*, *developer* and *manager* can be found in WordNet as hyponyms of *person*. In the case of *engineer* for example, it has two senses in WordNet and in one sense *engineer* is a direct hyponym of *person*. The contribution of this to the total subsumption coverage of *person* is then $\frac{1}{1.2} = 0.5$, and weighted by the confidence $0.5 \cdot 0.61 = 0.305$. The contributions from the other three terms are summed up and the result is $sub(person) = 0.845$ (equation 7.2). Analogously the coverage of *organisation* is determined to be $sub(organisation) = 0.677$. The vertical heuristic does not give any contribution, so the total coverages are equal to the subsumption coverages, thus $cover_c(person) = 0.845$, $cover_c(organisation) = 0.677$ and $cover_c(c_i) = 0$ for all other c_i in the pattern (equation 7.4). The total concept coverage of the pattern is then (according to equation 7.5):

$$CC(positions, T) = \frac{1}{7}(0.845 + 0.677) = 0.217$$

No direct concept matches were found in this pattern, thus no relation matches can be found this way. In addition no relation labels match, so in total no relation matches can be found, since the relation matching relies on the direct concept coverage $dc()$ and label matching. When computing the utility scores for the pattern only the two matched concepts *person* and *organisation* were used. When computing the density of the concept *person* it was noted that *person* has one taxonomically related concept, namely its superconcept *party*, but it also has one sibling in the taxonomy, namely *organisation*. Additionally *person* is related to the *positionfulfillment* concept. The density of *person* is therefore computed as $den(person) = \frac{(1+1+1)}{7} = 0.428$ (according to equation 7.8). After a similar computation for the *organisation* concept, the total density score can be computed as (according to equation 7.9):

$$DE(position, \{person, organisation\}) = \frac{1}{0.845 + 0.677}(0.845 + 0.677) \cdot 0.428 = 0.428$$

Next, the proximity between all possible pairs of matched concepts was considered. In this case there existed only one possible combination, the pair *person* – *organisation*. The length of the shortest path between the two concepts was 2 and the diameter of the pattern was 3. The proximity of *person* and *organisation* is then $prox(person, organisation) = 1 - \frac{2}{3}$.

Table 7.4: Ranking order (1-10) of the patterns (a-j).

Method — Position	1	2	3	4	5	6	7	8	9	10
Manual	b)	i)	h)	j)	e)	d)	a)	c)	g)	f)
Keyword	b),d),f)			h)	a)	c)	j)	e),g),i)		
Keyword(inexact)	b),e)		j)	d),f)		c)	h)	a)	g),i)	
AktiveRank	j)	b)	a)	h)	c)	e)	d),f)		g),i)	
OntoCase	h)	b)	a)	e)	i)	j)	f)	g)	d)	c)

$(1 - \frac{2}{7}) = 0.524$ and the total proximity score of the pattern is (according to equation 7.10):

$$PR(positions, \{person, organisation\}) = \frac{1}{1}(0.524) = 0.524$$

Finally, the rank of the pattern is computed through combining the four measures with equal weights (according to equation 7.11). The final ranking value is

$$Rank(positions) = 0.25 \cdot 0.217 + 0.25 \cdot 0 + 0.25 \cdot 0.428 + 0.25 \cdot 0.524 = 0.292$$

Results and analysis

The result of the ranking experiment can be viewed in Table 7.4. Since none of the ranking orders represent the 'correct' ranking there is no possibility to completely assess the overall performance of these ranking schemes. We will instead attempt to analyse their general features, and find benefits and drawbacks for the specific case of pattern ranking for automatic ontology construction.

First, we note that some patterns were ranked highly by all approaches, such as pattern *b* concerned with work effort. This is not surprising since the texts were gathered from plans and process documentation. In addition, the pattern uses mainly the same terminology as the texts since it is ranked highly even using exact term matching. In the general case however, such patterns will be rare and quite domain specific.

Much more commonly there are abstract and domain independent patterns, such as pattern *i*, the *positions* pattern described in the previous section. For a human assessor it is natural to rank such a pattern highly, since plans and process documents focus on different positions for various projects that are in turn filled by specific persons. In the experiment presented above, out of the automatic ranking approaches OntoCase is the only

one able to find the connections between specific extracted terms and the more general concepts, and thereby more intuitively rank this pattern.

Compared to *AktiveRank* the effect of excluding the centrality and tuning the measures in *OntoCase* is notable. *AktiveRank* ranks large patterns, with a substantial taxonomic structure, higher than the *OntoCase* approach. Examples of this can be seen in the ranking of patterns *j* and *c*, two of the largest patterns, which in both cases differ five steps in the ranking order between *AktiveRank* and *OntoCase*. Generally, patterns are reasonably small and abstract, to increase reusability and composability, thereby *OntoCase* is clearly better suited to assess such patterns.

Finally, a comparison can be made between the automatic approaches and the manual ranking order. As stated previously the manual ranking order does not represent a completely 'correct' order, but merely gives an indication of human intuition. Compared to the manual ranking order, the position of a pattern in the *OntoCase* ranking is on average 2.3 steps from the manual order, and the maximum difference of any pattern is 4 steps. The corresponding value of *AktiveRank* is 2.4, maximum 7 steps, and 2.7 and 3.2, maximum 7 and 9, for the approaches using keywords, inexact and exact respectively. Although no general conclusions can be drawn from such a small dataset we note that the *OntoCase* ranking scheme definitely has some desirable properties.

7.6 Notes on the *OntoCase* implementation

A first version of the *OntoCase* method, including the two first phases, was implemented as a proof of concept software during this research. This implementation was subsequently used for the evaluations. The software was implemented using the Java language, as a stand-alone command-line application. The Jena API* was used for handling ontologies. Initially the *SecondString*† string matching library was used, but later we switched to the *SimMetrics*‡ library which is more actively supported. Additional external software required is the *WordNet*§ lexical database.

The pattern base has to be provided separately, and is currently deployed as a *MySQL*¶ database, however the coupling to the software is loose and

*<http://jena.sourceforge.net/>

†<http://secondstring.sourceforge.net/>

‡<http://sourceforge.net/projects/simmetrics/>

§<http://wordnet.princeton.edu/>

¶<http://www.mysql.com/>

this could easily be exchanged for any other type of relational database supporting remote network access. The pattern files, the actual ontologies, are locally stored as OWL-files or directly linked to online ontologies on the web, as in the case of patterns present on the ODP portal^{||}.

The initial text processing can be done through the Text2Onto** tool. An interface to the tool is provided by OntoCase. The alternative is to provide an input ontology represented as an OWL-file. This can be constructed using any OL software, or an existing ontology can be used. Several variables can then be set for the construction process, e.g. the ranking thresholds and the mode of composition. There are two modes of ontology composition implemented, a pruning mode and an enrichment mode including the complete input representation in the output. The pruning mode uses the patterns as a basis and adds the matched parts of the input representation using the heuristics, then stores the ontology in an OWL-file. The enrichment mode uses the opposite approach, starting from the input representation and adding the matched parts of the patterns.

Some of the ranking and composition data is saved in a log file, e.g. the ranking scores of each pattern, the list of selected patterns, and the added elements of each pattern. For analysis purposes also some statistical measures for ontologies were implemented. Existing tools that provide statistics usually cannot handle ontologies with cyclic definitions. In our implementation, for example the average depth of the ontology is computed by only considering the shortest path from each concept to the root concept, in this way avoiding any cycles in the taxonomy.

The implementation is flexible, and portable since it is implemented in Java. Some external components are needed, but if used without the Text2Onto integration the only input needed is actually a set of OWL-files, i.e. the input representation and the set of patterns. At the moment the implementation does not provide a graphical user interface, but this is part of future work. The OntoCase method could easily be integrated as a plug-in for almost any existing ontology editing tool, only requiring support for the OWL language and supporting a plug-in structure. Plug-in implementations for both Protégé^{††} and the NeOn toolkit^{‡‡} will be considered in the future.

^{||}<http://ontologydesignpatterns.org>

^{**}<http://ontoware.org/projects/text2onto/>

^{††}<http://protege.stanford.edu/>

^{‡‡}<http://www.neon-toolkit.org/>

7.7 A small example

Before discussing the actual evaluations of OntoCase in the following chapter this section will provide an illustrative example, using a very small 'toy' ontology as input. The input ontology shows typical characteristic of ontologies extracted through existing OL methods, it is quite diverse and contains only a very shallow taxonomy with a high number of concepts directly beneath the root of the taxonomy, in this case owl:Thing. The aim of this example is to first give an intuition of how OntoCase typically transforms the input when reusing the patterns. Some typical characteristics and effects on the output of the OntoCase process can be noted. In later evaluations it will then be shown that OntoCase can actually improve the quality of larger ontologies, but in this case we are only trying to give an illustrative example whereby no evaluation measures are applied.

7.7.1 Ontology construction

In this case the input is not a text corpus, instead we use a small ontology as input to the OntoCase method. The ontology used is a smaller version of the example used in section 7.5.6. The ontology was reduced to 19 concepts and 8 object properties, not counting their inverses. An illustration of the ontology can be viewed in Figure 7.16 The illustration uses UML notation and has been produced using the visualisation support in the ontology editor TopBraid Composer*. In the illustration we can see that 15 out of 19 concepts have no superconcept at all, except owl:Thing which is the superconcept of all concepts in an OWL ontology, and that 7, i.e. over one third, of the concepts have no connection whatsoever to the rest of the concepts. All properties are unnamed, but has a domain and a range defined. This is a good illustration of typical ontologies produced by existing OL systems, such as Text2Onto that will be used later.

The output of running OntoCase, with the pruning alternative, on this input ontology can be seen in Figure 7.17, illustrated using the same notation as before. The resulting ontology was constructed based on 15 patterns that had sufficient matches to be included, but most patterns were only partially matched and thereby only partially included in the resulting ontology, i.e. only some concept or relation had any match. In total the resulting ontology contains 28 concepts and 13 object properties, not counting their inverses. One may wonder how 15 patterns were used for this small ontology,

*<http://www.topquadrant.com/topbraid/composer/>

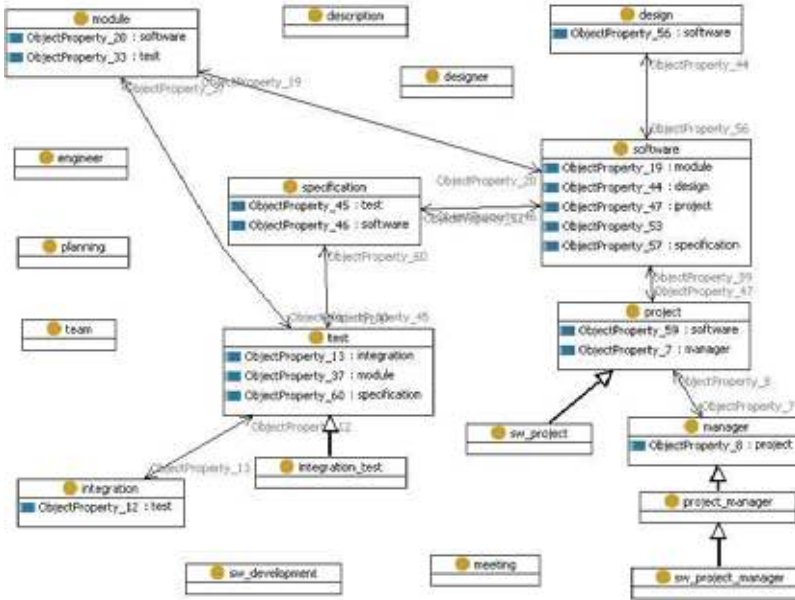


Figure 7.16: The concepts and properties of the input ontology.

but if considering the nature of the very general, domain independent, patterns that were reused here, many of the patterns are also overlapping. As an example 5 of the reused patterns contained the concept 'object'. These patterns are overlapping, but since all are extracted from the top-level ontology DOLCE, this is indeed the same notion of 'object' and not different concepts. The patterns in turn add different aspects of the 'object' concept. The 'types of entities' pattern adds object as a subconcept of 'entity', while 'participation' and 'co-participation' add the relations to 'event'. Other patterns contribute only with one concept, such as the 'person'-pattern that constitute of only of one concept, since in this case none of the properties are matched and included.

7.7.2 Result and analysis

The concept coverage of the output ontology over the input ontology terms is quite high, since 17 of 19 concepts present in the input ontology are also present in the output, this gives a coverage of 89% with respect to concept inclusion. All object properties of the input ontology are also present in the

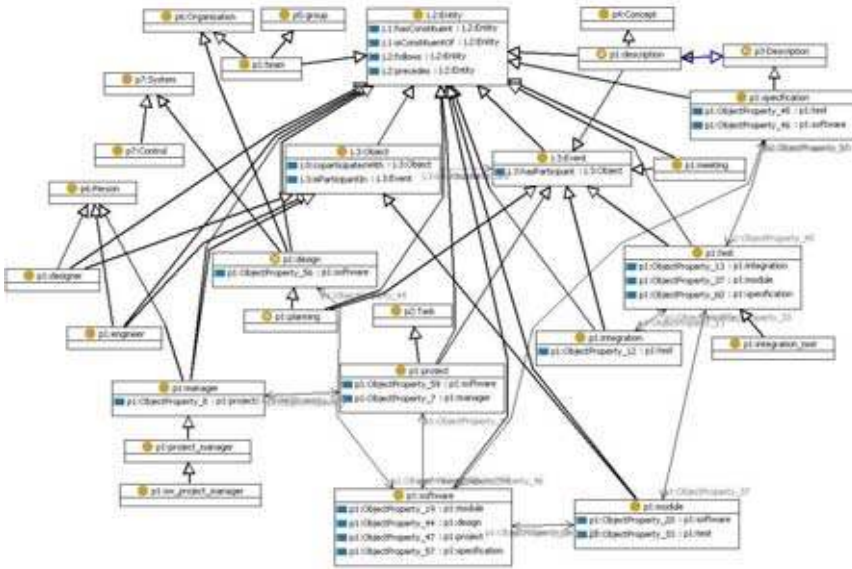


Figure 7.17: The concepts and properties of the output ontology.

output, hence the relation coverage is 100%. The pruning method chosen for this example has resulted in the removal of two concepts and no properties, based on the matching results. Concepts and relations have in this way mainly been added to the input ontology, rather than filtered or removed.

This is one of the major features of OntoCase. The intention is to add background knowledge through the use of patterns, and to thereby add both a general abstract layer to the ontology and give the ontology more structure in the form of more relations. Since learnt ontologies rarely have any formal axioms that define the concepts, the interrelations of concepts is sometimes all we have in order to draw conclusions about the semantics of the concepts, i.e. the 'meaning' behind the terms representing the concepts. Therefore it is essential that such a light weight ontology contains many properties that can give some clues as to what the meaning of concepts may be, in order to first of all increase the understandability of the ontology but possibly also the utility. This was noted in the initial SEMCO experiment described in section 6.2, where the domain experts found the concepts of the automatically constructed ontology easier to comprehend and interpret since they had more relations connecting them to other concepts.

The most remarkable change in the ontology is the addition of quite a few

subclass relations between the learnt concepts, the input ontology, and the added pattern concepts. At a first glance this may look like a complete mess, but looking a bit closer we may note that this is actually a set of alternative modelling choices, all included for the future evaluation and selection by the ontology engineer. Consider for example the concept 'specification' in the top right part of Figure 7.17. It is a direct subclass of both 'entity' and 'specification'. This is a common scenario where a redundant subclass statement has been added. This redundancy could quite easily be filtered out from the matching data before the statements are added, through only considering the best matching score for example, but at the moment it is by choice not done in the OntoCase implementation. The rationale behind it is to include as much hypotheses as possible, since the method is intended to be used in combination with manual editing and it is generally easier to erase parts that are not needed among a set of alternatives, rather than to think of new alternative that are not present.

In the top left corner of the figure the concept 'team' shows another good example of this. Team is here both an 'organisation', a 'group', and an 'entity'. This is not redundant in the same way as for the 'specification' concepts, but it still represents three alternatives. Probably a final version of the ontology will not contain all three statements, but including them at the moment will provide the opportunity to consider all three possibilities before choosing one or several to remove. Note also that both 'team' and 'specification' had no superclasses at all in the input ontology, and now they are connected to several pattern concepts, which give the added top structure we were looking for. In addition the patterns provide a set of general properties that may now be specialised and used for the subclasses. The 'entity' concept for example provides the properties `hasConstituent` and `isConstituentOf`, that may be used with other entities. Again we can consider the concept 'team' which is stated to be a kind of entity, and the concepts 'designer' and 'engineer' which are also proposed as entities. A next, manual, step could be to specialise the constituency relations into team membership relations, specifying that designers for example may be a constituent of a software development team. Finding such relation specialisations automatically is still future work.

In addition to the subclass relations added, also some equivalence axioms have been added based on matching results. The concepts 'design', 'planning' and 'description' were all found to directly match a concept in a pattern, using the same term as a concept label. In each case, both concepts were included in the output ontology, but proposed as equivalent through

an equivalence axiom. For the 'description' concept the situation is a bit more complex, since both evidence of a subclass relation and the equivalence relation was found. This is the reason why both those alternatives are included, as we can see in Figure 7.17.

As stated previously this example aimed at showing some typical effects of applying OntoCase, before going into detail with respect to the larger datasets used for the the evaluations presented in the following chapter. The example above shows how OntoCase adds a general top structure to the constructed ontology, thus adding some of the general knowledge that is implicit in the domain. For example stating that planning is a kind of design, that designers and engineers are people, although this is not always a perfect modelling choice, and that objects can participate in events. Numerous relations are also proposed, connecting the unconnected parts of the ontology and proposing different interpretations of the ontology concepts. At the moment a special purpose user interface to deal with the proposed alternatives and their rationale originating in the pattern matching, and even the text processing, is still future work. However, even just adding this structure gives some added meaning to the concepts that were in the input ontology simple isolated terms without any kind of explanation or suggestions for how to interpret them or how to continue their modelling.

Chapter 8

Evaluation of OntoCase

This chapter contains three different sets of experiments using the so far implemented parts of the OntoCase framework, i.e. the retrieval and reuse phases. The intention was partly to validate the proof of concept implementation, to show that it is in fact possible to use ontology design patterns semi-automatically to construct ontologies, but more important is showing that the OntoCase method actually improves the results of existing ontology learning (OL) methods. This chapter also aims to show the improvements compared to the initial method, as described in chapter 6.

The first evaluation is a rerun of the SEMCO experiment, previously described in chapter 6. It was repeated as closely as possible, with respect to the original setting, but taking into consideration that the project ended some time ago and that not all of the project resources were available, such as the domain experts used in the previous experiment. The results of this experiment provide a proof of concept, adding to the evidence from chapter 6 regarding the feasibility of the approach, and clearly shows the improvements made to the OntoCase framework.

The second experiment is set in the context of the university domain, where an initial ontology is constructed for the task of structuring and annotating documents and information on a university intranet. The domain is Jönköping International Business School, at Jönköping University, which gave the opportunity to evaluate the ontology together with domain experts from the organisation in question. This experiment shows the applicability of the method to a company in a different domain, and points out some of the benefits of the pattern selection and composition methods compared to typical existing OL methods. Finally, OntoCase was applied in the agricul-

ture domain, in order to show the applicability of OntoCase even outside its intended scope, i.e. enterprise ontologies, and to again show benefits compared to existing state of the art systems for OL. In this setting the focus is also on showing the beneficial effects of the interaction between several OL methods and ontology patterns.

8.1 Extended pattern catalogue

The pattern catalogue used for the subsequent experiments described in this chapter consists of 41 patterns. It is an extension of the previous catalogue, where additional patterns that are now available online have been included. The patterns harvested from the web primarily originate from the ontology design pattern portal* (ODP portal). All currently available design pattern candidates were included, except those that included incorrect import references or other errors, and thereby were incomplete. The complete list of patterns can be viewed in appendix B.

When conducting the initial SEMCO experiment all the patterns were represented in an F-logic based language, this time the patterns were translated into OWL. This is also the reason for excluding a few of the patterns in the initial catalogue, since they did not translate well into OWL without considerable modifications. The reason for transferring the patterns into OWL, and using OWL as the base representation for both the second version of the implemented OntoCase method and the patterns is primarily that existing work both on OL and ontology patterns tend to focus more on OWL. In order to compare results to these existing methods, and to use the ontology patterns that have recently emerged, it is essential to be able to handle OWL ontologies.

The patterns included in the catalogue originate from several different domains, some are very general while others are somewhat domain specific. They are also differing with respect to size, some include a small taxonomy while others include just one or two concepts but instead some essential properties. In appendix B some details are given about each pattern in the original catalogue, including domain and number of concepts. Examples of a selected number of patterns can also be found in appendix B and on the ODP portal there are details of the remaining set of patterns.

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

8.2 SEMCO revisited

The first version of the OntoCase retrieval and reuse phases was used to construct an ontology within the context of the SEMCO project, as described in section 6.2. The project in itself ended some time before the finalization of this thesis and the main industry project partner has undergone a major reorganisation, whereby a complete rerun of the experiment was not possible. The intention was to construct a new version of the automatically constructed ontology, throughout this chapter denoted the second version of the automatically constructed ontology. However also an intermediate version was constructed, using the new OntoCase method but only relying on the initial pattern catalogues, this will be denoted the 'intermediate ontology' in this chapter. These two ontologies would then be compared to the version constructed automatically during the first SEMCO experiment, throughout this chapter denote the initial, or first, version of the automatically constructed ontology. Additionally the second version is compared to the final version of the SEMCO ontology, the combined ontology here denoted the final SEMCO ontology, representing an approximation of a 'gold standard' for the task at hand. However, no 'gold standard'-based evaluation has been used since we still believe that a real 'gold standard' should not be proposed without firm evidence from using that ontology in a real-world setting. Since the intended application of the ontology was so far not put into use within the organisation of the SEMCO project partners, it is not reasonable to claim that the final SEMCO ontology is really a 'gold standard', i.e. completely correct in all parts representing the domain and optimised for the intended task. Instead it should be viewed as a more complete and correct version than the initially proposed manually or automatically constructed ontologies, but not as a completely 'correct' ontology since a task-based evaluation was never conducted.

8.2.1 Ontology construction

This second SEMCO experiment involved the construction of two ontologies, both using the same method for semi-automatic ontology construction but one using the original pattern catalogue and the other using the extended catalogue. For constructing the ontologies, this time the updated method, i.e. the implementation described in section 7.6, was used. The initial step of text processing was conducted by using the same text extraction methods and tools as in the previous experiment, together with the same input texts

Table 8.1: Ranking values of the patterns in the SEMCO case.

Pattern name	Source	Pattern name	Source
Actions*	0.11	Validate and test	0.037
Analysis and modelling	0.00	Work effort	0.029
Communication event	0.0076	Time interval	0.00
Employee and department*	0.080	Precedence*	0.26
Engineering change	0.0074	Participation*	0.17
Information acquisition	0.00	Information realisation	0.00
Organisation*	0.074	Description*	0.25
Parts*	0.049	Agent role*	0.17
Party*	0.086	Classification*	0.5
Person*	0.25	Collection entity*	0.27
Planning and scheduling	0.00	Constituency*	0.27
Positions	0.038	Co-participation*	0.098
Product*	0.12	GO top*	0.25
Product associations*	0.11	Invoice	0.00046
Product categories*	0.084	Metonymy 1 FAO	0.011
Product features*	0.061	N-ary participation*	0.063
Requirements*	0.045	Object role*	0.19
Requirements analysis	0.00	Situation*	0.27
System*	0.28	Species v1.0 model*	0.15
System analysis*	0.054	Task role*	0.30
System synthesis*	0.15	Types of entities*	0.31

this meant that the same 190 terms were extracted and used as a basis for pattern retrieval and selection.

This time an extended catalogue of ontology design patterns was applied for constructing one of the ontologies. A definite pattern selection threshold, a combination of a rank value threshold and a threshold on the total coverage of the patterns, was used for the selection, resulting in 29 patterns being selected for the construction of the second version of the SEMCO ontology. The rank values of all the patterns can be viewed in Table 8.1 and selected patterns are marked with a '*'. For the intermediate ontology the original pattern catalogue was used, and the focus was set on tuning the thresholds in order to achieve an ontology similar in size to the first version of the SEMCO ontology. This resulted in 16 patterns that were matched and selected, compared to 14 patterns in the original experiment.

To be comparable to the previous ontology the pruning version of the composition algorithm was used to construct both the ontologies from the selected patterns, i.e. only the terms and relations that matched some pattern were also included in the ontology. A threshold was set on the matching values for synonyms, but no threshold was set on subsumption matches. The selected patterns were added to the resulting ontologies, together with all the terms and relations that had some match, above the set thresholds. Finally, the ontologies were exported to OWL-files, disregarding the confidence values of each element. This resulted in the intermediate

ontology having 90 concepts and 37 object properties and the second version of the ontology based on the extended pattern catalogue having 150 concept and 48 object properties, both represented as OWL ontologies.

8.2.2 Evaluation setup

To be able to compare the results with the initial SEMCO ontologies the new versions, the intermediate and the second version, the ontologies were primarily evaluated using the same evaluation measures as during the first SEMCO experiment, described in chapter 6. First, some basic structural measures were used, to give a general idea of the nature and characteristics of the ontologies. For the intermediate ontology, this was considered enough to show the differences to the initial experiment, whereby it was not evaluated further. A discussion about the differences was provided instead.

However, for the second version of the SEMCO ontology, the same kinds of structural evaluations as in the initial SEMCO experiment were performed, in order to find errors in the taxonomy of the constructed ontology. First the taxonomical evaluation of Gómez-Pérez [84] was performed and then the OntoClean method was applied. The third and final part of the evaluation would have been the evaluation together with domain experts, but since the situation does not allow this, not only have the project ended but additionally a reorganisation of the main industry partner has made it impossible, instead we provide a discussion of the changes compared to the automatically constructed ontology of the initial SEMCO experiment and how these changes relate to the aspects of the previous evaluation with domain experts.

8.2.3 Evaluation results and analysis

Below results from the three parts of the evaluation are presented and discussed in detail. concerning the intermediate ontology results are only present for the first structural evaluations, but it is also mentioned in the discussion of functional measures later in this section.

General characteristics

The collected data from the structural measures can be seen in Table 8.2, together with a comparison to the ontologies of the initial SEMCO experiment. Aut_1 denotes the automatically constructed ontology from the initial experiment, 'Man' the manually constructed ontology, 'Comb' the

Table 8.2: General characteristics of the SEMCO ontologies.

Characteristic	Man	<i>Aut</i> ₁	Comb	<i>Aut</i> ₂	<i>Aut</i> ₃
Number of concepts	224	85	379	90	150
Number of root concepts	8	35	5	21	13
Number of leaf concepts	180	64	273	65	107
Avg depth of inheritance	2,52	1,95	3,5	2,10	1,62
Avg number of rel. concepts	0,13	0,79	1,30	0,82	0,64
Avg number of subclasses	1,00	0,57	1,00	1,06	1,62

combined ontology from the SEMCO project, *Aut*₂ the intermediate ontology constructed using the initial pattern catalogue but using the improved method, and finally *Aut*₃ denoting the final version of the automatically constructed ontology, constructed using the extended pattern catalogue. The count of attributes has been excluded since this version of the ontology is represented in a different way, i.e. in OWL, and the notion of attribute used in the first experiment is not comparable to what could be denoted attributes in this OWL ontology.

The intermediate ontology is comparable in size to the initial SEMCO ontology. The initial SEMCO ontology had a coverage of 34% over the input terms, while the intermediate ontology covers 38% of the input terms. This increase in coverage is not significantly large, but the intention was to produce an ontology of similar size using the improved method but the same patterns as before, and then study the differences in terms of structure. The number of root concepts has decreased from 35 to 21 in the intermediate ontology, and we can note general concepts such as 'activity' and 'facility' that were not present in the initial version of the ontology, and are not present in the input. These concepts have been matched using the methods in the improved OntoCase method for bridging the abstraction gap between specific terms and general patterns concepts, and provide a more general top structure for the intermediate ontology. The concept 'activity' has subclasses such as 'inspection', 'release' and 'test' that would not have been possible to connect to the general concept of 'activity' using the initial method in chapter 6.

There is a slight increase in the depth of the ontology, and in the number of general, non-taxonomic, properties per concept, from the initial version of the ontology to the intermediate ontology. Most significant, however, is the increase in the average number of subclasses per concept, from 0.57 in the initial version to 1.06 in the intermediate version. This is due to the ability

of the new OntoCase method to add such relations, i.e. subsumption, based on matching information, which was not present at all in the initial version of the method, where only direct overlap, i.e. equivalence, was considered. The conclusion we can draw from this comparison is that the improved OntoCase method does in fact construct ontologies with a more general top structure, and with an increased number of relations added, as was the intention when updating the method.

Next, we consider the final version of the ontology, where not only the method was changed but we also used the extended pattern catalogue. This ontology has a quite general top-structure with only 13 concepts directly beneath the root concept of the taxonomy, which is even less than for the intermediate ontology. This fact is due to the more general pattern that were now present in the pattern catalogue, and could be added to provide this abstract structure. The ontology is also considerably larger than both the automatically constructed ontology of the initial SEMCO experiment and the intermediate ontology, thus indicating that the OntoCase method is able to identify more connections between what can be extracted from the text input and what is present in the patterns, even though the patterns were now even more general.

The number of subclasses per concept has increased drastically, even compared to the intermediate ontology using the same construction method. This is due to that some very general concepts such as 'event' and 'object' present in the added patterns attracted a large number of subconcepts. This is also the main reason for the decrease in average depth, because many of the new concepts were added to some of the most general patterns, thereby creating a quite shallow taxonomy for those parts of the ontology. The average number of non-taxonomically related concepts has also decreased, this is due to the ratio between added concepts and added properties, since many more concepts have been added while only a small amount of additional properties were found.

A lot of new concepts were matched and added to the ontology, but a significant number of the relations, properties, originate in the patterns so there was no significant increase in the number of properties added from the extracted input. The ontology is thereby not very well connected through properties on the lower levels of the taxonomy, the object properties originating in the general patterns connect the top concepts. However, these are still very valuable, since they can easily be specialise manually and adapted to more specific concepts. Compared to the ontology constructed in the initial experiment the second version thus has a lower average number of

directly related concepts, but it is still very well connected on the upper levels of the taxonomy, while many more concepts are now present on the lower levels.

The coverage of the 190 input terms, percentage of extracted terms that also appear in the constructed ontology, is for this final version of the ontology 64%. This number should be compared to the 34% coverage of the automatically constructed ontology of the initial SEMCO experiment. When studying the terms extracted from the text corpus, a set of terms can immediately be determined to be 'junk' originating from flaws the text processing algorithms, such as letter combinations that are not English words or misspelled abbreviations, and single letters. Taking this into account, considering that it is intuitively correct not to include these in the ontology, the coverage over the reasonable input terms is instead 70%. This is quite a good coverage, which in this case means that an ontology engineer only would have to take care of 68 unconnected terms, including the 'junk' terms mentioned, if starting to refine the ontology in order to cover all the extracted terms.

Taxonomic evaluation

In this section the application of two methods for structural evaluation of the correctness of the taxonomy to the final version of the automatically constructed SEMCO-ontology is described. The results were analysed and some examples of problems and errors are discussed below.

The general taxonomic evaluations proposed by Gómez-Pérez [84] was applied, and during this evaluation some interesting observations were made. This time, compared to the initial SEMCO experiment in chapter 6, the ontology was much more 'tangled', i.e. it contains numerous cases of multiple inheritance and redundant relations, thereby this evaluation yielded more interesting results than for the simpler taxonomies of the initial SEMCO experiment. The reason for the increased 'tangledness' of the ontology is primarily the merging of different kinds of evidence that is conducted during the pattern composition and ontology construction in OntoCase. There can be both evidence for synonymy and hyponymy between two terms, for instance, and in this case both hypotheses were included, as equivalence and subclass relations, in the proposed ontology. The intuition behind this is that since it is an initial ontology, to be further developed either manually or by some other automatic methods, it is more useful to include all hypotheses than to discard some of them based on uncertain evidence.

The evaluation method proposes to first focus on circularity errors. Multiple inheritance is certainly present, and also some circularities were found, but only on the level of distance 0. This means that a concept was stated to be the subclass of itself. The reason for these errors is that the hypotheses from the matching phase are not filtered, thereby a term that is found to be a synonym of a pattern term will also possibly be found in for example WordNet and thereby proposed also as a subclass. This is on the other hand a small and simple error to correct, since if it is desirable to exclude such errors a filter could be applied during the construction process, e.g. putting a preference on equivalence before subclass relations in case both are suggested. On the other hand it is not entirely clear that it is desirable to exclude these relations, intuitively these circularities suggest a choice to be made by the ontology engineer, to either keep an asserted equivalent class axiom or the subclass relation. A future interface for OntoCase should use such redundancies and support the user in making informed choices.

The ontology does not contain any instances so we can only consider the structure of classes for the evaluation of partition errors. There are a few places in the ontology where there exist common classes in disjoint partitions. This includes the concept 'part' which is both a subclass of the concept 'object' and the concept 'role'. In this case it is a question of choosing how to model parts in the ontology. If the enterprise in question considers parts as physical components of their products, and thereby objects, this alternative should be chosen. However, depending on how the ontology will be used it might still be more suitable to model 'part' as a role that a certain object can take in a certain situation. This is an important modelling choice that is pointed out by this 'error' in the ontology.

Another such error found is that 'description' is a subclass of 'concept' but these two concepts are also disjoint. This error most likely originates in the fact that the very general concept of 'description' can be used on several levels of abstraction. Probably the enterprise in question refers to an actual textual description when they use the term 'description' while the ontology patterns applied use 'description' and 'concept' as two distinct and disjoint concepts interacting to form definitions of concepts and describe situations. In this case it is actually the question of 'description' having two slightly differing definitions, although in the ontology they are set as synonyms. This is a much harder issue to solve than the previous one described. In this case the description concept would have to be defined in detail and possibly there is a need for two distinct concepts, referring to the different meanings of 'description'.

There is one complete decomposition defined in the ontology; 'entity' is stated to have the complete partition 'abstract', 'quality', 'event' and 'object'. In total these five concepts have 208 subclasses, where 92 concepts are represented more than once or subclasses of 'entity' directly, i.e. 44% of the subclass relations in this sets violate the complete partition restriction. A large number of these violations occur because concepts are found to be both direct subclasses of 'entity' and subclasses of one of the four subclasses. This is again a matter of presenting the redundant findings to the user instead of resolving conflicting evidence automatically. The rest of the errors occur due to that a concept has been found to be subclass of two or more of the subclasses of 'entity'. In most of these cases it is again a question of disambiguation of the extracted concepts, and this task is currently up to the ontology engineer refining the ontology.

Next, the semantic errors of the ontology were studied. 29 semantic errors were found in the ontology, these are all due to invalid classifications of extracted concepts. The discovery of semantic errors was done based only on domain knowledge of the enterprise in question, not on any task requirements of the ontology. The majority of these errors occur because some general background knowledge has been used in the pattern matching process, whereby sometime the knowledge added is too general. It may be true that 'acceptance' is a kind of tolerance in a general sense, but in this specific setting 'acceptance' is intended to mean the situation when the customer in fact accepts the product as the solution they ordered. Fortunately this second, more specific, meaning is also represented in the ontology, because acceptance is also a subclass of 'situation'. This illustrates how many of these errors again represent alternatives that may be incompatible but can still be valuable to the ontology engineer in order to select the correct one.

Incomplete concept classifications might be present in the ontology, but since this is closely related to the task of the ontology it is not possible to clearly identify these errors. The same problem arises with the partition errors, what knowledge is really required to be present in the ontology has to be determined using a detailed task description, or by running the actual system using the ontology, but neither of these are available in our case. Finally, checking for redundancy is the last step in this first taxonomic evaluation. There are no direct repetitions of subclass relations, but as we have noted several times before there are many cases of redundant subclass statements in the ontology, i.e. there exist quite a few indirect repetitions. As also stated previously, many of these can be viewed as suggestions for fur-

ther refining the model, or alternatives for making some modelling decision, whereas we believe that it is not entirely a bad thing that these redundancies exist. There are no identical explicit definitions of any classes, but in a few cases some synonyms were not detected by OntoCase, and thereby for example 'organisation' and 'organization' are two different concepts, as well as 'doc' and 'document', and 'sw' and 'software', and some compound terms containing the software term.

The second part of the taxonomic evaluation consisted in applying the OntoClean methodology on the second version of the ontology. A summary of the OntoClean results can be viewed in table 8.3. The number of errors found is comparable to the number found within the initial ontology constructed previously, which is a reasonably good result since the new version is considerably larger and thereby could potentially contain more errors than the smaller initial ontology. First, the concepts in the ontology were tagged with the rigidity property, whereas a backbone taxonomy of 46 rigid concepts could be distinguished. Incompatible identity deals with how instances are identified, and in this category two errors were detected, both connected to the concept 'person'. Both 'supplier' and 'customer' are in the constructed ontology subconcepts of 'person', which in the general case might be correct, but in this specific case we are modelling an enterprise that has other enterprises as both customers and suppliers. Persons can be identified by, for instance, their social security number, while companies, i.e. customers and suppliers, can be identified by other means and do not have social security numbers. This fact points at a modelling problem, 'supplier' and 'customer' needs to be organisations rather than persons.

Among the incompatible unity criteria found in the ontology is the fact that 'name' and 'source' are kinds of objects. An object generally has some spatial limit and the constituents of an object can be seen as a unit connected through their spatial and physical relatedness. 'Name' on the other hand is an abstract concept, which intuitively does not have any requirement on spatial or physical relatedness of its individual parts, whether those parts are the characters making up the name, or the first name and family name for instance. 'Source' is in the context of this enterprise referring to source code of the software produced, and a unit of source code does not necessarily have any spatial or physical relation connecting its parts. The third error is the classification of 'doc' as a kind of 'department', which is a clear error since 'doc' in this context is an abbreviation of document, which is not a department. Considering unity, one of the unity and anti-unity violations originates in 'focus' being a kind of 'object'. While an object can usually

Table 8.3: Results of the OntoClean evaluation.

OntoClean rule	No of cases
Incompatible identity	2
Incompatible unity criteria	3
Unity/anti-unity conflict	2
Rigidity/anti-rigidity conflict	No

be viewed as a whole, connected through spatial or physical relations as mentioned above, 'focus' is an abstract notion which could refer to any combination of abstract notions, events or objects. Additionally 'audit' is found to be a kind of 'document', which is opposed by the unity and anti-unity criteria, since audit is a composite event able to contain any kind of participants and actions while document is a well-defined piece of text collected in a computer file or as a printed document. In this enterprise context 'audit' is more likely a kind of task or event, while a 'document' is a physical record that may record the results or aims of an audit session, but the audit itself is not a document.

Analysis of previous domain expert evaluation

An evaluation of the new versions of the ontology together with domain experts from the enterprise in question was no longer possible, as mentioned previously. Instead the evaluation results of the initial SEMCO experiment are compared to the changes that occurred in the ontology and a discussion is provided on how these changes most likely have affected the benefits and drawback that the domain experts found when evaluating the initial ontology. In the initial SEMCO case the evaluation was conducted through interviews with domain experts, where the domain experts examined the ontology and assessed its characteristics on a scale from 'very low' to 'very high'. The characteristics were related mainly to the content of the ontology, and were divided into four parts; concepts, taxonomy, relations and axioms.

In the concept section of the evaluation the domain experts found the initial automatically constructed ontology to contain very few essential concepts on the higher levels of abstraction, while the manually constructed ontology contained a high number. Top level concepts of the manually constructed ontology were concepts such as 'artefact' and 'process', while the automatically constructed ontology had concepts such as 'requirement status' and 'part specification' on the top level. The new versions of the au-

tomatically constructed SEMCO ontologies, the intermediate and the second version, contain much more general concepts. The most general top structure can be found in the final version, containing concepts such as 'entity', 'facility', 'dimension' and 'organisation'. The intermediate ontology lies somewhere in between, due to that only the domain specific patterns were used, but the abstraction level is still considerably higher than in the initial version of the SEMCO ontology. The possible level of abstraction, at the top level, has undeniably been raised, although the level of abstraction is still not uniform across all top level concepts of the automatically constructed ontologies. Whether these top concepts are really essential or not would however only be possible for domain experts to assess.

The initial automatically constructed ontology was deemed to contain too few essential concepts in general. This was mainly due to the low coverage of domain specific terms. This coverage has now been raised from 34% to 38% in the intermediate ontology and 64% in the final version, both due to the improved method and the extended pattern catalogue. This is however a coverage measured over the extracted terms, to improve this even further also improvements of the text processing and term and relations extraction algorithms are needed, to more precisely cover the intended scope. It can be noted that the combined ontology contained 379 concepts, neither of the automatically constructed ontologies are anywhere close to achieving this good coverage of the intended scope.

Formal definitions and attributes were already in the initial ontology quite highly rated, unfortunately this rating might have been slightly lower for the final version of the ontology since many more concepts have been added but the number of relations has not increased at the same rate. This is the main conclusion that can be drawn from the section of the evaluation regarding relations. The initial automatically constructed ontology was very rich in relations and by adding mostly concepts, and not a corresponding amount of relations, the final version is now slightly more sparse. However, that this is not a flaw in the actual method can be seen by comparing the initial ontology and the intermediate ontology, where both the coverages and the relation ratios are similar. It should be noted on the other hand that most relations are in the final version of the ontology on the higher levels of abstraction, whereby an ontology engineer continuing to refine the ontology could specialise those relations and create a more well-connected and well-defined ontology.

The evaluators were quite satisfied with the taxonomy of the initial version of the ontology. Although the average depth has somewhat decreased

in the final version, due to the addition of a large number of concepts, the taxonomy is still reasonable and would provide a nice starting point for further refinement. This conclusion is also supported by the fact that the depth actually increased slightly between the initial ontology and the intermediate version, using the same set of patterns.

The taxonomy of the final version of the automatically constructed ontology certainly provides a lot of different perspectives. This is both a benefit and a drawback, since the perspectives are not clearly separated and defined, instead perspectives are mixed and manifested as redundancies or alternative modelling choices presented together in the ontology. The benefit is that an ontology engineer refining the ontology has a lot of suggestions how to model, the problem is how to choose the right ones. For such tasks an appropriate user interface is a key feature, and tools to analyse consequences of the choices, but both these services are so far outside the scope of *OntoCase*. With respect to axioms more research is needed, no improvements has been made in the area of matching and selecting general axioms to be included in the resulting ontology in the second iteration of our research. In *OntoCase* axioms from both patterns and extracted from text are simply included as is, without further analysis.

8.2.4 Summary and discussion

In summary we can from this second run of the SEMCO experiment see clear improvements in the *OntoCase* method. Primarily the improvements relate to the ability to include more abstract knowledge in the form of more abstract patterns. This is due to the more elaborate matching methods used. Additionally we can note the increased coverage of the terms and relations extracted from text, which was both due to the improved method and the extended pattern catalogue. This is a very valuable improvement since these terms represent the core knowledge to be represented in the ontology, the domain specific knowledge. These improvements are valuable but it is also important to note that the method has, despite all changes and additions, not introduced any more structural errors, i.e. as defined in the *OntoClean* framework, into the new version of the ontology, even though the final version of the ontology is considerably larger than the initial one.

This is however based on not viewing the modelling choices present in parallel as errors, but precisely as choices for further refinement. Since the intention is only to produce an initial ontology, which will be refined in the *OntoCase* revise phase or further developed by an ontology engineer,

redundancies and modelling alternatives should not necessarily be viewed as errors but instead as suggestions to be considered and verified by a user. Intuitively the reader can agree that providing some guidance and choices to the user is most likely better than to select one alternative automatically, whereas wrong choices may sometimes be made. Also the results from the initial SEMCO experiment showed that the domain experts appreciated a dense structure of relationships in order to interpret and evaluate the concepts of an ontology.

8.3 JIBSNet - the JIBS enterprise ontology

The second evaluation was performed in a different domain, namely the university domain. JIBSNet is an intranet present at Jönköping International Business School (JIBS). The intranet contains internal documents of all kinds, from personnel instructions to meeting minutes and information on ongoing projects. The intranet is additionally used to provide internal information to JIBS' students on all levels. JIBSNet is suffering from similar information logistical problems as many other systems where large amounts of information is being stored. Information is hard to find, people store and annotate information according to many different schemas and viewpoints, and retrieval of documents is done solely based on classical file structure browsing or keyword search. An enterprise ontology representing the context of JIBS and the organisation, activities and processed at JIBS could help to improve the classification, presentation and retrieval of information from JIBSNet.

The intention of constructing an ontology for JIBSNet is to provide a structure from which to extend content and with which to annotate and reason about existing content. The aim is to improve the usability, understandability and effectiveness of JIBSNet. No complex reasoning is required, instead the focus should initially be on providing a formal structure to do simple categorical classification and reasoning to assess similarity and relevance of information with respect to contexts and queries. An example is that a document annotated with the concepts 'instruction' and 'professor' should be retrieved when searching for instructions for researchers, since 'professor' should be modelled as a kind of 'researcher'.

8.3.1 Ontology construction

An initial version of the JIBSNet ontology was constructed using a text corpus harvested from JIBSNet's internal documents in English and in addition all English language web pages from the JIBS public web[†]. The corpus consisted of 207 individual documents. The documents were of varying sizes, but the majority were short texts, scraped from public web pages for instance. To exclude the possibility of introducing a bias by using an old version of Text2Onto, as for the SEMCO case, the initial text processing and term and relations extraction was this time performed externally by the developers of Text2Onto, using the latest version of the software and experimentally optimised settings of variables and thresholds. This input representation contained 6535 terms organised in a shallow taxonomy with a maximum depth of three, but most classes directly beneath the root, and connected by 218 other properties.

Based on this input the OntoCase method was applied, matching the extended pattern catalogue to the set of terms and relations, and subsequently reusing the patterns to construct an initial ontology. The pruning option was used, which means that only the terms and relations from the input that could be somehow connected to the selected patterns were included in the constructed ontology. In this case 30 patterns were selected for inclusion in the ontology, and the resulting ontology contains 2576 concepts.

8.3.2 Evaluation setup

The evaluation was performed in a similar way as for the SEMCO case, consisting of three parts; collecting general structural characteristics, structurally evaluating the taxonomy and functionally evaluating the ontology through assessments of domain experts. Since this ontology is considerably larger than the SEMCO ontologies, a randomly selected subset of the concepts and relations was used as a sample of the content of the ontology and only this sample was evaluated. For the taxonomic evaluation a random selection of 100 concepts from each ontology were included. The selection was made through randomly selecting concepts and including all their superconcepts, until the level of 100 concepts was reached.

The evaluation by domain experts intended to assess the same issues as in the part of the OntoMetric framework used in the SEMCO ontology evaluation, but due to the fact that this time a sample of the ontology concepts

[†]<http://www.ihh.hj.se/eng/>

and relations was used the assessments cannot be performed directly by the domain experts, as assessments selected on a scale. For example, it is not reasonable for a domain expert to assess the amount of essential concepts in the complete JIBSNet ontology, and not even for the sample it is an easy task. Therefore another method was applied in this case, letting the domain experts assess single concepts and relations, and from the judgement of several domain experts, in agreement, we tried to deduce more general assessments of the ontology.

Six domain experts representing different roles, i.e. university teachers, PhD students, senior researchers, administrators and IT-support personnel, in the organisation were used for the evaluation, thereby covering the broad spectrum of interests different groups in the organisation have with respect to JIBSNet information. Randomly selected parts of the ontology were then represented in simple concept graphs and shown to the experts, asking them to assess the relevance and correctness of the displayed concepts and relations. For each assessment five choices were given; 'essential', 'accept', 'not sure', 'not correctly modelled', and 'incorrect or not appropriate'.

'Essential' meaning that the evaluator believed that the concept or relation was highly relevant for internal information on JIBSNet and that it was correctly modelled in its current setting in the ontology. 'Accept' indicating a concept or relation that was valid within the organisation but where the relevance was lower with respect to internal information. 'Not sure' indicating that the meaning of the concept or relation was not really intuitive to the evaluator, or that the evaluator was not certain how relevant or correctly modelled the concept or relation was. 'Not correctly modelled' stating that the concept or relation was somehow relevant, or at least valid, but that it was not modelled in a correct way. This assessment was used when the evaluators found something they believed was placed incorrectly in the taxonomy, although still relevant, or when a concept should be remodelled with a slightly different name. Finally, 'incorrect or not appropriate' was used for concepts or relations that the evaluators would like to delete from the ontology, since they believed they were incorrect or simply not relevant with respect to the JIBSNet scenario.

Two evaluators assessed all subsets of concepts and relations in the random sample, thereby the results were based on agreements or disagreements among the evaluators and represented both consensual views, but also the views of different roles in the organisation. The evaluators were asked to talk out loud when making their judgements, and to motivate their choices. Aside from motivations, the domain experts were also asked to provide com-

Table 8.4: Number of concepts and relations collected in the sample.

Evaluation set	No of concepts	No of relations
Input ontology (top structure)	94	63
Input ontology (total)	138	101
Output ontology (top structure)	62	62
Output ontology (total)	236	249

ments on the concept graph, e.g. noting how they would like to solve issues they discovered, and what they thought was missing in the ontology.

In order to be able to compare the results of OntoCase to typical OL tools, such as Text2Onto, the initial input ontology generated by the developers of Text2Onto was assessed in the same way. Using a random sample and the same interpretations of the evaluators' assessments, this ontology was evaluated with the domain experts. The top structure of the ontology was a problematic issue in the initial SEMCO case, and was showed to have improved since then when revisiting the SEMCO case, therefore also a special evaluation was made on the top structures of both the JIBSNet ontology and the input ontology created by Text2Onto. Table 8.4 shows the size of the random samples collected from each ontology. The reason for not using exactly the same sample as for the taxonomic evaluation was that in this case the 'context', meaning the surrounding relations and concepts to a certain depth, of each concept was included, in order to support the understandability by the domain experts.

8.3.3 Evaluation results and analysis

In this section the results of the above presented evaluation setting are described and analysed in detail.

General measures

The collected data from the basic structural measures can be seen in Table 8.5, where 'input ontology' denotes the initial ontology extracted from the text corpus using Text2Onto and 'output ontology' denotes the ontology constructed based on the input ontology using OntoCase. The set of measures is the same compared to the previously described evaluation of the SEMCO ontologies.

The input ontology is shallow and contains very few taxonomic relations compared to its size, only 167 of the 6535 concepts are *not* on the top level

Table 8.5: General characteristics of JIBSNet ontology.

Characteristic	Input ontology	Output ontology
Number of concepts	6535	2576
Number of non-taxonomic properties	218	147
Number of root concepts	6368	15
Number of leaf concepts	6399	2527
Avg depth of inheritance	1.03	2.72
Avg number of related concepts	0.067	0.11
Avg number of subclasses	0.029	1.83

of the taxonomical hierarchy of the ontology. The input ontology contained 218 named properties extracted from the text corpus, this is quite a low number for such a large ontology. In contrast, the output ontology has a quite general top-structure with only 15 concepts directly beneath the root concept of the taxonomy. It is also considerably smaller than the input ontology, only about 39% of the input size, since the pruning version of the composition algorithm was used in OntoCase. This additionally means that it covers about 39% of the input terms, since only 13 of the added pattern concepts are not found as synonyms to any extracted terms. If we again consider the amount of 'incorrect' concepts in the input, 9.2% of the concepts were deemed incorrect by the evaluators as noted later in the evaluation by domain experts, we can reduce the number of reasonable terms in the input to 5933 and arrive at a coverage of about 43%. The relation coverage, of non taxonomical relations, is 55%.

When analysed, some reasons for this low coverage of input ontology terms can be noted, e.g. the amount of proper names and abbreviations in the ontology and some Swedish terms that were present in the input. 137 concept labels contain the Swedish letter ä, 41 the Swedish letter ö and 34 the Swedish letter å. Also, it should be noted that in the SEMCO experiment the catalogue had 15 patterns specific for the product development domain, while in this case there were no patterns specifically for the domains of research and university education, although the more general patterns still apply. This indicates the importance of having a larger pattern catalogue and more domain specific patterns, but it also shows that we are actually able to achieve some reasonable results without these domain specific patterns, only using the most general and domain independent patterns.

Taxonomic evaluation

The analysis of the taxonomy was performed in two steps, applying two different methods. Below the results of these evaluations are described and discussed, together with some examples of problems or modelling errors that were discovered. For both these evaluations sets of 100 concepts connected into a small taxonomy, randomly extracted from each ontology, were used to perform the evaluation.

When applying the general taxonomic evaluation proposed by Gómez-Pérez [84] some interesting observations could be made also in this case. The input ontology contains some circularities and some multiple inheritance, but the output ontology is much more 'tangled', just as in the SEMCO case, i.e. it contains numerous cases of multiple inheritance and redundant relations. The reason for the increased 'tangledness' is primarily the merging of different kinds of evidence that is performed during the pattern composition and ontology construction in OntoCase, e.g. evidence for both synonymy and hyponymy might be present. We arrive at the same conclusion as in the previous evaluation, since this is an initial ontology to be further developed, either manually or by some other automatic methods, it is more useful to the user, i.e. the ontology engineer, to include all hypotheses than to discard some of them.

The method for taxonomic evaluation first focuses on circularity errors. Multiple inheritance was present in both ontologies, as well as circularities. These are however very few in the input ontology constructed by Text2Onto, and the multiple inheritance is mainly due to that some concepts are explicitly stated to be subclasses of owl:Thing as well as of another concept. In the output ontology constructed by OntoCase a few circularities could be found, but mainly on the level of distance 0. This means that a concept is stated to be the subclass of itself. The reason for these errors is that we do not filter the hypotheses, i.e. a term that is found to be a synonym of a pattern term might also be found to be a subclass of it. As noted previously this is an easy error to prevent if needed, a simple filter could be applied during the construction process, but again we note that this also represents a choice to be made by the ontology engineer and as such it may be a useful information. One circularity on the level of distance 1 is also found, where 'situation' is stated to be a 'position' but also the opposite, 'position' being a kind of 'situation'.

Partition errors is the second category of errors considered, but since neither ontology contains any instances we can only consider the structure

of classes for this evaluation. The input ontology does not contain any disjointness axioms, since such algorithms were not included when it was constructed using Text2Onto. In the output ontology however there are a few places in the ontology where there exist common classes in disjoint partitions. In fact, the exact same issue occurs this time as in the SEMCO ontology, where the concept 'part' is both a subclass of 'object' and 'role' although they are disjoint. This issue originates in the composition of two patterns, whereas it is not so unexpected that it might occur in several ontologies. Nevertheless, this is a questions of choosing how to model parts in the ontology, but since the issue originates from the pattern composition, it might be possible to provide additional support for resolving it already at composition time. Following what was previously stated, the input ontology does not contain any complete decompositions but there is one complete decomposition defined in the output ontology; again it is 'entity' with the complete partition 'abstract', 'quality', 'event' and 'object'. Just as in the SEMCO case there are a number of concepts that are subclasses of two or more of these general concepts, it is a matter of presenting the redundant findings to the user instead of resolving conflicting evidence automatically.

Next, the semantic errors of the ontologies were studied. In the output ontology 8 semantic errors were found among the 100 concepts studied, these are all due to invalid classifications of extracted concepts. The discovery of semantic errors was again done based only on domain knowledge of the enterprise in question, not on any task requirements for the ontology. The majority of these errors are due to that too general knowledge has been added, by using background knowledge such as WordNet, which leads to several modelling alternatives are present but some are then not correct in this domain.

With respect to incomplete concept classifications these might be present in both ontologies, but since this is closely related to the task of the ontology it was not possible to discover these errors without specifying the task requirements in more detail. The same problem arose with the partition errors, what knowledge is required to be present in the ontology has to be determined using a detailed task description. Some comments from the domain experts, in the following domain expert evaluations, are related to these issues however. Some domain experts pointed out areas where knowledge was missing or incomplete.

Checking redundancy in the two ontologies was the last step. There were no direct repetitions of subclass relations in either ontology, but as we have noted several times before there are many cases of redundant sub-

Table 8.6: Result of the OntoClean evaluation of the JIBSNet ontologies.

OntoClean rule	No of cases Input ontology (T2O)	No of cases OntoCase
Incompatible identity criteria	3	2
Incompatible unity criteria	8	4
Unity/anti-unity conflicts	7	1
Rigidity/anti-rigidity conflicts	10	5

class statements in the output ontology, i.e. there exist quite a few indirect repetitions. As also stated previously, many of these can be viewed as suggestions for further refining the model, or alternatives for making modelling decisions, hence we believe that it is in fact not desirable to remove all these 'errors' automatically, unless a completely correct automatic selection can be guaranteed. There were no identical explicit definitions of any classes, but in the input ontology some concepts were in fact represented by terms that are synonyms and also a few cases of synonyms were not detected by OntoCase, and hence not corrected in the output ontology. An example is that 'endeavour' and 'endeavor' were modelled as two different concepts, although they are obviously only two spellings of a term representing the same concept.

When annotating the set of 100 concepts from the initial ontology constructed by Text2Onto using the OntoClean properties, a backbone taxonomy of 26 concepts was identified. 7 Concepts were not annotated, due to the fact that they could not be interpreted in any reasonable way, and were probably errors introduced by Text2Onto. Next, the ontology constructed through OntoCase was treated in the same way, annotating the set of 100 randomly selected concepts with the OntoClean properties. Only two concepts were left untreated, due to that they could not be interpreted clearly enough. The results of studying the rigidity, identity and unity criteria of the remaining 93 and 98 concepts, in each ontology respectively, can be seen in Table 8.6.

There were a few cases of each problem type present in the input ontology constructed by Text2Onto, but it should also be noted that several of the cases in the different categories actually refer to the same pair of concepts. An incorrectly modelled taxonomic relation might give rise to several of the conflicts at the same time. The concept 'business school' being modelled as a subclass of 'area' both violates the compatibility of identity criteria and the compatibility of unity criteria. A business school

is an organisation being identified by an organisation number while an area can be identified by its extent in terms of location coordinates. A business school is an organisation where the parts are joined together with the abstract notion of being organised within the same unit, while an area is a whole through the proximity of its coordinates.

Other examples of issues discovered in the input ontology are several cases where the subclass relation was used in the opposite direction, compared to intuition. 'Event' was modelled as a subclass of 'inauguration', while one would normally expect the opposite. Several cases where subclass relations were used instead of partonomy could also be noted, such as where 'faculty' was modelled as a subclass of 'industry professor'. The direction of the relation should be the opposite, but additionally the relation should be 'part of' and not subclass.

When considering the output ontology, constructed using OntoCase, there are fewer errors in all categories. However, errors exist and one example is the 'agent' concept which is used both as a kind of actor, a person or an artificial agent, and a chemical agent, such as medicines and drugs. This gave rise to both incompatible identity and unity criteria. Drugs can be identified by their chemical composition or their medical names, while an agent in the sense of a person would be identified by his personal identification number or his DNA. A medicine seen in a concrete sense may be a whole, the physical instance of a drug, and the connecting relation would be the chemical bindings connecting the compounds in the drug. A person might also be viewed as a whole and could in theory be seen as a set of molecules, it is not obvious however that it is chemical bindings that connect the parts of a person, more intuitively it is the physical presence of body parts within one body. The rigidity and anti-rigidity criteria was violated mostly in cases where a concrete concept, such as the technical concept of a 'bus' was a subclass of a general abstract notion, such as 'configuration'. In the case of 'bus' it was again a case of using the subclass relation instead of a more appropriate relation such as 'involved in', to state that choosing a specific hardware bus might be a part of the configuration of a hardware and software system.

Evaluation by domain experts

When collecting the results from the domain expert assessments some interpretations had to be made in order to arrive at results giving an indication of the overall evaluation criteria concerning the complete ontology. Results

were interpreted in different ways for concepts and relations, due to the way the subjects used the assessment alternatives when judging concepts and relations. The subjects noted that in their opinion 'not correctly modelled' chosen for a relation usually meant that the relation was wrong, it should be replaced with some other relation. When chosen for a concept, 'not correctly modelled' usually meant that the concept in itself was relevant in some context, but the relations and concepts surrounding it did not fit. A concept assessed as 'not correctly modelled' might thereby be either essential or at least acceptable but still assessed to be wrongly modelled if the surrounding relations and concepts were incorrect or inappropriate.

The interpretations used for concepts and relations were based on the set of assessments from the evaluators. Table 8.7 shows the interpretations when combining two evaluator assessments, similar interpretations were used when three or more evaluators assessed the same concepts and relations. The difference of interpretation between concepts and relations when considering the 'not correctly modelled' alternative, as mentioned above, should be noted specifically, but additionally the special treatment of the 'essential' assessment. It was not expected that all evaluators would agree since they represent different roles in the organisation. As long as some role believed that a concept or relation was essential this had to be taken seriously, even if somebody representing another role might be more sceptical. Only if an assessment as 'essential' was combined with the 'incorrect' assessment the total interpretation was set as a disagreement.

Table 8.8 presents the results of the evaluation of concepts and taxonomic relations. The results are shown as a percentage of the total number of concepts or relations. The 'essential' and 'accept' assessments were added under the heading 'correct', while disagreements and incorrect results are defined as described above. Not surprisingly the input ontology has a high number of correct concepts, regardless whether we are studying only the top structure or the complete ontology. This is due to that all the concepts are derived directly from the input text corpus, which means that the concepts are terms collected from the internal documents of JIBS and merely filtered for relevance. When studying the taxonomic relations of the input ontology the results are not encouraging, around half of the relations are deemed incorrect by the evaluators. This is mainly due to two issues, one is the fact that relations are harder to correctly extract from texts than terms and concepts, and thereby many relations on all levels are actually incorrect, but also due to that the evaluators assess almost all relations to the top concept (owl:Thing) as incorrect. The latter issue can be interpreted as

Table 8.7: Interpretation of assessment combinations.

Interpretation of concept assessments	
Interpretation	Assessment combination
Essential	Agreement on essential Essential and acceptable Essential combined with not sure Essential combined with not correctly modelled
Accepted	Agreement on accept Accept combined with not sure Accept combined with not correctly modelled Agreement on not correctly modelled Not correctly modelled combined with not sure
Disagreement	Agreement on not sure Essential and incorrect Accept and incorrect Not sure and incorrect Not correctly modelled and incorrect
Incorrect	Agreement on incorrect
Interpretation of relation assessments	
Interpretation	Assessment combination
Essential	Agreement on essential Essential and acceptable Essential combined with not sure Essential combined with not correctly modelled
Accepted	Agreement on accept Accept combined with not sure
Disagreement	Agreement on not sure Essential and incorrect Accept and incorrect Accept combined with not correctly modelled
Incorrect	Agreement on not correctly modelled Not correctly modelled combined with not sure Agreement on incorrect Not sure and incorrect Not correctly modelled and incorrect

the evaluators disagreeing with the concepts being modelled as direct sub-concepts of the top concept. This was confirmed by the subjects, voluntary by 'talking out loud' and when explicitly asked for an explanation during the evaluation. This indicates that the evaluators really do miss a more abstract structure and division of the concepts into categories.

The OntoCase output ontology has a slightly lower percentage of correct concepts, both on the top level and overall, and instead a slightly larger number of disagreements. This indicates that some problems arise when interpreting concepts in the ontology constructed by OntoCase. Such results are quite natural, considering that in this case concepts were added based on patterns that were matched, whereby the concepts have only an indirect connection to the actual terminology used at JIBS and are in some cases not easy to interpret appropriately. The top concepts are sometimes very

Table 8.8: Interpretation results.

Evaluation set	Assessment	% of concepts	% of relations
Input ontology (top structure)	Correct	85.1%	33.4%
	Disagreement	11.7%	17.5%
	Incorrect	3.2%	49.2%
Input ontology (total)	Correct	75.3%	26.8%
	Disagreement	15.9%	15.8%
	Incorrect	9.2%	57.4%
Output ontology (top structure)	Correct	80.6%	58.1%
	Disagreement	16.1%	38.7%
	Incorrect	3.2%	3.2%
Output ontology (total)	Correct	73.7%	53.4%
	Disagreement	16.1%	24.9%
	Incorrect	10.2%	21.7%

general, these are harder to interpret than more specific concepts. In favour of the OntoCase method we note that the number of incorrect concept only increase very slightly for the complete ontology, and is stable when considering the top structure. As mentioned previously the evaluators missed a general top structure when studying the input ontology, in the output ontology this had been added and thereby also 'pushed' some incorrect concepts further down the taxonomic hierarchy, fortunately without adding incorrect concepts on the top level.

When considering the taxonomic relations we can note a considerable improvement in the percentage of correct relations, both on the top level and overall. There was some disagreement around the interpretation of relations introduced by OntoCase, but at least the number of incorrect relations decreased drastically. Very general relations can be hard to interpret and it was hard to judge whether they were really essential to the ontology or not. However, even in the hypothetical case that all disagreements should in reality have been classified as incorrect relations we would have been able to note a considerable decrease in the amount of incorrect relations compared to the input ontology. This is also partly due to the fact that a more intuitive top structure had been introduced, whereby relations leading directly to a top concept were no longer perceived as incorrect to the same extent. There are other improvements as well, many of the new relations introduced by the patterns are actually found to be essential, a conclusion additionally supported by comments made by the evaluators during evaluation.

Non-taxonomical relations were assessed separately, and the results can be seen in Table 8.9. At the bottom of the table the added pattern relations in the output ontology are presented separately, in order to show that these were actually mostly correct according to the evaluators. There was even

Table 8.9: Interpretation results concerning non-taxonomic relations.

Evaluation set	Assessment	% of relations
Input ontology	Correct	50.7%
	Disagreement	16.2%
	Incorrect	33.1%
Output ontology	Correct	65.0%
	Disagreement	11.9%
	Incorrect	23.1%
Relations added from patterns	Essential	88,9%
	Incorrect	11.1%
Relations added from input ontology	Correct	59.5%
	Disagreement	14.7%
	Incorrect	25.9%

a stronger agreement between the evaluators within this set of relations, whereby no acceptable relations or disagreements were found, hence only presenting the amount of essential and incorrect concepts. The selected relations from the input ontology are presented separately. We note that there is an increase in correct relations between the input and output ontologies and a decrease in incorrect relations, while the disagreements are kept on a relatively stable level, a small decrease can be noted. The results when separating between added pattern relations and relations selected from the input ontology indicate that when pruning some of the relations and concepts from the input ontology we actually manage to select a larger amount of correct relations rather than the incorrect ones. The amount of correct relations thereby improved compared to the input ontology.

The domain experts were asked to talk 'out loud' during their evaluation, and to give comments with respect to missing items and unexpected modelling choices. Mostly the comments were pointing out missing relations, since the evaluators were only shown the taxonomy some of the relations were in fact present just not shown to them, and missing concepts. For example, pointing out that 'informatics' is not the only department of JIBS, and asking "Where are the others?". Similarly only 'civilekonomprogrammet', an educational program given at the economics department, was present, while other educational programs were missing. This is a flaw in the results, although it could be observed that the evaluators easily listed the rest of the missing concepts, such as the rest of the programs, when given one example in the ontology. The evaluators also noted that dif-

ferent views were mixed in the ontology, and requested some intermediate structure in some cases, e.g. to divide the concept 'design' into different categories depending on what the design was intended for, design of systems or design of educational programs, before adding the subclasses that were at the moment direct subclasses of design.

Missing relations were mostly non-taxonomical relations, the evaluators commented on numerous occasions that "this concept has to do with that one", indicating some kind of relation. In some cases however the relation was a missing subclass relation, this occurred particularly often where two different patterns had been included in the ontology but without proper composition. An example was the composition of the employee, party and organisation patterns. In the employee pattern we can see that employees are employed within a department, and when combined with parties and organisations the employees are also classified as persons. The pattern concept department was however not considered in the composition process, and since it did not have any prior matches to organisations or other included concepts the connection was not identified at this stage. Similar problems arose for synonyms, it was perceived that some reasonably easy cases of synonymy should be detected by the system, i.e. differences in spelling of terms such as 'organisation' and 'organization'.

At the end of the session the evaluators that had evaluated both top structures of the two ontologies, 4 out of 6 of the evaluators did both tasks, were asked which one of the two top structures they would prefer to base a further refinement of the information structure for JIBSNet on [‡]. Three out of four answered in clear favour of the ontology produced by OntoCase and motivated it by the generality and understandability of the structure, that it covered most of the general areas that were needed, and that it contained less mistakes than the other one. The fourth evaluator was unsure and explained that although the structure of the ontology produced by OntoCase was much simpler and more clear, the evaluator also liked some specific modelling solutions in the other one and would like to combine parts of both.

[‡]Exact wording of the interview question was: "When considering these two structures and your recent evaluation of them, if you had to choose one, which one would you choose to continue to refine and why?" The evaluators had no prior knowledge about the ontologies, such as how they had been constructed, and by what method.

8.3.4 Summary and discussion

The JIBSNet ontology was not yet applied in a real-world setting, since no ontology-based ILOG system is present for structuring and retrieval of information on JIBSNet. Despite this we performed a rigorous evaluation of the constructed ontology, both with respect to structural characteristics, correctness of the taxonomy, and functional measures, i.e. correctness and understandability as viewed by domain experts. One main aim of this evaluation was to compare OntoCase to a typical state of the art system for ontology learning. Text2Onto was chosen for this task, since it is one of the few systems that work almost completely automatically, it is used by the community, and it has been shown to perform well. For this system we were able to let the developers of Text2Onto themselves perform the ontology construction, in order to be certain not to introduce any bias in the process.

The results show that with respect to the concepts of the ontologies themselves, OntoCase perform on the same level of accuracy as Text2Onto, while with respect to the structure OntoCase considerably improves on the initial input. The top structure added gives a more intuitive structure to the ontology and the relations are deemed correct to a larger extent than the relations of the input ontology. The same issues with respect to the taxonomy can be noted in this evaluation as for the SEMCO evaluation. The tangledness is also in this case present throughout the ontology and some redundancy can be noted as well. It is important to point out that this might actually be an advantage if used in the right way, providing some options for an ontology engineer when continuing to refine the ontology. When applying OntoClean a difference in the number of errors can be noted, between the input ontology and the OntoCase result. The fact that there are less errors after applying the patterns is mainly due to that 'strange' and unclear concepts are pruned when applying the patterns, instead the ontology is enriched with relations originating in the patterns and from the matching process.

8.4 FAO - agricultural ontologies

The third evaluation performed using OntoCase was set in the agriculture domain. The two previous evaluations clearly show the merits and characteristics of OntoCase when applied within the core focus of the method, namely enterprise ontology construction for ILOG applications. It is im-

portant to note however that OntoCase is actually more general than that. Depending on what patterns are available, and what input can be provided, OntoCase could be used for almost any domain. There are limitations, such as the use of WordNet in the matching process which would degrade the performance of the pattern matching if the domain was too specific. Trying to apply OntoCase on certain types of biomedical ontologies, for example would probably yield less good results, since the common sense style of matching between patterns and extracted terms and relations might not be applicable in this domain, provided only general background knowledge, i.e. WordNet. Nevertheless, there are many domains in addition to enterprise ontologies for structuring of information, where light weight ontologies of reasonable quality can be of great value, either as a starting point for further refinement or in their own right. One such domain is the agriculture domain. The intention of this experiment is mainly to display the applicability of OntoCase to other domains and point at the benefits of an interplay between OntoCase and other ontology learning techniques.

The setting of this experiment is the Food and Agriculture Organisation (FAO) of the United Nations and their work on improving the use of agricultural resources around the globe. In the context of assisting countries in improving their agriculture, forestry and fishing practises the organisation monitors, structures, and provides information relevant to the agriculture domain. To facilitate these tasks numerous thesauri, taxonomies and other linguistic resources are used. Currently the organisation is trying to improve their processes by moving from simple structures, such as a thesaurus of terms, to more complex definitions of concepts, such as an ontology. This is for example investigated in the context of the use-cases in the research project NeOn[§]. This experiment was conducted as a part of the NeOn project, through the cooperation project DEON[¶], although separately from the actual case studies of the project. The experiment was additionally reported in the NeOn Deliverable 3.8.2 [211].

The ontologies to be constructed were set within the agriculture domain. The FAO have numerous taxonomies and light weight ontologies constructed by reengineering of different sources, but these are quite simple and also sometimes error prone, whereas the intention was to further develop these into full fledged ontologies and correct some errors that were present. For this experiment one such existing light weight ontology was

[§]<http://www.neon-project.org>

[¶]Development and Evolution of Ontologies for Networked Organizations, financed by the Swedish Foundation for International Cooperation in Research and Higher Education

used as a starting point. Ontologies were also constructed directly from text corpora and textual concept definitions, and the results of OntoCase were compared to the input and the results of the ontology learning tool Text2Onto. Text2Onto is, as mentioned previously, a state of the art OL tool that was during this thesis previously used in order to represent the level of a 'typical' current OL system.

Two hypotheses were stated in order to be tested in this experiment. The hypotheses were the following:

1. Patterns can improve the structure of learnt ontologies, even in another domain than enterprise ontologies.
2. Enrichment of ontologies, by adding concepts and relations using existing OL methods, can improve the pattern matching in OntoCase.

The first hypothesis states that applying OntoCase on top of results from other OL methods, such as the ones in Text2Onto, will improve those results, for example by connecting unconnected parts, such as unconnected concepts, and give the ontology a more general top structure by adding missing general background knowledge. The structure has to be added without introducing errors into the ontology, whereas it is important to assess the correctness of the ontologies both before and after applying OntoCase. This is the same focus as for the two previous experiments presented in this chapter, although the domain is different. The second hypothesis proposes that applying other OL methods, such as the ones in Text2Onto, as a pre-processing step in order to enrich the input ontology before applying OntoCase can actually assist the pattern matching, so that more relevant patterns can be identified and included. The general aim is to show that OL methods can benefit from each other, and that combining and iterating different methods can provide better results than applying the individual methods.

8.4.1 Ontology construction

The data used for the experiments was provided by the FAO and was set within the agriculture domain. A simple lightweight ontology was provided, generated and translated into OWL from a manually engineered graphical concept network representation. The ontology was focused on concepts related to the concept of 'rice'. This ontology is here denoted *Ontology_rice*, and some information about the ontology can be seen in Table 8.10. Three

additional ontologies were produced using the Text2Onto tool based on texts provided by FAO. Out of the three text corpora, two were generated by extracting DBpedia abstracts and comments respectively, collected by using the concepts in the Ontology_rice as search terms. DBpedia^{||} is a semantic version of Wikipedia^{**}. It contains a large amount of annotated data, but not everything is reliable since it is based on the wiki idea of community created content. The third text corpora was produced by FAO based on article abstracts connected to the AGROVOC thesaurus^{††}, related to the term 'rice'. Additional information about the ontologies generated from these text corpora using Text2Onto can be seen in Table 8.10. They are named T2O_DBPabstracts, T2O_DBPcomments, and T2O_AgrovocAbs respectively. Finally, one manually engineered translation of a part of the AGROVOC thesaurus connected to rice concepts were also provided, here denoted Thesaurus_rice, although it is in fact an OWL ontology.

	No. of concepts	No. of top concepts	No. of subclass relations	No. of properties	Avg. depth
Ontology_rice	266	155	110	37	1.68
T2O_DBPabstracts	1086	1018	89	17	1.06
T2O_DBPcomments	365	290	189	3	1.34
T2O_AgrovocAbs	3575	1822	1954	49	1.68
Thesaurus_rice	525	4	542	138	3.60

Table 8.10: The experiment input ontologies.

For the second part of the experiment the Ontology_rice ontology was then enriched with additional concepts and relations produced by Text2Onto, using the two text corpora extracted from DBpedia. Two new ontologies were thereby constructed, named OntologyRice_EnrichedAbs and OntologyRice_EnrichedComm. Details about these can be seen in Table 8.11.

	No. of concepts	No. of top concepts	No. of subclass relations	No. of properties	Avg. depth
OntologyRice_EnrichedAbs	1256	1080	199	53	1.20
OntologyRice_EnrichedComm	535	352	299	39	1.57

Table 8.11: The enriched ontologies for the second part of the experiment.

Next, OntoCase was run with all these ontologies as input, in the pruning mode, meaning that only the parts matching any pattern were included in the output. The reason in this case was to minimise the evaluation

^{||}<http://dbpedia.org/>

^{**}<http://www.wikipedia.org/>

^{††}<http://www.fao.org/agrovoc/>

effort. The pattern ranking threshold was set to 0.08 (level experimentally determined). The extended catalogue of 41 ontology patterns were used, containing all content patterns that were at the time available, and correctly represented, on the ontology design pattern portal* and in addition a set of patterns previously used with OntoCase, see section 8.1. The ontologies resulting from applying OntoCase to all the above presented ontologies can be seen in Table 8.12 together with their structural characteristics.

	No. of concepts	No. of top concepts	No. of subclass relations	No. of properties	Avg. depth
OC.Ontology_rice	280	112	245	46	2.19
OC.T2O.DBPabstracts	812	22	1471	28	2.37
OC.T2O.DBPcomments	321	22	628	16	2.39
OC.T2O.AgrovocAbs	2823	27	4162	63	2.84
OC.Thesaurus_rice	344	6	377	27	4.50
OC.OntologyRice.EnrichedAbs	1293	758	1679	73	3.36
OC.OntologyRice.EnrichedComm	570	248	921	62	3.06

Table 8.12: The experiment output ontologies.

8.4.2 Evaluation setup

After producing the ontologies, we proceeded by evaluating the ontologies. First, some general structural characteristics were gathered for each ontology, see tables above and results in the following section. This aimed at providing an idea of the structure of the ontologies, e.g. indicating the number of top concepts and average depth of the taxonomy. The sizes of the ontologies is an interesting feature that shows that these are in fact not toy examples. After this collection of structural characteristics, the taxonomical correctness was evaluated, in order to show that the added concepts and relations had not affected the correctness negatively.

Since some of the ontologies were quite large and the evaluations had to be performed manually we used a random sample of concepts and relations for the evaluations, and not the complete ontologies. This introduces some uncertainty into the results, but in this case we aim to show the general trends and that the correctness is not affected negatively, rather than the exact amount of improvement, hence this uncertainty can be accepted. Due to time and resource restrictions the ontologies were only evaluated by one person (ontology expert), which also introduces uncertainty. Although this is reduced by the fact that the same method and the same judgements were

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

applied to all ontologies, whereas the relations between the error rates of the input and the output should stay the same. In summary, the values presented should not be taken too seriously as absolute numbers, but can be seen as very reliable as a group of results showing an overall trend when comparing the input and output of OntoCase.

Additionally, since the evaluator was an ontology expert and not a domain expert, i.e. not from the agriculture domain, the evaluation had to be based on textual sources of agriculture information. These sources were primarily the AGROVOC thesaurus, of agricultural terms and their relations, and secondarily agriculture information available on the web. However, there were cases when only a domain expert could have evaluated the correctness properly, in these cases the evaluator had the choice of marking the concept or relation with an 'I'm not sure' annotation. The other two alternatives were to annotate the concept or relation with 'correct' or 'not correct', within the scope of a domain ontology in the agriculture domain. The sample size for each ontology was between 104 and 180 for the randomly selected concepts, and between 51 and 102 for the relations. The relations were a mix of both subclass statements and general properties. The reason for the differences in sample size was that the random selection process was slightly dependent on the size of the ontology.

8.4.3 Evaluation results and analysis

In the sections below the results of the evaluations are presented. The first part is related to the first hypothesis stated previously, and subsequently the second hypothesis is treated.

Improving learnt ontologies

The ontologies resulting from running OntoCase on all the above presented ontologies can be seen in Table 8.12 above, together with their characteristics. Without going to details of the ontologies it can be noted that for most ontologies the number of top concepts is reduced and the average depth is increased, this is due to the added top structure provided by the quite general ontology design patterns that were matched and reused by OntoCase. This is how OntoCase attempts to add some of the missing general background knowledge that is explicit in the input texts, and thus not included in the learnt input ontologies.

Additionally, the number of subclass relations increased for most ontologies, even though the input ontologies were pruned of some of their concepts and thus the output ontologies were generally smaller. This is due to that OntoCase adds all 'hypotheses' found in the pattern matching process, i.e. all correspondences between pattern concepts and input concepts that may be interpreted as subclass relations. This results in a certain amount of redundancy and can even result in cycles in the taxonomy, as exemplified in section 7.7. At a first glance these may be considered as errors, but it is actually an intended feature of OntoCase, providing the ontology engineer with suggestions for alternative modelling choices. However, the user interaction is not properly supported by a graphical user interface at the moment. The ontologies were also enriched with more properties from the patterns, which can be noted in the properties column of Table 8.12.

The next step was to evaluate the ontologies with respect to correctness, using the method stated in the previous section. The results of this evaluation can be seen in Table 8.13. From the results in the table we can conclude that the correctness is generally not affected negatively by applying OntoCase. Only in one single case is the correctness of concepts slightly lower for the output ontology, the case of T2O_DBPcomments. However, this specific result is quite inconclusive since the amount of incorrect concepts actually decreased, and it was the amount of concepts that the evaluator was uncertain of that increased for the output ontology. This could thereby be an effect of the random selection of concepts. Nevertheless, the trend is clearly an increased correctness of the ontologies after applying OntoCase. The increase of concept correctness is quite small, while the increase of relation correctness is considerable.

The increase in correctness is however mainly a side-effect of applying the method rather than a direct feature, since OntoCase does not attempt to explicitly filter out 'irrelevant' parts. Rather, the results originate from the pattern matching, since when concepts are clearly incorrect, i.e. represented by misspelled terms and strangely combined multi-word terms, they will not match anything in a pattern and will therefore be pruned. In some domains, if the terms are very specific, some additional domain specific background knowledge might be needed for the pattern matching in order not to prune too many domain specific concepts and thereby reduce the ontology quality. At the moment, only domain independent background knowledge was used for the matching, whereby these results are encouraging, showing that even in a quite specific domain like agriculture the method and the general patterns produce a reasonable result for most cases.

Ontology	Concepts Properties	Correct %	Not sure %	Incorrect %
Ontology_rice	C	92.8	5.4	1.8
	P	90.0	5.7	4.3
OC_Ontology_rice	C	97.8	2.2	0.0
	P	92.8	2.9	4.4
T2O_DBPabstracts	C	85.5	3.4	11.1
	P	61.6	12.8	25.6
OC_T2O_DBPabstracts	C	87.9	4.6	7.6
	P	77.5	2.5	20.0
T2O_DBPcomments	C	94.2	2.9	2.9
	P	64.1	11.5	24.4
OC_T2O_DBPcomments	C	93.0	4.4	2.6
	P	86.1	6.3	7.6
T2O_AgrovocAbs	C	84.9	5.7	9.4
	P	65.9	13.2	20.9
OC_T2O_AgrovocAbs	C	88.5	5.8	5.8
	P	79.3	6.9	13.8
OntologyRice_EnrichedAbs	C	87.8	5.6	6.7
	P	76.2	10.7	13.1
OC_OntologyRice_EnrichedAbs	C	88.1	5.6	6.4
	P	79.1	8.1	12.8
OntologyRice_EnrichedComm	C	92.0	4.3	3.7
	P	78.4	4.9	16.7
OC_OntologyRice_EnrichedComm	C	93.6	3.6	2.7
	P	88.2	5.9	5.9

Table 8.13: The evaluation results of the ontologies.

The observant reader may have note that one pair of ontologies is missing from Table 8.13, namely *Thesaurus_rice* and *OC_Thesaurus_rice*. During this experiment it was determined more or less impossible to evaluate the *OC_Thesaurus_rice* in the above described manner, and thereby this pair was not included in the evaluation. Instead we provide a short discussion about why this was the case and in what way this ontology was not possible to use with the current version of *OntoCase*. Inherent in *OntoCase* are some common principles of ontological modelling, such as the common practice that concepts are given humanly understandable names, or at least labels in natural language.

The ontology *Thesaurus_rice* was, as mentioned previously, in fact an ontology, in the sense that it was represented in OWL and used some of the features in OWL, but it was in all other respects more of a thesaurus. The transformation had been done more or less like a 'direct translation', whereby the structure was not a common ontological structure. Almost none of the concepts in the ontology have natural language names or labels, but are instead encoded using a number, such as *c.3259*. The ontology provides a very general top structure which is basically a metamodel of a thesaurus, containing concepts such as lexicalization, term, noun, category, and domain

concept. There is a taxonomy and properties specialising this structure, but all actual domain knowledge, concepts and terms, are instances of the concepts in this structure. Even the concept instances are connected to their lexicalization only through special purpose properties.

Some of the general metamodel concepts were recognised by *OntoCase*, and matched general pattern concepts, thereby some kind of result was actually produced by *OntoCase* after the pattern reuse phase. This was not a comprehensive result however, since at the moment *OntoCase* does not treat instances, whereas the part where the information of the input ontology actually resides was completely ignored by the method. In the pruning mode, *OntoCase* even prunes all the instances of the ontology, whereby the output was complete nonsense since all concepts had been stripped of their lexicalizations, which in this case was the only form of concept definitions existing. Due to this fact, the output ontology was just a set of numbered concepts, in a logical structure without meaning, hence it was impossible to evaluate the correctness of this structure. This provided a good example of a type of ontology that *OntoCase*, in its current version, cannot handle.

Enrichment as support for pattern matching

To find support for the second hypothesis the pattern matching results from running *OntoCase* on the original ontology, called *Ontology_rice* above, and the two ontologies that were enriched by the results from *Text2Onto* were recorded. The general characteristics of the input ontology *Ontology_rice* were already presented in Table 8.10 and the characteristics of the enriched ontologies in Table 8.11. Results after running *OntoCase* were shown in Table 8.12, see above.

The intention in this case was to show that enrichment using other OL methods would ensure that more patterns are correctly identified to match the input ontology, without introducing errors. In this case errors could be both incorrect concepts and relations in the ontology, as discussed previously, or an inappropriate pattern that was matched and included. For the first type of errors we have already seen in the last section, see Table 8.13, that the enrichment in itself will in fact introduce some errors, compare the correctness results for *Ontology_Rice* and *OntologyRice_EnrichedAbs* in Table 8.13. The OL methods used are not exact, and the original ontology was in this case manually engineered and thereby quite accurate in its definitions. This introduces a trade-off, whether it can be worth extending the ontology or not. We are not able to answer this question in general

here, we merely show that there are also some benefits when applying OntoCase on top of these enriched ontologies compared to applying it on the non-enriched ontology.

The results in terms of the number of selected patterns can be viewed in Table 8.14. 19 patterns out of the catalogue of 41 were selected for inclusion by OntoCase based on the `Ontology_rice` ontology. This number was increased to 24 and 23, for the two enriched ontologies, respectively. We can also note that patterns are not only added, see + column, some are also no longer selected, see the - column. This is due to that the enrichment also slightly changes the focus of the ontologies, whereas some patterns can then be considered more relevant while some are considered less relevant.

Ontology	No. of	
	selected patterns	- +
<code>Ontology_rice</code>	19	
<code>OntologyRice.EnrichedAbs</code>	24	-2 +7
<code>OntologyRice.EnrichedComm</code>	23	-3 +7

Table 8.14: The number of patterns selected.

To show that the change in pattern selection is reasonable, we also studied the patterns that were selected in more detail. It is generally hard to say if a pattern is valid or not for a specific domain or case at hand, rather one has to study what parts of the pattern that were actually used in the ontology. There can be concepts and properties from a pattern that are valid in a certain domain, even though the complete pattern is not appropriate. In this case many patterns in the pattern catalogue are quite general and can be appropriate to include in any domain, whereas the task of deciding if a pattern was correctly selected is more connected to if the matches found are indeed correct. In Table 8.15 below, the patterns selected for each of the ontologies are listed.

The patterns were assessed in a similar way as the correctness of concepts and properties previously, assigned either 'complete match', 'incorrect' or 'partial match' by an ontology engineer. 'Complete match' denoting that the complete pattern fitted the domain and case at hand, 'partial match' meaning that there were some parts of the pattern that could be used for this domain and case at hand, and finally 'incorrect' meaning that this pattern did not fit the domain or case at hand. The task was described as to judge if the pattern had been correctly or incorrectly selected, based on if any part of the pattern could be applicable in the domain, rather than to evaluate

Ontology_rice	OntologyRice_EnrichedAbs	OntologyRice_EnrichedComm
Actions.owl	Actions.owl	Actions.owl
AgentRole.owl	AgentRole.owl	AgentRole.owl
Classification.owl	Classification.owl	Classification.owl
CollectionEntity.owl	CollectionEntity.owl	CollectionEntity.owl
Constituency.owl	Constituency.owl	Constituency.owl
CoParticipation.owl	CoParticipation.owl	CoParticipation.owl
Description.owl	Description.owl	Description.owl
GOtop.owl	GOtop.owl	GOtop.owl
ObjectRole.owl	ObjectRole.owl	ObjectRole.owl
Participation.owl	Participation.owl	Participation.owl
Person.owl	Person.owl	Person.owl
Precedence.owl	Precedence.owl	Precedence.owl
Product.owl	Product.owl	Product.owl
SpeciesModel.owl	SpeciesModel.owl	SpeciesModel.owl
System.owl	System.owl	System.owl
TypesOfEntities.owl	TypesOfEntities.owl	TypesOfEntities.owl
Organisation.owl	Organisation.owl	EmployeeDepartment.owl
ProductAssociations.owl	EmployeeDepartment.owl	Metonymy.owl
ProductCategory.owl	Metonymy.owl	NaryParticipation.owl
	NaryParticipation.owl	Party.owl
	Party.owl	Situation.owl
	Situation.owl	SystemSynthesis.owl
	SystemSynthesis.owl	TaskRole.owl
	TaskRole.owl	

Table 8.15: The patterns selected.

all the actual matching results. Missing patterns were not considered. In Table 8.16 the number of patterns in each category can be seen for each ontology.

Ontology	Complete match	Partial match	Incorrect
Ontology_rice	15 (79%)	4	0
OntologyRice_EnrichedAbs	20 (83%)	4	0
OntologyRice_EnrichedComm	19 (83%)	4	0

Table 8.16: The number of completely and partially applicable patterns.

The four patterns considered to only partially match the domain of *Ontology_rice* are the *Person*, *Product*, *ProductAssociations*, and *ProductCategory* patterns. The *Person*-pattern was not completely suitable since it is more intended for ontologies about people and the information associated to individuals, i.e. social security numbers, age, height, weight. The other three patterns are domain specific patterns from the product development domain. However, some general parts of these patterns, treating products and their features could also be applicable in the agriculture domain, agriculture does in fact produce some types of products. Two of these, the *ProductAssociations* and *ProductCategory* patterns, were removed for the *OntologyRice_EnrichedAbs* and *OntologyRice_EnrichedComm* but instead two other patterns deemed as only partially suitable for the domain, i.e. the

EmployeeDepartment and SystemSynthesis patterns were introduced. No completely unsuitable pattern was included in any ontology. Nevertheless, this means that 5 completely appropriate patterns were added into each of the enriched ontologies, patterns that were not possible to identify and include before the enrichment of the input ontology.

8.4.4 Summary and discussion

To summarise these result we can conclude that OntoCase in fact provides an added structure to the ontologies, and does connect unconnected parts of the ontologies produced by other OL methods. A general top structure is introduced, adding some of the missing general background knowledge not found explicitly in the input texts. These improvements are achieved without increasing the error rate of the ontologies, hence the first hypothesis is supported.

Although this was a completely different domain and focus, than for the two first evaluation experiments, and there were no domain specific patterns available, the results are reasonable. OntoCase was able to connect large parts of the ontologies to the patterns, even with this very small pattern catalogue. It has to be noted though, that the ontologies produced have still to be viewed only as a basis for further development, hence they contain for example multiple modelling choices from which the user may choose the needed ones.

Based on the results presented above it can also be noted that enrichment of ontologies, whether learnt or handcrafted, can lead to the selection of more relevant patterns. The enrichment in itself might introduce some errors compared to manually constructed ontologies, but this is natural since enrichment is done automatically. However, the error rate did not increase by applying OntoCase on top of the results, and the additional patterns selected were to a high extent correct. The overall conclusion is clear, the second hypothesis is supported, enrichment supports matching of patterns. This is an additional merit of OntoCase, that it is suited for interaction with other OL methods, not only in a purely sequential manner where output from one method becomes input of the other, but in a more iterative fashion. Nevertheless, no conclusion can be drawn with respect to the trade-off between the errors introduced when enriching the ontologies and the benefits of being able to include more relevant patterns.

Chapter 9

Discussion and future work

This chapter reflects on the results presented in the previous chapters. The first section revisits the evaluation questions, regarding research method and presentation, that were proposed in chapter 4. This discussion puts the research in perspective and comments on the method and the research process itself. Then the chapter continues with a discussion of the results themselves, and especially future work potential. Most results, although supportive of the OntoCase method, also indicate additional improvements that can be attained, such open issues and future work will be noted and discussed throughout this chapter.

9.1 Research evaluation

In chapter 4 a method for evaluating the research approach as such was discussed, and some questions were posed in order to be answered at the end of this research. After finishing the evaluation of the actual results it is now time to briefly also evaluate the methods applied in the research. The questions were divided into five categories that will be treated below:

1. Significance of the study
2. Internal validity
3. External validity
4. Objectivity and confirmability
5. Reliability and auditability

9.1.1 Significance

Significance concerns the relevance and the contribution of the research, as well as the quality of the research results. To determine the significance of the research results the following questions were posed:

- Who are the beneficiaries of these results and what relevance do these research results have to the beneficiaries' problems?
- What are the main research contributions of this research to the field?
- How is this solution better than previously proposed solutions to the same kind of problems?

With respect to the first question it may be noted that this research is relevant mainly to ontology engineers and ontology researchers. However, ontology engineers are a growing category of people, in contrast to when ontologies were used for expert systems during the 90's, today ontologies are constructed and presented on the web by different categories of people. Thereby the term 'ontology engineer' can include almost any person who constructs an ontology, just for fun or for some specific purpose, whether in their spare time or as a part of a research or development enterprise. This is also precisely why approaches such as the one presented in this thesis are most relevant, if inexperienced ontology engineers or domain experts are to be able to construct reasonable ontologies they need help. Tools and automatic methods can provide this help, and patterns provide a way to reuse best practices and well-established knowledge. These are crucial facilitators for the development of ontology-based applications on a larger scale, whereby this research is highly relevant.

The second question addresses the main research contributions. These are additionally stated and discussed in chapter 10, but in brief the main novel research contributions are the following:

- Patterns
 - The typology of ontology patterns.
 - Characteristics and descriptions of the pattern types.
 - Catalogue of content design patterns.
 - Experiments showing the usefulness of patterns.
- Semi-automatic ontology construction

- The overall framework for pattern-based semi-automatic ontology construction, called OntoCase.
 - A method for pattern ranking and selection.
 - A method for pattern composition.
 - Implementation of the above, facilitating experimentation on the methods.
- Evaluations
 - Evaluations of pattern-based semi-automatic ontology construction.
 - Experiences from ontology construction with industry partners.
 - Experiences from adapting and using ontology evaluation measures and methods.

The focus of the third question is on how these results improve the results achieved with previously existing methods. This issue was addressed in chapter 8, mainly during the evaluation of the JIBSNet ontology where results were compared to typical results of an existing OL system. Additionally in the SEMCO experiments reported, we noted that relation elicitation was hard to do even manually, whereby the patterns proved to provide many good suggestions and added some structure even in comparison with a manually engineered ontology. The interplay between existing OL systems and OntoCase was discussed in the third evaluation case, in the context of the FAO agricultural ontologies. It seems that combining existing OL techniques with the use of patterns gives some promising results, and this aspect has not been explored in any other research effort so far.

9.1.2 Internal validity

Internal validity is concerned with the evidence that supports the research conclusions and the credibility of the arguments made. Questions asked with respect to internal validity are:

- Does the method actually solve the problem? Completely or only partly?
- Does the method meet all stated requirements?
- Is the method compared to alternative methods, and are results for such comparisons presented?

- Is negative evidence sought for and presented?
- Does the evidence support the claims for the research results sufficiently?

The first question can be answered with both yes and no. The very general problem of automatically constructing ontologies is far from solved, but this was not the intention with the thesis. With respect to the specific research questions stated, the implemented parts of the OntoCase method, in combination with the pattern typology and the pattern catalogue, does really provide one possible solution to that specific problem. Resulting ontologies are not perfect but as noted in the last section, they are indeed an improvement over ontologies constructed by comparable existing methods. If we consider the research questions as indicating the overall requirements of the method, the solution does meet all the requirements, e.g. it produces ontologies of better quality than existing methods. Nevertheless, only a part of the OntoCase method is treated in detail, so the result can be seen as a partial solution to the problem.

The method is compared to alternative methods, although one weakness is that during the experimentation phase the method is only compared to one specific alternative system. This is due to several practical reasons, e.g. the availability of alternate tools and details of other methods, and theoretical reasons such as fairness. Since all the methods for OL are semi-automatic in some way, the user needs to provide some minimum amount of input, such as setting variables or choosing input data sets, which makes comparisons hard. The tools usually require user input in different ways, and most require some user validation throughout the construction process. Setting up an experiment that fairly compares supporting tools for ontology construction when all the tools require different kinds of input and decision-making from the users, is a very hard problem. The comparison easily becomes a comparison of user abilities rather than tool capacity. To avoid this we have chosen to select one typical tool, the one that is most similar to our own approach, requiring a minimum of user intervention. We then mainly show that our method can improve the results of this tool, which in turn can be viewed as producing typical results from state of the art OL algorithms today. These comparisons and a brief comparison to a manually engineered ontology was presented previously in chapter 8.

There is no description of a complete failure of the method, since we have no such evidence at all, as far as concerns the presence of negative evidence. Nevertheless, there are results that show that even though the

method does improve on results produced by other systems it still produces some errors, i.e. the method is not perfect and has its own benefits and drawbacks. Negative evidence is provided in the evaluation results in the last chapter, as well as in the discussion of open issues and future work in the following sections and chapter 10. One specific example of an ontology that could not be handled by OntoCase was provided in section 8.4. The positive evidence presented comes from four larger experiments, including the first SEMCO experiment, and some minor examples and studies, within three different domains. We believe that this is sufficient evidence for the benefits and drawbacks of the proposed method, and enables us to draw the conclusion that OntoCase really improves existing methods.

9.1.3 External validity

External validity is connected to the generalisability of the results, in the sense that if external validity is low then the results only hold for the specific cases tested. Questions asked to confirm external validity are:

- Is the method based on existing theories? Does it confirm and support those theories?
- Are assumptions, personal opinions and biases clearly stated and analysed?
- Are the procedures and methods used clearly described?
- Are there limitations to the evidence collected thus suggesting limited generalisability?

The work of this thesis is firmly based on theories and empirical evidence from many existing research efforts. A thorough state of the art survey is presented in chapter 2 and chapter 3. In these chapters, for example, existing methods for OL are discussed and presented. Using these theories and methods as a basis, our method provides another layer that improves these results. The algorithms proposed to solve the concrete problems are, in many cases, based on or at least inspired by, well established theories and methods, although considerably adapted to this specific field and the specific problem at hand. The OntoCase approach confirms the hypothesis underlying all OL systems, that it is possible to construct some parts of an ontology automatically starting from existing knowledge sources. Additionally it builds on the notion of patterns and shows that ontology patterns can also be useful in semi-automatic ontology construction.

As far as possible, personal opinions are kept outside this thesis, although judgements and conclusions drawn are always to some extent based on personal opinions. During the method development, ideas and approaches were developed based partly on personal ideas and 'hunches' of how to solve problems, but all these were then confirmed through the implementation and the empirical evidence provided in chapter 8. Any assumptions made are clearly stated throughout this thesis, whether these are assumptions of specific ontology languages being used or assumptions of available input to the method. The aim is to describe the method and the experiments in such detail that any procedure or method can easily be scrutinised for biases or any implicit assumptions still remaining.

The generalisability of the typology of patterns is very high. This is a general theoretical result that is valid for the entire ontology engineering field without restrictions, although the patterns themselves might sometimes be domain-dependent. Such a typology of patterns can also be applied to other fields where patterns are used, although this is beyond the scope of this thesis. Even though we have chosen to restrict our claim of applicability to the construction of enterprise ontologies, the final experiment within the agriculture domain shows that OntoCase and even existing patterns might actually be applicable to a much broader field than the one explored. There is no empirical evidence or theoretical considerations that prevent OntoCase from being applied to ontology engineering in general. A current limitation is the low complexity of the ontologies constructed, since there are few methods for extracting, matching or composing general ontological restrictions and axioms. This restricts the applicability of the method slightly. Although there is nothing that prevents us from using the method on complex extracted ontologies, the only issue is that these features cannot currently be matched to the patterns, but may be included in the ontology as is.

9.1.4 Objectivity and reliability

The final two criteria can be combined into one set of questions, actually already partly covered by the previous ones. Questions regarding objectivity, reliability, confirmability, and auditability are:

- Are the research questions clear?
- Are basic constructs and basic assumptions clearly specified?
- Are all methods, experiments and procedures described in detail?

- Is it clear what data is used for testing and experiments?
- Are all assumptions made explicit during the process?
- What biases exist for the persons and methods involved in the research?
- Is the status and role of the researcher explicitly described?
- What limitations are imposed by biases or subjectivity in the research results and conclusions?

The research questions are clearly stated and available in section 1.2.4, they are discussed throughout the thesis and subsequently answered in the conclusions chapter. Basic constructs and assumptions, such as ontology definition and details about OL methods, are presented in chapters 2 and 3, and additional assumptions are discussed in the context of the method implementation and the evaluations in chapter 8. Detailed descriptions are provided for each method proposed, and the exact setup of each experiment is described in detail. The only details missing are the proprietary company specific data and information that was used for constructing the ontologies, as well as the complete content of the ontologies constructed. Some of this data unfortunately cannot be published publicly due to property rights issues and non-disclosure agreements. Other data was excluded due to space restrictions in this thesis, but can be retrieved from the author on request.

Assumptions during the process are explicitly stated, or at least derivable from detailed descriptions of the proposed methods and the experiments. Biases are naturally present in all experimentation settings that involve both humans and methods developed by humans, this is unavoidable. The effects of such biases can be minimised through careful engineering of experimentation settings and selection of subjects and data sets. By conducting several experiments in different domains, with different settings and different domain experts involved, we have shown that biases are indeed not the cause of the positive results achieved.

Biases are also closely connected to the role of the researcher in the experiments, since taking a too active part in the experiments could introduce a bias. Running the automatic methods was a task of the researcher in the experiments presented above, except for the external tool (Text2Onto) that was run by the tool's developers themselves. Input data was also selected by the researcher, although in almost every case all available information was used, such as the complete set of English texts available at JIBSNet, or the

complete set of available pattern candidates from the ODP portal, hence no subjective decision-making was involved. Some results were evaluated by the researcher, such as the collection of structural characteristics, and the application of OntoClean and other taxonomic evaluations. This is of course a weakness, but due to practical limitations and the availability of alternative personnel with ontology engineering experience, there were no other options available. To compensate for this a very important part of the evaluations were the evaluations conducted by domain experts, where external evaluators assessed the results. The researcher was also involved in these experiments, but strictly refrained from interfering other than to present the task and the setting.

For practical reasons there were some limitations on the selection of data and evaluation methods. It is important to be aware of these limitations, as discussed above, but by presenting several examples and evaluations that all agree and point at certain general conclusions, sufficient rigour is achieved. The exact numbers presented in each evaluation setting should not be viewed as an absolute objective truth, but the trends are nevertheless clear and unmistakable. The level of detail in the descriptions additionally makes it possible to conduct similar experiments and confirm these results if needed.

9.2 Ontology patterns

One of the main contributions provided in this thesis is the typology of patterns and the initial pattern catalogue that was developed. Below, the implications of having such a typology as a frame of reference are discussed, and the possible benefits and drawbacks of different kinds of patterns are noted.

9.2.1 Benefits of ontology patterns

The typology of patterns presented in chapter 5 contains four levels of abstraction; the application level, the architecture level, the design level, and the syntactic level. It is easy to see that these levels cover all possible levels of abstraction when constructing ontologies. The levels of granularity for each abstraction level are quite intuitive. This framework of pattern levels, i.e. a typology, is beneficial in several ways. As noted in chapter 3 patterns are commonly stated to have three kinds of benefits:

- reuse,
- guidance,
- and communication benefits.

Reuse benefits mainly concern constructing better artefacts, in our case ontologies, due to the reuse of some best practices, in the form of patterns. At the design pattern level the reuse benefits are many, although the production of formal evidence has only begun, e.g. see the experiment reported in section 5.3.4 and the evaluations in chapter 8. On the syntactic and design level, patterns are ontological elements, pieces of ontologies or restrictions on the overall architecture, whether described for a specific representation or on an abstract logical level. Since such patterns are usually provided as implemented building blocks, that can be directly reused, it is easy to see that the reuse benefits are stronger in ontology engineering than in for example software engineering, where reusable components are rarely shipped together with patterns. Design patterns in software engineering are usually just abstract guidelines for how to solve a certain problem, where ontology design patterns in addition provide the implemented solution.

Reuse benefits for architecture and application patterns cannot immediately be accepted, without further study, since many architecture patterns or application patterns can be quite abstract. It is not trivial to see how applying an ontology architecture pattern would always improve the quality of the ontology. Since researchers are today still asking what is a good architecture, there is no clear answer to whether or not ontology architecture patterns could provide better ontologies. Similarly, as far as concerns application patterns it is not clear that such patterns would necessarily make the ontology 'better' in some sense. Nevertheless, these patterns are valuable in other respects, as we shall see later.

Guidance benefits are given by patterns that offer rich problem descriptions, point at difficult issues and provide suggestions for solutions. Sometimes the actual solution of the pattern is not the most valuable thing. Equally valuable is for example the problem description itself, which can help developers to avoid common mistakes and notice issues that would have otherwise gone unnoticed. When ontology design patterns are reused, even as ready-made building blocks, they also provide guidance as to how things should be modelled. Then the developer is to extend and adapt the pattern to fit the case at hand. These benefits are likely to be the most important, for all patterns, especially for inexperienced ontology engineers.

Although experienced ontology engineers may find some patterns trivial, the patterns can be used as checklist, ensuring that nothing important has been overlooked. Some research is even proposing to use design patterns for evaluating ontologies, since they are encoded best practices and can guide ontology evaluation as well as ontology construction.

Communication benefits have been discussed and promoted in software engineering. Communication benefits can be seen both when teaching a craft, such as ontology engineering, when performing it, when discussing it between developers, and when documenting the artefacts and the processes. Patterns provide a common vocabulary to talk about difficult problems and their solutions. If two developers both know ontology design patterns, then they have a much easier and more compact way of expressing a solution to the *n*-ary relation problem by saying that you should use the 'situation pattern' than by describing the individual concepts and relations that are needed. Additionally this is the aim of anti-patterns, describing 'bad practices' and showing common mistakes, in order to communicate those in a standard way. Anti-patterns can be viewed as a vocabulary for talking about problems and common mistakes.

When used automatically content design patterns need to be formally represented and machine processable, this increases the chance of achieving reuse benefits, since the patterns are really used as building blocks, automatically composed into an ontology. How good the resulting ontology will actually be is highly dependent on other factors as well, therefore it is not automatically true that by reusing patterns we achieve a high quality, as we have seen when evaluating *OntoCase* in previous chapters. Guidance benefits are not currently present when patterns are reused automatically. However, if architecture patterns were present, these could be used to guide the automatic composition of the ontology, rather than reusing them directly, and thereby provide guidance benefits of a sort. Communication benefits are not provided by the patterns used in *OntoCase* at the moment, since no detailed provenance information or other documentation is produced by the system. In the future however it would be desirable to provide a way of tracking what patterns were used to construct what parts of the ontologies, and how the patterns were adapted and composed. Such information would increase human trust in such a system and would help when manually changing or refining the ontology. In such a case patterns would provide communication benefits, being a means for communicating the solutions provided by an automatic method.

In addition to the ontology patterns discussed in this thesis, where the focus has been on ontology patterns in the sense 'patterns for ontology construction', there can be a need for other patterns concerning ontologies, but not directly related to their construction. Re-engineering patterns have for example been proposed by other researchers, as well as reasoning patterns to describe common services provided by reasoning engines and logical languages. Although all such patterns are valuable, we believe that there is need for a more coherent and informative terminology with respect to ontology patterns, which is one intention behind the typology proposed in this thesis. Today, too many things are called patterns, this makes it confusing for both developers, researchers and when teaching ontology engineering. We believe that it would be a great benefit to the community if a common terminology could be used and if the term pattern was used a bit more restrictively.

9.2.2 Pattern construction

At the moment, patterns are constructed more or less ad-hoc, sometimes without proper grounding in best practices or existing solutions. Many of the current patterns presented in this thesis have been reengineered from different kinds of knowledge sources. Even if those sources are patterns in themselves there is no proof, neither theoretical nor empirical, that the resulting ontology pattern candidates are really appropriate patterns for the field. Generally, patterns should represent some consensus within a community, whereby a process for pattern review and promotion can be beneficial. Such a process is applied in the ODP portal* where the portal community is responsible for reviewing and proposing patterns. The patterns used and produced in this thesis are all candidates in the sense that they have not yet passed this process of peer review. Other than development by reengineering most patterns today are constructed through extracting self-contained pieces of top-level ontologies, which result in very abstract and high-level patterns. Sometimes less rigorous methods have to be used however, for bootstrapping a first set of patterns for some particular domain. As discussed in chapter 3 the initiation of a design by/for reuse process is a crucial phase, where patterns need to be produced without any initial reward for this extra effort.

In the context of this research we have taken a very pragmatic approach to patterns that slightly diverges from the stricter process mentioned above.

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

We believe that being a pattern is simply a role of an artefact, e.g. a small ontology. Something becomes a pattern when it is reused in a new setting, i.e. when it takes the role of 'pattern'. This is very close to the notion of reusable stored cases in the context of CBR. We have thereby chosen not to consequently call all our applied patterns 'pattern candidates' but instead have shown that they are indeed reusable and, thereby, are patterns. The restriction that a pattern should also represent best practices is somewhat relaxed in this thesis, and it is not altogether clear what a best practice is. For instance, a pattern representing a 'work around', solving something that cannot be represented in a certain type of logic, might not be a pattern, since the best practice might be to choose another type of logic for representing the ontology. Nevertheless, this might be a very useful pattern in practise, if the choice of representation is not made by the ontology developer.

In the OntoCase framework the retain phase that closes the development cycle is devoted to pattern extraction. The phase as such is suggested based on the inspiration from CBR, and for the detailed realisation of this phase we mainly propose to take inspiration from approaches to ontology modularisation and approaches from graph theory, e.g. algorithms for finding strongly connected graph components. Finding coherent parts of the ontology that might constitute suggestions for new patterns could be one way of evolving the pattern catalogue, while the OntoCase method is used. We envision some user involvement in this step, validating and possibly generalising the candidates before including them in the pattern base. Additionally, a feedback process for existing patterns is needed, one that supports the user in making pattern changes or updates. Purely manual pattern construction methods are beyond the scope of OntoCase, but manually constructed patterns are very valuable and can of course be stored and used.

9.2.3 Ontology content design patterns

The focus of this thesis is on ontology content design patterns for use in semi-automatic ontology construction. The requirements on such patterns are to provide an implemented building block ready to be reused, and to either represent some best practices or common solutions. This thesis does not bother with the description and presentation of patterns. Although an important issue when patterns are intended for human reuse, it is not equally important when reusing patterns semi-automatically. Nevertheless, even for automatic or semi-automatic use additional information about the patterns can be valuable, such as the domain of the pattern or the compati-

bility with other patterns. To take such additional information into account when matching, selecting and composing patterns could improve the results of the OntoCase retrieval and reuse phases.

An important prerequisite for applying OntoCase is that a sufficient number of patterns are present. Even though it was shown in the last chapter that the implementation of OntoCase performs reasonably well even with no patterns specific to the domain at hand, as in the JIBSNet case, it would most likely perform even better with additional patterns tailored to that domain. Domain specific patterns, like any other patterns, can be developed either by specialising some existing domain independent patterns or by developing new patterns based on solutions within the domain. Using domain specific pattern catalogues would provide more specialised ontologies, with fewer errors due to the generality of the patterns. Future work include to specialise general patterns and develop catalogues specific to enterprise ontology domains.

Other useful information are compatibility, and other relations between patterns. At the moment OntoCase takes a quite naive approach to compatibility, assuming that if nothing is said about incompatibility there is none and it is left to the ontology developer, or evaluator, to resolve any inconsistencies or other issues introduced in the ontologies. Composition is done only through pattern overlap, but could in the future also benefit from using other relations between patterns. For example, it was noted throughout the evaluations that some patterns could actually have been connected, although since the matching methods were only applied between extracted elements and the patterns, and not between patterns, such connections were not added. Patterns could be matched 'off line' and the matchings validated by an ontology engineer, in order to obtain a preexisting set of connections between the patterns in a catalogue, to then be used at runtime.

9.2.4 Ontology architecture patterns

Very few ontology architecture patterns have been proposed, and they are usually very general, such as the architecture styles layering or modularization. At the moment the OntoCase implementation does not make use of any architecture styles or patterns. To improve the pattern reuse phase further, and to support pattern composition, some formal guidance would be desirable. Such guidance could be provided by an ontology architecture pattern, or even more specifically by an ontology reference architecture. From our point of view, a reference architecture displays the use of one or more

architecture patterns but is more specific and tailored to a specific domain. A reference architecture for enterprise ontologies, or even more specifically enterprise ontologies of a certain type of enterprise, could provide a general structure where design patterns could fit together like pieces of a puzzle.

9.3 OntoCase

Many benefits of the OntoCase method were described in the previous chapter, and supported by experimental results. Nevertheless, only parts of the framework have been studied in detail and implemented so there is great potential for future improvements and for covering the complete OntoCase cycle. First, we will discuss the retrieval and reuse phases, their benefits and drawbacks, and state some future work possibilities. Then the potential of the revision phase will be explored and ideas for its realisation are presented. Retaining patterns, and issues related to this task, were mentioned in the discussion of pattern construction above.

9.3.1 Pattern retrieval

The pattern retrieval phase can be divided into two steps, i.e. input processing and pattern matching and selection. Input processing is done by using existing ontology learning methods, such as algorithms for term extraction and relation extraction. The assumptions made in this research are that the input to the method is a text corpus and that, at a minimum, the input processing method provides a set of terms and unnamed binary relations, with confidence values, as output. This is the typical output of OL systems existing today, although most systems additionally provide some more advanced features, possibly requiring some human intervention. Starting from these basic elements is an obvious choice, but in order to improve the overall results further improved methods are also needed for this step, such as methods for axiom extraction and generation of complex concept definitions.

In the current implementation of OntoCase an alternative to exploiting Text2Onto is provided, through the possibility to start from an existing OWL ontology. This gives the opportunity to use any OL tool able to produce an OWL ontology as the tool for input processing. Current research topics within the OL field include support for reengineering existing knowledge structures, such as databases, thesauri and community created tag sets. Another topic is the extraction of more complex and expressive ontologies,

where recent developments have provided methods to extract disjointness axioms and complex class definitions from dictionary definitions in natural language. The key challenge is then to make use of these more expressive elements for pattern matching and selection.

Other improvements for matching between patterns and extracted elements are to use more specific background knowledge. At the moment the complete WordNet dictionary is used as background knowledge in the matching process, but there exist methods to for example prune WordNet, selecting only a domain specific part of the dictionary. Other kinds of background knowledge could also be used, and hypotheses could be tested against knowledge present on the web. The 'asking Google' approach has been used successfully in many other contexts. If a hypothesis can be transformed into a natural language sentence that is submitted to the Google API as a query, the number of hits hint at the likeliness that the hypothesis is true. Also methods for word sense disambiguation could help improve the quality of the matching results.

Matching currently involves only content design patterns and elements extracted from a text corpus. One issue noted at the beginning of this thesis was the lack of focus that can be observed in current OL systems, there is no way to state formal requirements for the ontology and to restrict the extraction process. This is also true for the OntoCase method. The only way to restrict the process is to select a different input, in the form of a different set of patterns or a different set of input texts. Ideally the ontology developer using the OntoCase method would be allowed to input a set of requirements to guide the ontology development process, for example by specifying a set of competency questions. Automatically matching these competency questions to competency questions of the patterns would then give a new level of matching, above single elements of patterns and extracted elements. This is additionally a step in the direction towards a pure CBR methodology, where case descriptions are matched rather than the actual case solutions.

Another future work direction would be to include architecture patterns, possibly in the form of reference architectures in the matching process. Automatically matching and selecting reference architectures seems like a hard problem requiring a lot of research, so in a first stage this selection would probably have to be done manually. Nevertheless, when used to guide the design pattern reuse in the following phase, a general overall structure would have great benefits, whether expressed as constraints on the composition or as a frame structure where design patterns act as components.

9.3.2 Pattern reuse

The pattern reuse phase takes the set of elements extracted from the input text corpus, a set of patterns, and the matching results from the previous phase as input. An initial ontology is constructed from this input. Two different approaches are currently used, either the extracted elements are used as the basis and matched parts of the patterns are added, or the extracted input elements are reduced to the set that were matched to any pattern element. If the first approach is used the coverage of extracted input elements is complete. In this way nothing is lost compared to only using the OL method, without adding any patterns. The second approach has been used for most of the evaluations in this thesis, since by pruning the extracted 'ontology' there is more focus on what is added from the pattern catalogue, which allows us to evaluate things like the coverage of extracted terms.

The pattern composition done in this phase uses simple heuristics to combine the selected patterns, and the matching results are used to connect the patterns to the extracted elements. Things that are not included in the matching process, such as additional axioms, are generally included if the concepts or relations involved in their definitions are added to the ontology. The set of heuristics seem to work quite well, a reasonable ontology is produced as we could see in the evaluations in the last chapter, but still there is potential for improvement. The heuristics deal at the moment with simple constructs, such as adding a subclass of a subclass of a concept, as a direct subclass of the concept, even if the intermediate direct subclass was not matched. Future improvements could be to exploit the formal definitions and restrictions of the model, and to use an inference engine to find more connections, rather than to only treat the asserted model.

Another issue in pattern composition was noted earlier when discussing patterns and relations between patterns. Since no matching between patterns is currently performed, and no preexisting relations between patterns are used, the patterns are only treated as small ontologies that are added as is. Pattern overlap is used to connect patterns, but the existence of pattern overlap cannot be assumed in the general case.

The ontologies constructed by the current OntoCase method are very rich in structure and quite 'tangled' with respect to the number and structure of relations present. This is due to the number of hypotheses developed during the matching step in the retrieval phase, that are later added to the ontology mainly as relations. This could be viewed as making the ontology

very complex and hard to comprehend, but on the other hand this also gives a lot of options to the ontology developer for choosing the correct way of modelling some issue. It is also our firm belief that it is harder to spot a missing relation than to decide whether a suggested relation is correct or incorrect, and thereby to add more structure, more relations, to the ontology is beneficial, up to a certain limit of course. Similarly to the famous statement by Firth [66] guiding the development of many of today's NLP and information retrieval approaches based on collocations: "you shall know a word by the company it keeps", we would like to propose the statement "you shall know a concept by its relations to other concepts". This is something also noted by domain experts in the evaluations presented previously, relations are very important for the understanding of a concept. Then a future user interface for OntoCase should provide assistance to resolve the tangled sets of hypotheses, such support is not yet developed.

9.3.3 Ontology revision

The third phase of the method is not treated in detail in this thesis, this means that the complete study and realisation of this phase is still future work. The idea of this phase is to semi-automatically improve the ontology constructed through pattern reuse in the previous phase until it is ready to be applied in its intended setting. This would involve cleaning and pruning of the ontology, as was discussed previously, to remove incorrect alternatives added in the pattern composition. Additionally, this would involve semi-automatic evaluation of the ontology in order to determine what issues remain and what improvements and refinements can be performed. If future versions of OntoCase include the specification of requirements of the ontology, such as a set of competency questions, these evaluations can be more precise and concrete, but otherwise the method would have to rely on general quality measures, such as the ones used in chapter 8.

Based on the evaluation results the ontology should be refined and possibly extended. OntoCase can be used as an iterative process. If the evaluations show some lack of content or incorrect focus, another iteration may be run, providing the ontology as input but adding new patterns or additional elements extracted from a new text corpus. Apart from a complete rerun of the OntoCase method, some ontology enrichment methods or ontology population algorithms could also be applied in this phase. Revision would still be partly a manual process. Although many OL approaches aim at bringing ontology engineering into the grasp of domain experts, this is a

far off vision, whereby some parts of the revision would most likely be done manually by an ontology engineer. In this sense the method could be used as one tool among others, within a manual ontology engineering process, for example to provide the first draft of an ontology.

9.4 Summary of future work

To summarise this discussion the typology of patterns needs to be embraced by the ontology pattern community, it is a first step towards a common terminology for ontology patterns and a coherent way of dividing patterns for ontology engineering into categories. In the future, all these categories should be populated and used for ontology engineering, but at the moment only a few are actually used in practise. When patterns are considered in a strict best practices sense, there is need for processes and methods how to develop patterns and how to agree and promote pattern candidates to become 'real' patterns. We have taken a more pragmatic approach where pattern candidates can be produced by reengineering, or extracted from existing solutions, and directly reused, without necessarily providing any other proof of their status than that they were used to solve some real-world problem.

Future work concerning the OntoCase method include improvements on current methods, such as disambiguation of terms, and tailoring of both input and output more strongly to the domain. Pattern matching could include additional background knowledge, as well as requirements in the form of competency questions. Pattern composition might benefit from additionally matching patterns, and applying more advanced heuristics as well as background knowledge about the patterns in the composition process. Implementation-wise the next steps would be to provide a tailored user interface to the method, allowing the user to follow and understand the process steps, as well as providing support for utilising the different results when further refining the ontology.

Chapter 10

Conclusions

A set of conclusions can be drawn from the theoretical studies and the experiments performed. As far as concerns patterns, it can be noted that there are currently several kinds of patterns for ontologies. Ontology design patterns are already being used today when constructing ontologies, but more kinds of patterns are emerging and believed to be beneficial. We have structured and characterised different kinds of ontology engineering patterns, producing a coherent typology of patterns for ontology construction. Ontology engineering patterns can be described on four levels of abstraction, ranging from syntactic patterns, via design and architecture patterns, to application patterns. Patterns with differing scopes can be introduced on all these levels, i.e. patterns can have different levels of granularity. As the level of abstraction decreases several levels of granularity become possible. A unified and thoroughly described mental and theoretical model for ontology engineering patterns, with defined levels of abstraction and granularity, is a novelty and will assist ontology engineers and researchers when discussing, developing and using patterns.

There are challenges in semi-automatically constructing ontologies from sources such as text corpora, and one of the prime challenges is how to incorporate the 'missing' information that is not explicitly stated in domain specific texts. This is both common sense knowledge and domain knowledge that is assumed and not stated explicitly, due to assumptions by the information provider with respect to the intended audience. We have addressed this challenge by introducing ontology content design patterns into semi-automatic ontology construction, and we have proposed a general framework for pattern-based semi-automatic ontology construction

called OntoCase. More specifically, ontology content design patterns on the granularity level of ontology pieces of the ontology, solving some specific modelling problem, are used to assist ontology construction. Experiments have shown that the ontologies produced are reasonable with respect to their intended domain and topics, and that they have both certain benefits compared to manually constructed ontologies, and improve the quality of output compared to typical existing semi-automatic methods. This improved quality may come at the expense of coverage, but since the ontologies are intended to be further processed manually, or enriched automatically, quality and comprehensibility are deemed more important than coverage of the extracted information.

More specifically we have developed a pattern ranking scheme that is used for ranking patterns according to their relevance with respect to an extracted set of terms, and relations between those terms, with the intent to select a set of patterns for ontology construction. Experiments have shown the benefits of this scheme compared to existing ontology ranking approaches, and it was then used in the implementation of the OntoCase framework. Pattern composition is also a crucial issue, which is included and implemented into the current version of OntoCase. Benefits of the pattern composition method are the variety of alternative modelling choices that are presented to the user for further selection and refinement, the added level of abstraction of the produced ontology, i.e. the addition of a general top level, and the improved structure of the output ontology, in terms of added relations.

The following research questions were posed in section 1.2.4:

- What kinds of ontology patterns can be differentiated?
- How can ontology design patterns be used in semi-automatic ontology construction?
- How does ontology design pattern-usage in the ontology composition step affect the quality of the resulting ontologies?

The first question was addressed by proposing the pattern typology, characteristics, and pattern definitions in chapter 5. The second question was addressed by proposing the OntoCase framework, and specific methods for realising the details of the first and second phases, such as the pattern ranking scheme. The OntoCase framework was treated in detail in chapter 7. The third and final question was addressed through a set of experiments using the implemented OntoCase method and the quality measures discussed

in chapter 4. The experimental results were presented in chapter 8, and an illustrative example can also be found in section 7.7.

To summarise, we have shown that four different kinds of ontology patterns can be differentiated, and are needed, for ontology construction. A pattern catalogue was developed that contains a set of ontology content design patterns, reengineered from other pattern sources. The general framework, OntoCase, presents a method for applying patterns in semi-automatic ontology construction, and the proof of concept implementation demonstrates the feasibility of the method. In order to study the characteristics and quality of the produced ontologies, to determine the appropriateness of the method, a set of experiments were performed in several different settings. Results clearly show that OntoCase improves the quality of the ontologies produced, compared to existing OL methods.

Bibliography

- [1] Nationalencyklopedin, entry on the term "ontologi". Available online: http://www.ne.se/jsp/search/article.jsp?i_art_id=276179. (Accessed 2008-08-27). [cited at p. 25]
- [2] Nationalencyklopedin, entry on the term "semiotik". Available at: <http://www.ne.se/artikel/1174236>. (Accessed 2008-11-11). [cited at p. 29]
- [3] Representing classes as property values on the semantic web. Available at: <http://www.w3.org/TR/swbp-classes-as-values/>, April 2005. [cited at p. 95, 193]
- [4] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues , methodological variations, and system approaches. *AICom*, 7:39–59, 1994. IOS Press. [cited at p. 98]
- [5] R. Al-Halimi, R. C. Berwick, J. F. M. Burg, M. Chodorow, Christiane Fellbaum, Joachim Grabowski, Sanda Harabagiu, Marti A. Hearst, Graeme Hirst, Douglas A. Jones, Rick Kazman, Karen T. Kohl, Shari Landes, Claudia Leacock, George A. Miller, Katherine J. Miller, Dan Moldovan, Naoyuki Nomura, Uta Priss, Philip Resnik, David St-Onge, Randee Tengi, Reind P. van de Riet, and Ellen Voorhees. *WordNet - An Electronic Lexical Database*. MIT Press, 1998. [cited at p. 30, 48, 53, 67]
- [6] H. Alani and C. Brewster. Ontology Ranking based on the Analysis of Concept Structures. In *Proceedings of K-CAP'05*, Banff, Alberta, Canada, October 2005. [cited at p. 71, 211, 223, 226]
- [7] T. Albertsen and E. Blomqvist. Describing ontology applications. In *Proceedings of the 4th European Semantic Web Conference (ESWC07)*, Innsbruck, Austria, June 3-7 2007, 2007. [cited at p. 146, 147]
- [8] C. Alexander. *The Timeless Way of Building*. Oxford University Press, New York, 1979. [cited at p. 80]

- [9] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York, 1977. [cited at p. 80]
- [10] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999. [cited at p. 47, 121]
- [11] C. F. Baker, C. J. Fillmore, and J. B. Lowe. The berkeley framenet project. In *Proceedings of the COLING-ACL*, Montreal, Canada, 1998. [cited at p. 90]
- [12] J. Bao, D. Caragea, and V. G Honavar. Towards collaborative environments for ontology construction and sharing. In *Proceedings of the International Symposium on Collaborative Technologies and Systems (CTS 2006)*, 2006. [cited at p. 69]
- [13] K. Barker, B. Porter, and P. Clark. A Library of Generic Concepts for Composing Knowledge Bases. In *Proceedings of the International Conference on Knowledge Capture*, pages 14–21, Victoria, British columbia, 2001. ACM Press, New York. [cited at p. 67]
- [14] V. R. Basili. The role of experimentation in software engineering: Past, current and future. In *Proceedings of ICSE-18*, 1996. [cited at p. 105]
- [15] K. Beck, J. O. Coplien, R. Crocker, L. Dominick, G. Meszaros, F. Paulisch, and J. Vlissides. Industrial experience with design patterns. In *Proceedings of the 18th International Conference on Software Engineering*. IEEE Computer Society Press, 1996. [cited at p. 11]
- [16] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American Magazine*, 2001. [cited at p. 26, 38]
- [17] A. Bernstein and E. Kaufmann. Gino - a guided input natural language ontology editor. In *Proceedings of the 5th International Semantic Web Conference (ISWC 2006)*, Athens, Georgia (US), November 2006. [cited at p. 44]
- [18] A. Billig and K. Sandkuhl. Match-Making based on Semantic Nets - The XML-based Approach of BaSeWeP. In *XML Technologien für das Semantic Web - XSW 2002, Proceedings zum Workshop*, June 2002. [cited at p. 70, 94]
- [19] A. Billig and K. Sandkuhl. Enterprise ontology based artefact management. *GI Jahrestagung*, P134:681–687, 2008. [cited at p. 5, 193, 194, 344]
- [20] G. Bisson, C. Nédellec, and D. Canamero. Designing clustering methods for ontology building: The Mo’K workbench. In S. Staab, A. Maedche, C. Nédellec, and P. Wiemer-Hastings, editors, *Proceedings of the First Workshop on Ontology Learning OL’2000, Berlin, Germany, August 25, 2000*, August 2000. [cited at p. 54]

- [21] S. Björk, S. Lundgren, and J. Holopainen. Game Design Patterns. In M. Copier and J. Raessens, editors, *Level Up - Proceedings of Digital Games Research Conference 2003, Utrecht, The Netherlands, 4-6 November 2003*, 2003. [cited at p. 83, 86]
- [22] E. Blomqvist and K. Sandkuhl. Patterns in Ontology Engineering: Classification of Ontology Patterns. In *Proceedings of the International Conference on Enterprise Information Systems 2005*, Miami Beach, Florida, May 24-28 2005. [cited at p. 146]
- [23] J. Bos. Wide-Coverage Semantic Analysis with Boxer. In Johan Bos and Rodolfo Delmonte, editors, *Semantics in Text Processing. STEP 2008 Conference Proceedings*, volume 1 of *Research in Computational Semantics*, pages 277–286. College Publications, 2008. [cited at p. 59]
- [24] C. Brewster, H. Alani, S. Dasmahapatra, and Y. Wilks. Data Driven Ontology Evaluation. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC 2004)*, Lisbon, Portugal, 2004. [cited at p. 61, 120, 121]
- [25] C. Brewster, F. Ciravegna, and Y. Wilks. User-centred Ontology Learning for Knowledge Management. In *Proceedings 7th International Workshop on Applications of Natural Language to Information Systems*, Stockholm, 2002. [cited at p. 55, 82, 93]
- [26] C. Brewster, F. Ciravegna, and Y. Wilks. Background and foreground knowledge in dynamic ontology construction. In *Proceedings of the Semantic Web Workshop, SIGIR*, 2003. [cited at p. 62]
- [27] P. Buitelaar, T. Eigner, , and T. Declerck. Ontoselect: A dynamic ontology library with support for ontology selection. in . In *Proceedings of the Demo Session at the ISWC04, Hiroshima, Japan, Nov. 2004*, 2004. [cited at p. 70, 71]
- [28] P. Buitelaar, T. Eigner, and T. Declerck. OntoSelect: A Dynamic Ontology Library with Support for Ontology Selection. In *Proc. of the Demo Session at the ISWC'04*, Hiroshima, Japan, Nov. 2004. [cited at p. 222]
- [29] P. Buitelaar, D. Olejnik, and M. Sintek. A protégé plug-in for ontology extraction from text based on linguistic analysis. In *Proceedings of the 1st European Semantic Web Symposium (ESWS)*, Heraklion, Greece, May 2004. [cited at p. 59]
- [30] F. Burstein and S. Gregor. The systems development or engineering approach to research in information systems: An action research perspec-

- tive. In *Proceedings of ACIS99*, pages 122–134, Wellington, NZ, 1999. [cited at p. 111, 112, 113]
- [31] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-oriented Software Architecture - A System of Patterns*. John Wiley & Sons, Chichester, 1996. [cited at p. 64, 80, 81, 83, 84, 85, 86, 87, 343]
- [32] A. Chalmers. *What Is This Thing Called Science?* Open University press, 1999. [cited at p. 104]
- [33] H. Cherfi and Y. Toussaint. How far Association Rules and Statistical Indices help Structure Terminology. In *15th European Conference on Artificial Intelligence (ECAI'02): Workshop on Machine Learning and Natural Language Processing for Ontology Engineering, Lyon, France, 2002*. [cited at p. 49, 50]
- [34] P. Cimiano. *Ontology Learning and Population from Text - Algorithms, Evaluation and Applications*. Springer, 2006. [cited at p. 8, 30, 32, 34, 45, 46, 49, 51, 92, 208, 209]
- [35] P. Cimiano, P. Buitelaar, and B. Magnini, editors. *Ontology Learning and from Text: Methods, Evaluation and Applications*. IOS Press, 2005. [cited at p. 46]
- [36] P. Cimiano, A. Hotho, and S. Staab. Learning concept hierarchies from text corpora using formal concept analysis. *Journal of Artificial Intelligence Research*, 24:305339, 2005. [cited at p. 49]
- [37] P. Cimiano and J. Völker. Text2onto - a framework for ontology learning and data-driven change discovery. In *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB)*, Lecture Notes in Computer Science, Alicante, Spain, June 2005. [cited at p. 51]
- [38] P. Clark, J. Thompson, and B. Porter. Knowledge Patterns. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *KR2000: Principles of Knowledge Representation and Reasoning*, San Francisco, 2000. Morgan Kaufman. [cited at p. 90, 91, 94, 343]
- [39] W. Cohen, P. Ravikumar, and S. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *Proc. of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03), August 9-10, 2003, Acapulco, Mexico*, 2003. [cited at p. 178, 212]
- [40] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001. [cited at p. 143]

- [41] N. Cross. Designerly ways of knowing: Design discipline versus design science. *Design Issues*, 17(3):49–55, Summer 2001. [cited at p. 107]
- [42] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proc. of ACL'02*, 2002. [cited at p. 209]
- [43] M. d'Aquin, C. Baldassarre, L. Gridinoc, M. Sabou, S. Angeletou, and E. Motta. Watson: Supporting next generation semantic web applications. In *Proceedings of the WWW/Internet conference, Vila real, Spain, 2007*, 2007. [cited at p. 70]
- [44] I. Davies, P. Green, S. Milton, and M. Rosemann. Using Meta Models for the Comparison of Ontologies. In *Proc. of Eval. of Modeling Methods in Systems Analysis and Design Workshop-EMMSAD'03*, 2003. [cited at p. 120]
- [45] B. Davis. The Power of Knowledge Pattern Recognition. White Paper, available at: http://www.kikm.org/pattern_recog.htm, 2001. [cited at p. 90]
- [46] G. de Chalendar and B. Grau. How to Classify Words Using their Context. In *Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management, EKAW2000, Juan-les-Pins, France, October 2000*, pages 203–216. Springer, 2000. [cited at p. 48]
- [47] A. Dearden, J. Finlay, L. Allgar, and B. McManus. Evaluating pattern languages in participatory design. In *Proceedings of CHI2002*, Minneapolis, USA, April 2002. ACM. [cited at p. 11]
- [48] W. Deiters, T. Löffeler, and S. Pfennigschmidt. The information logistics approach toward user demand-driven information supply. In *Cross-media service delivery : Based on papers presented at the Conference on Cross-Media Service Delivery - CMSD-2003*. Kluwer Academic Publishers, 2003. [cited at p. 26]
- [49] K. Dellschaft, H. Engelbrecht, J. Monte Barreto, S. Rutenbeck, and S. Staab. Cicero: Tracking design rationale in collaborative ontology engineering. In *Online Demo Proceedings of the 5th European Semantic Web Conference (ESWC2008)*, Tenerife, Spain, June 2008. [cited at p. 45]
- [50] K. Dellschaft and S. Staab. On How to Perform a Gold Standard Based Evaluation of Ontology Learning. In *In Proceedings of the 5th International Semantic Web Conference (ISWC2006)*, volume 4273 of *LNCS*, Athens, GA, USA, November 2006. [cited at p. 121]

- [51] V. Dimitrova, R. Denaux, G. Hart, C. Dolbear, I. Holt, and A. G. Cohn. Involving domain experts in authoring owl ontologies. In *Proceedings of the 7th International Semantic Web Conference (ISWC2008)*, 2008. [cited at p. 44]
- [52] L. Ding, R. Pan, T. Finin, A. Joshi, Y. Peng, and P. Kolari. Finding and ranking knowledge on the semantic web. In *Proceedings of ISWC 2005*, 2005. [cited at p. 70]
- [53] Y. Ding and R. Engels. IR and AI: Using Co-occurrence Theory to Generate Lightweight Ontologies. In *Proceedings of the 12th International Workshop on Database and Expert Systems Applications, 3-7 sept. 2001*. IEEE, 2001. [cited at p. 49]
- [54] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to Map between Ontologies on the Semantic Web. In *Proceedings of the eleventh international conference on World Wide Web*, pages 662–673, Honolulu, Hawaii, 2002. ACM. [cited at p. 73]
- [55] M. Dzbor, M. C. Suárez-Figueroa, A. Gómez-Pérez, E. Blomqvist, H. Lewen, and M. Espinoza. D5.6.2 Experimentation with parts of NeOn methodology. Technical report, NeOn Project, 2009. Available at: <http://www.neon-project.org>. [cited at p. 161, 165]
- [56] J. Euzenat and P. Shvaiko. *Ontology Matching*. Springer Verlag Berlin-Heidelberg, 2007. [cited at p. 72, 73]
- [57] A. Faatz and R. Steinmetz. Ontology Enrichment with Texts from the WWW. In *Proceedings of ECML - Semantic Web mining 2002, Helsinki, Finland*, 2002. [cited at p. 60]
- [58] D. Faure and C. Nédellec. A Corpus-based Conceptual Clustering Method for Verb Frames and Ontology Acquisition. In *LREC Workshop on Adapting Lexical and Corpus Resources to Sublanguages and Applications*, Granada, Spain, 1998. [cited at p. 48, 54]
- [59] D. Faure, C. Nédellec, and C. Rouveirol. Acquisition of Semantic Knowledge using Machine Learning Methods: The System "ASIUM". Technical Report ICS-TR-88-16, Université Paris Sud, Paris, 1998. [cited at p. 54, 55, 92]
- [60] E. A. Feigenbaum. Themes and case studies of knowledge engineering. In D. Michie, editor, *Expert Systems in the Micro-Electronic Age*. Edinburgh University Press, Edinburgh, Scotland, 1979. [cited at p. 2]

- [61] D. Fensel and A. Gómez Pérez. Deliverable 1.3: A survey on ontology tools. OntoWeb, Ontology-based information exchange for knowledge management and electronic commerce, IST-2000-29243, May 2002. [cited at p. 44, 72]
- [62] E. B. Fernandez and X. Yuan. Semantic Analysis Patterns. In *Proceedings of the 19th International conference on conceptual Modelling, ER2000*, pages 183–195, Salt Lake City, October 2000. [cited at p. 83]
- [63] M. Fernández, A. Gómez-Pérez, and N. Juristo. Methontology: from ontological art towards ontological engineering. In *Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering*, 1997. [cited at p. 6]
- [64] M. Fernández López. Overview of Methodologies for Building Ontologies. In *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem Solving Methods (KRR5) Stockholm, Sweden, August 2, 1999*, 1999. [cited at p. 43]
- [65] R. Fikes and A. Farquhar. Distributed repositories of highly expressive reusable ontologies. *IEEE Intelligent Systems*, 14(2):73–79, March-April 1999. [cited at p. 43, 66, 70]
- [66] J. Firth. A synopsis of linguistic theory 1930-1955. In *Studies in Linguistic Analysis*. Philological Society, Oxford, 1957. reprinted in Palmer, F. (ed. 1968) *Selected Papers of J. R. Firth*, Longman, Harlow. [cited at p. 293]
- [67] B. Fortuna, M. Grobelnik, and D. Mladenic. Background knowledge for ontology construction. In *Poster proceedings of WWW 2006*, 2006. [cited at p. 57]
- [68] B. Fortuna, M. Grobelnik, and D. Mladenic. Ontogen: Semi-automatic ontology editor. In *Proceedings of HCI International 2007*, 2007. [cited at p. 57]
- [69] M. Fowler. *Analysis Patterns - Reusable Object Models*. Addison-Wesley, 1997. [cited at p. 83, 84, 85, 160, 330, 331, 343]
- [70] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2003. With contributions from Rice, D., Foemmel, M., Hieatt, E., Mee, R. and Stafford, R. [cited at p. 83, 85, 86, 343]
- [71] K. Frantzi, S. Ananiadou, and J. Tsuji. The c-value/nc-value method of automatic recognition for multi-word terms. In *Proceedings of the ECDL*, 1998. [cited at p. 47, 209]
- [72] P. Gamallo, M. Gonzalez, A. Augustinin, G. Lopes, and V. S. de Lima. Mapping Syntactic Dependencies onto Semantic Relations. In *15th European Conference on Artificial Intelligence (ECAI'02): Workshop on Ma-*

- chine Learning and Natural Language Processing for Ontology Engineering, Lyon, France, 2002.* [cited at p. 48]
- [73] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995. [cited at p. 80, 81, 83, 85, 86, 343]
- [74] A. Gangemi. Ontology design patterns for semantic web content. In *The Semantic Web ISWC 2005*, volume 3729 of *Lecture Notes in Computer Science.* Springer, 2005. [cited at p. 96]
- [75] A. Gangemi, C. Catenacci, M. Ciaramita, and J. Lehmann. Ontology evaluation and validation - an integrated formal model for the quality diagnostic task. Technical report, LOA , ISTC-CNR, 2005. Available at: http://www.loa-cnr.it/Files/OntoEval4OntoDev_Final.pdf. [cited at p. 115, 116, 122, 126]
- [76] A. Gangemi, C. Catenacci, M. Ciaramita, and J. Lehmann. Modelling ontology evaluation. In *Proceedings of the Third European Semantic Web Conference.* Berlin, Springer, 2006. [cited at p. 120]
- [77] A. Gangemi, C. Catenacci, M. Ciaramita, and J. Lehmann. Qood grid: A metaontology-based framework for ontology evaluation and selection. In D. Vrandečić, M.C. Suarez, A. Gangemi, and Y. Sure, editors, *Proceedings of Evaluation of Ontologies for the Web, 4th International EON Workshop*, Located at the 15th International World Wide Web Conference WWW 2006, 2006. [cited at p. 114, 115, 117, 120, 121, 123, 183]
- [78] A. Gangemi and V. Presutti (eds.). Ontology Design Pattern portal. Available at: <http://www.ontologydesign-patterns.org>, 2008. [cited at p. 331]
- [79] A. Gangemi, J. Lehmann, V. Presutti, M. Nissim, and C. Catenacci. C-ODO: an OWL Meta-model for Collaborative Ontology Design. In *Proceedings of the Workshop on Social and Collaborative Construction of Structured Knowledge (CKC 2007) at the 16th International World Wide Web Conference (WWW2007)*, 2007. [cited at p. 44]
- [80] A. Gangemi and P. Mika. Understanding the Semantic Web through Descriptions and Situations. In *Proc. of the International Conference on Ontologies, Databases and Applications of SEMantics (ODBASE 2003)*, Catania, Italy, November 3-7 2003. [cited at p. 160, 330]
- [81] A. Gangemi and V. Presutti. Ontology design for interaction in a reasonable enterprise. In P. Rittgen, editor, *Handbook of Ontologies for Business Interaction*, 2007. [cited at p. 96]

- [82] K. Gardner, A. Rush, M. Crist, R. Konitzer, and B. Teegarden. *Cognitive Patterns - Problem-solving Frameworks for Object Technology*. Cambridge University Press, 1998. [cited at p. 160, 330, 331]
- [83] A. Geyer-Schulz and M. Hahsler. Software Reuse with Analysis Patterns. In *Proceedings of AMCIS 2002*, Dallas, Texas, August 2002. [cited at p. 83, 84]
- [84] A. Gómez-Pérez. Evaluation of Taxonomic Knowledge in Ontologies and Knowledge Bases. In *Banff Knowledge Acquisition for Knowledge-Based Systems, KAW'99*, volume 2, 16-21 October 1999. [cited at p. 117, 123, 183, 185, 241, 244, 256]
- [85] A. Gómez-Pérez, M. Fernández-Liópez, and O. Corcho. *Ontological Engineering*. Springer, 2004. [cited at p. 41, 117, 118, 119]
- [86] S. Gourlay. Towards conceptual clarity for tacit knowledge: a review of empirical studies. *Knowledge Management Research & Practice*, 4:60–69, 2006. [cited at p. 2]
- [87] T. Gruber. A translation approach to portable ontology specifications. In *Knowledge Acquisition*, volume 5, pages 199–220, 1993. [cited at p. 3, 31, 32, 33]
- [88] M. Gruninger and M. S. Fox. The role of competency questions in enterprise engineering. In *Proceedings of the IFIP WG5.7 Workshop on Benchmarking - Theory and Practice*, 1994. [cited at p. 44]
- [89] N. Guarino. Ontology and Information Systems. In *Proceedings of FOIS'98*, pages 3–15, 1998. [cited at p. 32, 39, 40, 343]
- [90] N. Guarino and P. Giaretta. Ontologies and Knowledge Bases: Towards a Terminological Clarification. In *Towards Very Large Knowledge Bases: Knowledge Building & Knowledge Sharing*, pages 25–32. IOS Press, 1995. [cited at p. 31, 32, 33]
- [91] N. Guarino and C. Welty. Evaluating Ontological Decisions with OntoClean. *Communications of the ACM*, 45(2):61–65, February 2002. [cited at p. 118, 123, 183]
- [92] U. Hahn and K. G. Markó. An integrated, dual learner for grammars and ontologies. *Data and Knowledge Engineering*, 42(3):273–291, 2002. [cited at p. 57, 92]
- [93] U. Hahn and K. G. Markó. Ontology and Lexicon Evolution by Text Understanding. In *15th European Conference on Artificial Intelligence*

- (*ECAI'02*): *Workshop on Machine Learning and Natural Language Processing for Ontology Engineering, Lyon, France, 2002*. [cited at p. 57]
- [94] U. Hahn and M. Romacker. Content Management in the SYNDIKATE system - How technical documents are automatically transformed to text knowledge bases. *Data & Knowledge Engineering*, 35:137–159, 2000. [cited at p. 57]
- [95] J. Hartmann, P. Spyns, A. Giboin, D. Maynard, R. Cuel, M. C. Suarez-Figueroa, and Y. Sure. D1.2.3 methods for ontology evaluation. Version 1.3.1, Available at: <http://knowledgeweb.semanticweb.org/>, Downloaded 2005-05-10, 2005. [cited at p. 115]
- [96] H. Hasan. Information systems development as a reasearch method. *Australasian Journal of Information Systems*, 11(1), 2003. [cited at p. 110]
- [97] D. C. Hay. *Data Model Patterns - Conventions of Thought*. Dorset House Publishing, 1996. [cited at p. 87, 88, 343]
- [98] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the Fourteenth International Conference on Computational Linguistics*, pages 539–545, Nantes, France, July 1992. [cited at p. 49, 82, 93, 95]
- [99] M. Hepp and J. de Bruijn. Gentax: A generic methodology for deriving owl and rdf-s ontologies from hierarchical classifications, thesauri, and inconsistent taxonomies. In *Proceedings of the 4th European Semantic Web Conference (ESWC2007)*. Springer Verlag, 2007. [cited at p. 43]
- [100] A. R. Hevner, S. T. March, J. Park, and S. Ram. Design Science Research in Information Systems. *Management Information Systems Quarterly*, 28(1):75–105, March 2004. [cited at p. 107, 109]
- [101] G. Heyer, M. Läuter, U. Quasthoff, T. Wittig, and C. Wolff. Learning Relations using Collocations. In Maedche, Staab, Nédellec, and Hovy, editors, *Proceedings IJCAI Workshop on Ontology Learning, Seattle/WA, 19-24 August 2001*, 2001. [cited at p. 49]
- [102] L. Holder, D. Cook, J. Gonzalez, and I. Jonyer. Structural Pattern Recognition in Graphs. In D. Chen and X. Cheng, editors, *Pattern Recognition and String Matching*, volume 13 of *Combinatorial Optimization*. Kluwer Academic Publishers, November 2003. [cited at p. 93]
- [103] R. Ichise, H. Takeda, and S. Honiden. Rule Induction for Concept Hierarchy Alignment. In *Proceedings of the IJCAI-01 Workshop on Ontology Learning (OL-2001)*, pages 26–29, Seattle, 2001. [cited at p. 73]

- [104] A. Inokuchi, T. Washio, and H. Motoda. An Apriori-based Algorithm for Mining Frequent Substructures from Graph Data. In *Proceedings of the 4th European Conference on Principles and Practice of Knowledge Discovery in Data Bases (PKDD)*, pages 13–23, Lyon, September 2000. [cited at p. 93]
- [105] Y. R. Jean-Mary and M. R. Kabuka. Asmov: Results for oaei 2008. In *Proceedings of the Third International Workshop on Ontology Matching*, 2008. [cited at p. 75]
- [106] D. Jones, T. Bench-Capon, and P. Visser. Methodologies for ontology development. In *Proceedings of IT&KNOWS Conference of the 15th IFIP World Computer Congress*, pages 62–75. Chapman and Hall Ltd, 1998. [cited at p. 43]
- [107] D. Jurafsky and J. H. Martin. *Speech and Language Processing - An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2000. [cited at p. 30, 31, 47]
- [108] Y. Kalfoglou and M. Schorlemmer. IF-Map: An Ontology-Mapping Method based on Information-Flow Theory. *Journal on Data Semantics I*, pages 98–127, 2003. Lecture Notes in Computer Science, Springer. [cited at p. 73, 74]
- [109] Y. Kalfoglou and M. Schorlemmer. Ontology Mapping: the state of the art. *The Knowledge Engineering Review*, 18(1):1–31, 2003. Cambridge University Press. [cited at p. 72]
- [110] M. Kavalec and V. Svátek. A study on automated relation labelling. In *Ontology Learning from Text: Methods, Evaluation and Applications*. IOS Press, 2005. [cited at p. 50]
- [111] M. Keet. Aspects of Ontology Integration. Available at: <http://www.dcs.napier.ac.uk/~cs203/AspectsOntologyIntegration.pdf>, 2004. [cited at p. 73]
- [112] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object oriented and frame based languages. *Journal of the ACM*, 42(4):741–843, 1995. [cited at p. 36]
- [113] M. Klein. Combining and relating ontologies: an analysis of problems and solutions. In A. Gomez-Perez, M. Gruninger, H. Stuckenschmidt, and M. Uschold, editors, *Workshop on Ontologies and Information Sharing, IJCAI'01*, Seattle, August 2001. [cited at p. 73]
- [114] M. Kokla and M. Kavouras. Fusion of Top-level and Geographic Domain Ontologies Based on Context Formation and Complementarity. *Interna-*

- tional Journal of Geographical Information Science*, 15(7):679–687, 2001. [cited at p. 73]
- [115] M. Kolp, P. Giorgini, and J. Mylopoulos. Organizational Patterns for Early Requirements Analysis. In *Proceedings of the 15th International Conference on Advanced Information Systems Engineering (CAISE'03)*, Velden, Austria, June 2003. [cited at p. 83, 84, 343]
- [116] M. Kuramochi and G. Karypis. Finding Frequent Patterns in a Large Sparse Graph. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, Lake Buena Vista, April 2004. [cited at p. 93]
- [117] D. B. Leake, A. Maguitman, T. Reichherzer, A. Caas, M. Carvalho, M. Arguedas, S. Brenes, and T. Eskridge. Aiding knowledge capture by searching for extensions of knowledge models. In *Proceedings of the Second International Conference on Knowledge Capture (K-Cap 2003)*, 2003. [cited at p. 98]
- [118] P. De Leenheer and C. Debruyne. DOGMA-MESS: A Tool for Fact-Oriented Collaborative Ontology Evolution. In *Proceedings of OTM Confederated International Workshops and Posters, Monterrey, Mexico, November 9-14, 2008.*, LNCS. Springer, 2008. [cited at p. 67]
- [119] D. B. Lenat. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995. [cited at p. 67]
- [120] T. V. Levashova, M. P. Pashkin, N. G. Shilov, and A. V. Smirnov. Ontology Management, II. *Journal of Computer and Systems Sciences International*, 42(5):744–756, 2003. [cited at p. 43, 66]
- [121] H. Lewen, K. Supekar, N. F. Noy, and M. A. Musen. Topicspecific trust and open rating systems: An approach for ontology evaluation. In *Proceedings of the 4th International EON Workshop - Evaluation of Ontologies for the Web, located at the 15th International World Wide Web Conference WWW 2006*, 2006. [cited at p. 72, 120]
- [122] A. Lozano-Tello and A. Gómez-Pérez. ONTOMETRIC: A Method to Choose the Appropriate Ontology. *Journal of Database Management*, 15(2), April-June 2004. [cited at p. 119, 124, 183, 187, 188]
- [123] A. Maedche. *Ontology Learning for the Semantic Web*. Kluwer Academic Publishers, 2002. [cited at p. 8, 32, 45, 46, 51, 82, 226]
- [124] A. Maedche and S. Staab. The TEXT-TO-ONTO Ontology Learning Environment. Software Demonstration at ICCS-2000 - Eight International

- Conference on conceptual Structures August 14-18, 2000, Darmstadt, Germany, available at: <http://www.aifb.uni-karlsruhe.de/~sst/Research/Publications/Publications.htm>, August 2000. [cited at p. 51]
- [125] A. Maedche and S. Staab. Learning Ontologies for the Semantic Web. In *Semantic Web 2001 (at WWW10)*, May 2001. [cited at p. 51]
- [126] A. Maedche and S. Staab. Ontology Learning. In S. Staab and Studer R., editors, *Handbook on Ontologies in Information Systems*. Springer, 2003. [cited at p. 51]
- [127] A. Maedche and R. Volz. The ontology Extraction & Maintenance Framework Text-To-Onto. In *ICDM'01: The 2001 IEEE International Conference on Data Mining Workshop on Integrating Data Mining and Knowledge Management*, November 2001. [cited at p. 51, 92, 178]
- [128] S. T. March and G. f. Smith. Design and natural science research on information technology. *Decision Support Systems*, 15:251–266, 1995. [cited at p. 107]
- [129] D. L. McGuinness. Ontologies Come of Age. In D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster, editors, *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press, 2002. [cited at p. 4, 38]
- [130] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An Environment for Merging and Testing Large Ontologies. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, Breckenridge, Colorado, April 2000. [cited at p. 73]
- [131] E. Mena, A. Kashyap, A. Illarramendi, and A. P. Sheth. OBSERVER: An approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies. In *Proceedings of the 1st IFCIS International Conference on Cooperative Information Systems (CoopIS '96), Brussels, Belgium, June 1996*, 1996. [cited at p. 73]
- [132] T. Menzies. Object-oriented patterns: Lessons from expert systems. *Software - Practice and Experience*, 1(1), December 1997. [cited at p. 11, 78, 79]
- [133] H. Mihoubi, A. Simonet, and M. Simonet. Towards a Declarative Approach for Reusing Domain Ontologies. *Information Systems*, 23(6):365–381, 1998. [cited at p. 73]
- [134] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller. Introduction to WordNet: an on-line lexical database. *International Journal of Lexicography*, 3(4):235–244, 1990. [cited at p. 53, 67]

- [135] M. Minsky. A Framework for Representing Knowledge. Technical Report AIM-306, Massachusetts Institute of Technology, 1974. [cited at p. 90]
- [136] P. Mitra, G. Wiederhold, and M. Kersten. A Graph-oriented Model for Articulation of Ontology Interdependencies. In *Proceedings Conference on Extending Database Technology 2000 (EDBT'2000)*, Konstanz, 2000. [cited at p. 73]
- [137] E. Motta and Z. Zdrahal. A library of problem-solving components based on the integration of the search paradigm with task and method ontologies. *Int. Journal of Human-Computer Studies*, 49:437–470, 1998. [cited at p. 90]
- [138] M. Nagy, M. Vargas-Vera, and P. Stolarski. Dssim results for oaei 2008. In *Proceedings of the Third International Workshop on Ontology Matching*, 2008. [cited at p. 75, 76]
- [139] D. Nardi, R. J. Brachman, F. Baader, W. Nutt, F. M. Donini, U. Sattler, D. Calvanese, R. Mitor, G. De Giacomo, R. Ksters, F. Wolter, D. L. McGuinness, P. F. Patel-Schneider, R. Mller, V. Haarslev, I. Horrocks, A. Borgida, C. Welty, A. Rector, E. Franconi, M. Lenzerini, and R. Rosati. *The Description Logic Handbook - Theory, Implementation and Applications*. Cambridge University Press, 2003. [cited at p. 36]
- [140] R. Navigli, P. Velardi, A. Cucchiarelli, and F. Neri. Automatic Ontology Learning: Supporting a Per-Concept Evaluation by Domain Experts. In *Workshop on Ontology Learning and Population (ECAI 2004)*, Valencia, Spain, 2004. [cited at p. 120, 121]
- [141] R. Navigli, P. Velardi, and A. Gangemi. Ontology Learning ant Its Application to Automated Terminology Translation. *IEEE Intelligent Systems*, 18(1):22–31, Jan-Feb 2003. [cited at p. 47, 53]
- [142] A. Newell. The knowledge level: Presedential address. *AI Magazine*, 2(2):1–20, Summer 1981. [cited at p. 28]
- [143] I. Niles and A. Pease. Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems - Volume 2001*, pages 2–9. ACM Press, New York, 2001. [cited at p. 67]
- [144] N. F. Noy and M. A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, Austin, Texas, 2000. [cited at p. 73]
- [145] N. F. Noy and M. A. Musen. Anchor-PROMPT: Using Non-Local Context for Semantic Matching. In *Workshop on Ontologies and Information*

- Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, Seattle, WA, 2001. [cited at p. 73]
- [146] N. F. Noy and M. A. Musen. Evaluating Ontology-Mapping Tools: Requirements and Experience. In *Workshop on Evaluation of Ontology Tools at EKAW'02 (EON2002)*, 2002. [cited at p. 73]
- [147] N. F. Noy, N. H. Shah, B. Dai, M. Dorf, N. Griffith, C. Jonquet, M. J. Montegut, D. L. Rubin, C. Youn, and M. A. Musen. Bioportal: A web repository for biomedical ontologies and data resources. In *Poster and Demo Proceedings of ISWC 2008*, 2008. [cited at p. 70]
- [148] J. F. Nunamaker and M. Chen. Systems development in information systems research. In *Proceedings of The 23rd Annual Hawaii International Conference on System Sciences*, 1990. [cited at p. 110]
- [149] C. Ogden and I. Richards. *The Meaning of Meaning - A Study of the Influence of Language upon Thought and of the Science of Symbolism*. Routledge, 1923. [cited at p. 29]
- [150] A. Öhgren. Ontology development and evolution: Selected approaches for small-scale application contexts. Technical Report ISSN 1404-0018;2004:7, Jnkping University, School of Engineering, 2004. [cited at p. 43]
- [151] A. Öhgren and K. Sandkuhl. Towards a Methodology for Ontology Development in Small and Medium-Sized Enterprises. In *IADIS Conference on Applied Computing*, Algarve, Portugal, February 2005. [cited at p. 177, 180]
- [152] A. Öhgren and K. Sandkuhl. DO SME NEED ONTOLOGIES? Results from a Survey among Small and Medium-sized Enterprises. In *To appear in: Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS2008)*, 2008. [cited at p. 1]
- [153] A. Oliveira, F. Câmara Pereira, and A. Cardoso. Automatic Reading and Learning from Text. In *Proceedings of the International Symposium on Artificial Intelligence, ISAI 2001*, 2001. [cited at p. 54]
- [154] L. Orbst, T. Hughes, and S. Ray. Prospects and possibilities for ontology evaluation: The view from ncor. In *Proceedings of Evaluation of Ontologies for the Web, 4th International EON Workshop*, Located at the 15th International World Wide Web Conference WWW 2006, 2006. [cited at p. 120]
- [155] S. K. Pal and S. Shiu. *Foundations of Soft Case-based Reasoning*. John Wiley & Sons Inc, 2004. [cited at p. 98, 99]

- [156] C. Patel, K. Supekar, Y. Lee, , and E. K. Park. OntoKhoj: A Semantic Web Portal for Ontology Searching, Ranking and Classification. In *Proceeding of the Workshop On Web Information And Data Management*. ACM, 2003. [cited at p. 70]
- [157] A. Pease, I. Niles, and J. Li. The Suggested Upper Merged Ontology: A Large Ontology for the Semantic Web and its Applications. In *Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web*, Edmonton, July-August 2002. [cited at p. 67]
- [158] H. S. Pinto and J. P. Martins. A Methodology for Ontology Integration. In *Proceedings of the First International Conference on Knowledge Capture 2001*, New York, USA, 2001. ACM. [cited at p. 73]
- [159] T. Pirlein and R. Studer. An environment for reusing ontologies within a knowledge engineering approach. *International Journal of Human-Computer Studies*, 43:945–965, 1995. [cited at p. 69]
- [160] M. Polanyi. *The Tacit Dimension*. Routledge, 1966. [cited at p. 2]
- [161] R. Porzel and R. Malaka. A Task-based Approach for ontology Evaluation. In *Workshop on Ontology Learning and Population (ECAI 2004)*, 2004. [cited at p. 120]
- [162] L. Prechelt, B. Unger, M. Philippsen, and W. Tichy. Two controlled experiments assessing the usefulness of design patterns documentation in program maintenance. *IEEE TRansactions on Software Engineering*, 2001. [cited at p. 11]
- [163] V. Presutti and A. Gangemi. Content ontology design patterns as practical building blocks for web ontologies. In *Proceedings of ER2008*, Barcelona, Spain, 2008. [cited at p. 96]
- [164] V. Presutti, A. Gangemi, S. David, G. Aguado de Cea, M. C. Suárez-Figueroa, E. Montiel-Ponsoda, and M. Poveda. D2.5.1: A library of ontology design patterns: reusable solutions for collaborative design of networked ontologies. Available at: <http://www.neon-project.org/> (Downloaded 2008-05-23), 2008. [cited at p. 94, 96, 97, 343]
- [165] F. Puppe. Knowledge Formalization Patterns. In *Proceedings of PKAW 2000, Sydney, Australia, 2000*, 2000. [cited at p. 90, 91, 343]
- [166] A. Rector. Modularisation of Domain Ontologies Implemented in Description Logics and related formalisms including OWL. In *Proceedings of the international conference on Knowledge capture*, pages 121–128, Sanibel island, 2003. ACM Press. [cited at p. 68]

- [167] J. R. Reich. Ontological design patterns: Metadata of molecular biological ontologies, information and knowledge. In *Database and Expert Systems Applications, 11th International Conference, DEXA 2000*, 2000. [cited at p. 95]
- [168] T. Russ, A. Valente, R MacGregor, and W. Swartout. Practical Experiences in Trading Off Ontology Usability and Reusability. In *Proceedings of the Twelfth Banff Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Alberta, Canada, October 1999. [cited at p. 73]
- [169] M. Sabou, V. Lopez, E. Motta, and V. Uren. Ontology Selection: Ontology Evaluation on the Real Semantic Web. In *Proceedings of the Evaluation of Ontologies on the Web Workshop, held in conjunction with WWW'2006*, 2006. [cited at p. 71, 72, 119]
- [170] K. Sandkuhl. Information logistics in networked organisations: Selected concepts and applications. In Jorge Cardoso, Jos Cordeiro, and Joaquim Filipe, editors, *Enterprise Information Systems VIII*, LNBP. Springer, To appear 2008. [cited at p. 2, 26, 27, 343]
- [171] R. C. Schank. *Explanation Patterns: Understanding Mechanically and Creatively*. Hillsdale, NJ: Erlbaum, 1986. [cited at p. 89]
- [172] R.C. Schank and R. Abelson. *Scripts, Plans, Goals, and Understanding*. Hillsdale, NJ: Earlbaum Assoc, 1977. [cited at p. 89]
- [173] F. Scharffe, Y. Ding, and D. Fensel. Towards correspondence patterns for ontology mediation. In *Proceedings of The Second International Workshop on Ontology Matching*, 2007. [cited at p. 76, 95]
- [174] A. Schlicht and H. Stuckenschmidt. Towards structural criteria for ontology modularization. In *Workshop on Modular Ontologies (WoMO)*, 2006. [cited at p. 68]
- [175] M. Schorlemmer. Duality in Knowledge Sharing. In *Seventh International Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, Florida, January 2002. AI&M 20-2002. [cited at p. 74]
- [176] M. Shaw. Some Patterns for Software Architectures. In J. M. Vlissides, J. O. coplien, and N. L. Kerth, editors, *Pattern Languages of Program Design*, volume 2, pages 255–269. Addison-Wesley, 1996. [cited at p. 83, 84, 85, 343]
- [177] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *Journal on Data Semantics*, IV:146–171, December 2005. [cited at p. 72]

- [178] P. Shvaiko and J. Euzenat. Ten challenges for ontology matching. In *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part II on On the Move to Meaningful Internet Systems*, number 5332 in LNCS, Monterrey, Mexico, 2008. [cited at p. 74, 77]
- [179] L. Silverston. *The Data Model Resource Book - A Library of Universal Data Models by Industry Types*, volume 2. John Wiley & Sons, 2001. [cited at p. 88, 160, 330, 331]
- [180] L. Silverston. *The Data Model Resource Book - A Library of Universal Data Models for All Enterprises*, volume 1. John Wiley & Sons, 2001. [cited at p. 88, 157, 158, 160, 330, 331, 343]
- [181] E. Simperl, C. Tempich, and D. Vrandečić. A methodology for ontology learning. In *Ontology Learning and Population: Bridging the Gap between Text and Knowledge*. IOS Press, 2008. [cited at p. 45, 61]
- [182] J. F. Sowa. *Knowledge Representation - Logical, Philosophical, and Computational Foundations*. Brooks/Cole, 2000. [cited at p. 3]
- [183] C. Sporleder. A Galois Lattice based Approach to Lexical Inheritance Hierarchy Learning. In *15th European Conference on Artificial Intelligence (ECAI'02): Workshop on Machine Learning and Natural Language Processing for Ontology Engineering, Lyon, France, 2002*. [cited at p. 48]
- [184] R. Srikant and R. Agrawal. Mining Generalized Association Rules. In *Proceedings of VLDB '95*, pages 407–419, 1995. [cited at p. 49]
- [185] S. Staab, M. Erdmann, and A. Maedche. Engineering Ontologies using Semantic Patterns. In D. O’Leary and A. Preece, editors, *Proceedings of the IJCAI-01 Workshop on E-business & The Intelligent Web*, Seattle, 2001. [cited at p. 89]
- [186] R. D. Stacey. *Complex Responsive Processes in Organizations - Learning and Knowledge Creation*. Routledge, 2001. [cited at p. 28]
- [187] M. Stevenson. Combining Disambiguation Techniques to Enrich an Ontology. In *15th European Conference on Artificial Intelligence (ECAI'02): Workshop on Machine Learning and Natural Language Processing for Ontology Engineering, Lyon, Franc, 2002*. [cited at p. 60]
- [188] H. Stuckenschmidt. Modularization of Ontologies. WonderWeb Deliverable D21, available at: <http://wonderweb.semanticweb.org/deliverables/D21.shtml>, May 2003. [cited at p. 68]

- [189] H. Stuckenschmidt and J. Euzenat. Ontology Language Integration: A Constructive Approach. In *Proceedings of the Workshop on Application of Description Logics at the Joint German and Austrian Conference on AI, CEUR-Workshop Proceedings*, volume 44, 2001. [cited at p. 89]
- [190] H. Stuckenschmidt and M. C. A. Klein. Integrity and change in modular ontologies. In *Proceedings of IJCAI'03*, pages 900–908, 2003. [cited at p. 67]
- [191] R. Studer, R. Benjamins, and D. Fensel. Knowledge Engineering: Principles and Methods. *Data and Knowledge Engineering*, 25:161–197, 1998. [cited at p. 3, 31, 33]
- [192] G. Stumme and A. Maedche. FCA-MERGE: Bottom-Up Merging of Ontologies. In *Proceedings of the 7:th International Joint Conference on Artificial Intelligence, Seattle, USA, August 1-6 2001, San Fransisco/CA*. Morgan Kaufmann, 2001. [cited at p. 73]
- [193] M. C. Suárez-Figueroa, G. Aguado de Cea, C. Buil, C. Caracciolo, M. Dzubor, A. Gómez-Pérez, G. Herrero, H. Lewen, E. Montiel-Ponsoda, and V. Presutti. D5.3.1 neon development process and ontology life cycle. NeOn deliverable, available at: <http://www.neon-project.org>, August 2007. [cited at p. 43]
- [194] M. C. Suárez-Figueroa and A. Gómez-Pérez. Building Ontology Networks: How to Obtain a Particular Ontology Network Life Cycle? In *Proceedings of the International Conference on Semantic Systems (ISEMANTICS08)*, Graz, Austria, September 2008. [cited at p. 41]
- [195] M. C. Suárez-Figueroa and A. Gómez-Pérez. Neon methodology: Scenarios for building networks of ontologies. In *16th International Conference on Knowledge Engineering and Knowledge Management, Knowledge Patterns, Poster and Demo Proceedings*, Acitrezza, Catania, Sicily, September-October 2008. (Online proceedings). [cited at p. 44]
- [196] M. C. Suárez-Figueroa and A. Gómez-Pérez. Towards a Glossary of Activities in the Ontology Engineering Field. In European Language Resources Association (ELRA), editor, *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, May 2008. [cited at p. 41]
- [197] V. Sugumaran and V. C. Storey. Ontologies for conceptual modeling: their creation, use, and management. *Data & Knowledge Engineering*, 42:251–271, 2002. [cited at p. 66]

- [198] K. Supekar, C. Patel, and Y. Lee. Characterizing Quality of Knowledge on Semantic Web. In *Proceedings of AAAI Florida AI Research Symposium (FLAIRS-2004)*, Miami Beach, Florida, May 17-19 2004. [cited at p. 120]
- [199] A. Sutcliffe. *The Domain Theory - Patterns for Knowledge and Software Reuse*. Lawrence Erlbaum Associates, 2002. [cited at p. 63, 64, 65, 66, 67, 69, 70, 91, 92, 159, 160, 330, 331, 343]
- [200] E. Thomas, J. Z. Pan, and D. Sleeman. ONTOSEARCH2: Searching Ontologies Semantically. In *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions*, 2007. Online publication: <http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-258/>. [cited at p. 70, 72]
- [201] C. Thörn, Ö. Eriksson, E. Blomqvist, and K. Sandkuhl. Potentials and limits of graph-algorithms for discovering ontology patterns. In *Proceedings of the International Conference on Intelligent Agents, Web Technology and Internet Commerce - IAWTIC'2005*, Wien, Austria, November 2005. [cited at p. 93, 151, 156]
- [202] C. Thrn, K. Sandkuhl, and W. Webers. Enterprise Ontology and Feature Model Integration: Approach and Experiences from an Industrial Case. In *Proceedings of the Second International Conference on Software and Data Technologies (ICSOFT 2007)*, Barcelona, Spain, July 2007. [cited at p. 6, 194]
- [203] H. Tsoukas and E. Vladimirou. What is organizational knowledge. *Journal of Management Studies*, 38(7), November 2001. [cited at p. 28]
- [204] T. Tudorache and N. Fridman Noy. Collaborative protege. In *Proceedings of the Workshop on Social and Collaborative Construction of Structured Knowledge (CKC 2007) at the 16th International World Wide Web Conference (WWW2007)*, 2007. [cited at p. 45]
- [205] R. Ungrangsi, C. Anutariya, and V. Wuwongse. Enabling Efficient Knowledge Reuse in the Semantic Web with SQORE. In *Proceedings of the Third International Conference on Semantics, Knowledge and Grid*. IEEE Computer Society, 2007. [cited at p. 70, 72]
- [206] M. Uschold. Building Ontologies: Towards a Unified Methodology. In *Proceedings of Expert Systems '96, the 16th Annual Conference of the British Computer Society Specialist Group on Expert Systems*, Cambridge, UK, December 1996. [cited at p. 66]

- [207] M. Uschold, M. King, S. Moralee, and Y. Zorgios. The enterprise ontology. *The Knowledge Engineering Review*, 13, Special Issue on Putting Ontologies to Use, 1998. [cited at p. 4, 39, 40, 207]
- [208] G. van Heijst, A. T. Schreiber, and B. J. Wielinga. Using explicit ontologies for KBS development. *International Journal of Human-Computer Studies*, 46(2-3):183–292, February 1997. [cited at p. 39]
- [209] P. Visser and T. Bench-Capon. On the Reusability of Ontologies in Knowledge-System Design. In *Seventh International Workshop on Databases and Expert Systems Applications, DEXA '96, Zurich, Switzerland, September 9-10, 1996*. IEEE-CS Press, 1996. [cited at p. 69]
- [210] J. Voelker, D. Vrandečić, Y. Sure, and A. Hotho. Learning disjointness. In *Proceedings of the 4th European Semantic Web Conference*, Innsbruck, Austria, June 2007. [cited at p. 186]
- [211] J. Völker, E. Blomqvist, C. Baldassarre, and S. Rudolph. D3.8.2 Evaluation of Methods for Contextualized Learning of Networked Ontologies. NeOn Deliverable, Available at: <http://www.neon-project.org/>, March 2009. [cited at p. 266]
- [212] J. Völker, P. Haase, and P. Hitzler. Learning expressive ontologies. In *Ontology Learning and Population: Bridging the Gap between Text and Knowledge*, Frontiers in Artificial Intelligence and Applications. IOS Press, 2008. [cited at p. 48, 58]
- [213] J. Völker, P. Hitzler, and P. Cimiano. Acquisition of OWL DL Axioms from Lexical Resources. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, *Proceedings of the 4th European Semantic Web Conference (ESWC'07)*, Lecture Notes in Computer Science. Springer, 2007. [cited at p. 48]
- [214] J. Völker and S. Rudolph. Lexico-logical acquisition of owl dl axioms - an integrated approach to ontology refinement. In *Proceedings of the 6th International Conference on Formal Concept Analysis (ICFCA'08)*, volume 4933 of *Lecture Notes in Artificial Intelligence*, February 2008. [cited at p. 49]
- [215] D. Vrandečić and Y. Sure. How to design better ontology metrics. In *Proceedings of the 4th European Semantic Web Conference (ESWC07)*, Innsbruck, Austria, June 2007. Springer. [cited at p. 95]
- [216] P. Wang and B. Xu. Lily: Ontology alignment results for oaei 2008. In *Proceedings of the Third International Workshop on Ontology Matching*, 2008. [cited at p. 75]

- [217] Y. Wang, P. Haase, and J. Bao. A survey of formalisms for modular ontologies. In *Proceedings of the International Joint Conference on Artificial Intelligence 2007 (IJCAI'07) Workshop SWeCKa*, 2007. [cited at p. 69]
- [218] R. O. Weber, K. D. Ashley, and S. Brüninghaus. Textual case-based reasoning. *The Knowledge Engineering Review*, 20(3):255–260, 2006. [cited at p. 98]
- [219] P. Wiemer-Hastings, A. C. Graesser, and K. Wiemer-Hastings. Inferring the Meaning of Verbs from Context. In *Proceedings of the Twentieth Annual Conference of the Cognitive Science Society*, 1998. [cited at p. 48]
- [220] S.-H. Wu and W.-L. Hsu. SOAT: A Semi-Automatic Domain Ontology Acquisition Tool from Chinese Corpus. In *COLING 2002: The 17th International Conference on Computational Linguistics: Project Notes*, Taipei, Taiwan, 2002. [cited at p. 47, 50, 51, 92]
- [221] G. Xexéo, A. Vivacqua, J. Moreira de Souza, B. Braga, J. Nogueira D’Almeida Jr, B. Kinder Almentero, R. Castilho, and B. Miranda. Coe: A collaborative ontology editor based on a peer-to-peer framework. *Advanced Engineering Informatics*, 19(2):113–121, April 2005. Collaborative Environment for Design and Manufacturing. [cited at p. 45]
- [222] H. Yao, A. M. Orme, and L. Eitzkorn. Cohesion Metrics for Ontology Design and Application. *Journal of Computer Science*, 1(1):107–113, 2005. [cited at p. 117, 123, 183]
- [223] P. Yeh, B. Porter, and K. Barker. Using Transformations to Improve Semantic Matching. In *Proceedings of the international conference on Knowledge capture*, pages 180–189, Sanibel Islands, 2003. ACM Press. [cited at p. 70, 94]
- [224] M. V. Zelkowitz and D. R. Wallace. Experimental models for validating technology. *IEEE Computer*, 31:23–31, 1998. [cited at p. 105]
- [225] X. Zhang, Q. Zhong, J. Li, J. Tang, G. Xie, and H. Li. Rimom results for oaei 2008. In *Proceedings of the Third International Workshop on Ontology Matching*, 2008. [cited at p. 75, 76]
- [226] Y. Zhang, W. Vasconcelos, and D. Sleeman. Ontosearch: An ontology search engine. In *Proceedings of the AI-2004 Conference*. Springer, December 2004. [cited at p. 70]
- [227] J. Zhong, H. Zhu, J. Li, and Y. Yu. Conceptual Graph Matching for Semantic Search. In *Proceedings of the 10th International Conference on Conceptual Structures (ICCS 2002)*, Borovets, Bulgaria, July 2002. Springer. [cited at p. 70, 94]

Appendices

Appendix A

Ontology metamodel

To help the reader of this thesis understand our perspective on ontologies and what such ontologies generally contain, a metamodel of ontologies was developed. The metamodel describes the parts that constitute an ontology and how they are related. The model should mainly be understood as a way of clarifying terminology for this thesis, thereby the model does not aim to be complete and some of the parts of the model are not defined in full detail. Due to visualisation constraints, the model is here shown in several parts, concerning the different elements of an ontology. The parts of the model relevant for the currently implemented version of the OntoCase method are shown in a darker colour, to give the reader an idea of what is currently the elements explicitly treated by OntoCase. The metamodel has been described using the UML modelling language*. Figure A.1 describes the terminology concerning ontologies and knowledge bases. Figure A.2 takes a closer look at the term 'concept', while Figures A.3 and A.4 focus on hierarchical relations and other relations respectively.

*Information about the UML language can be found at <http://www.uml.org/>.

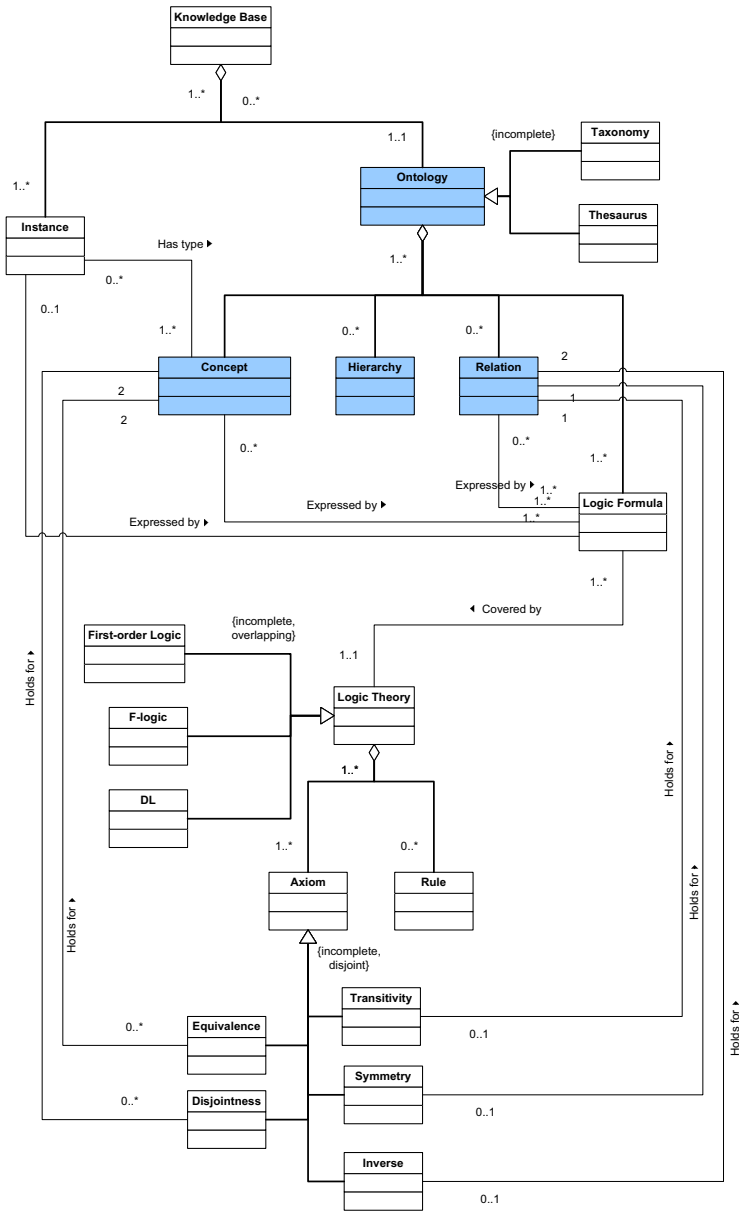


Figure A.1: The relation between knowledge bases, ontologies and logic.

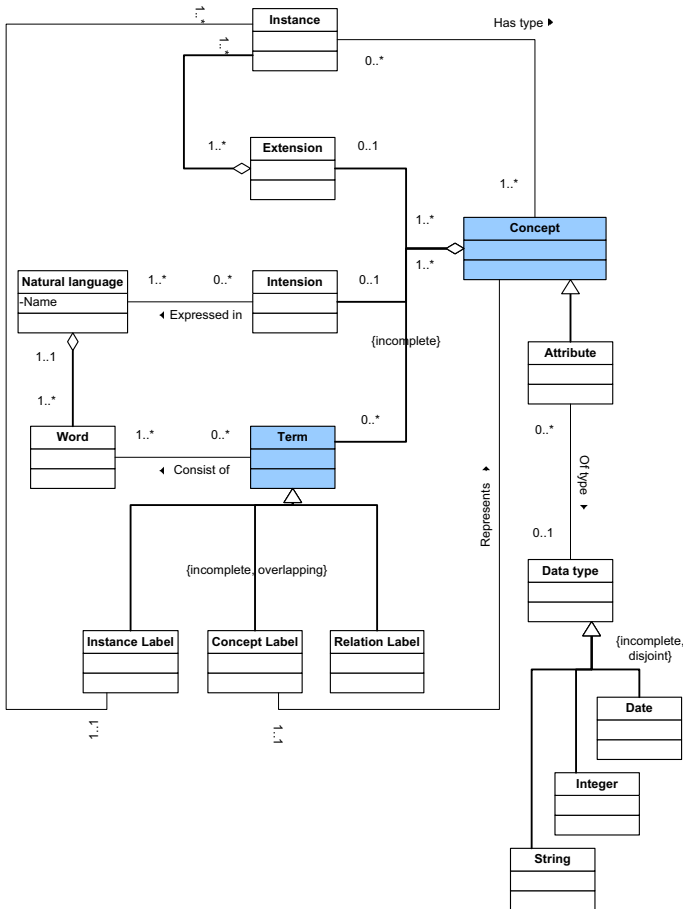


Figure A.2: Metamodel describing our view of ontology concepts.

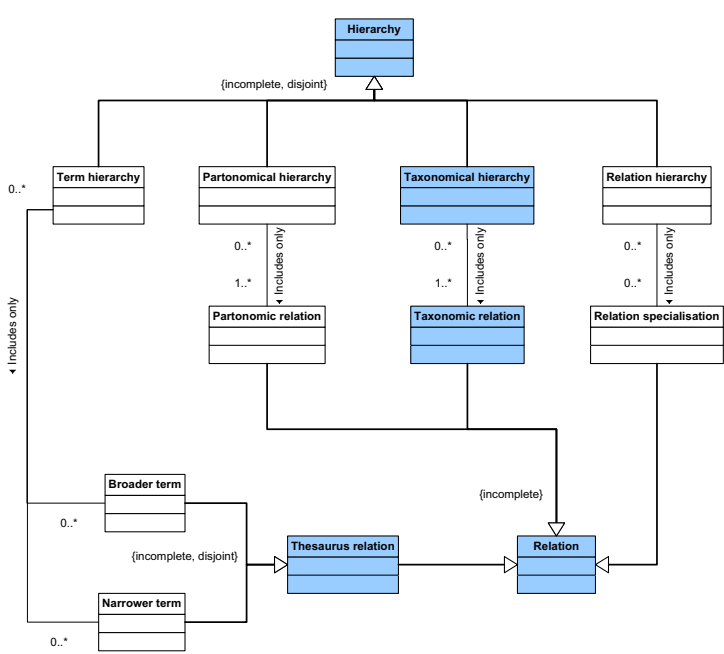


Figure A.3: Hierarchies found in ontologies.

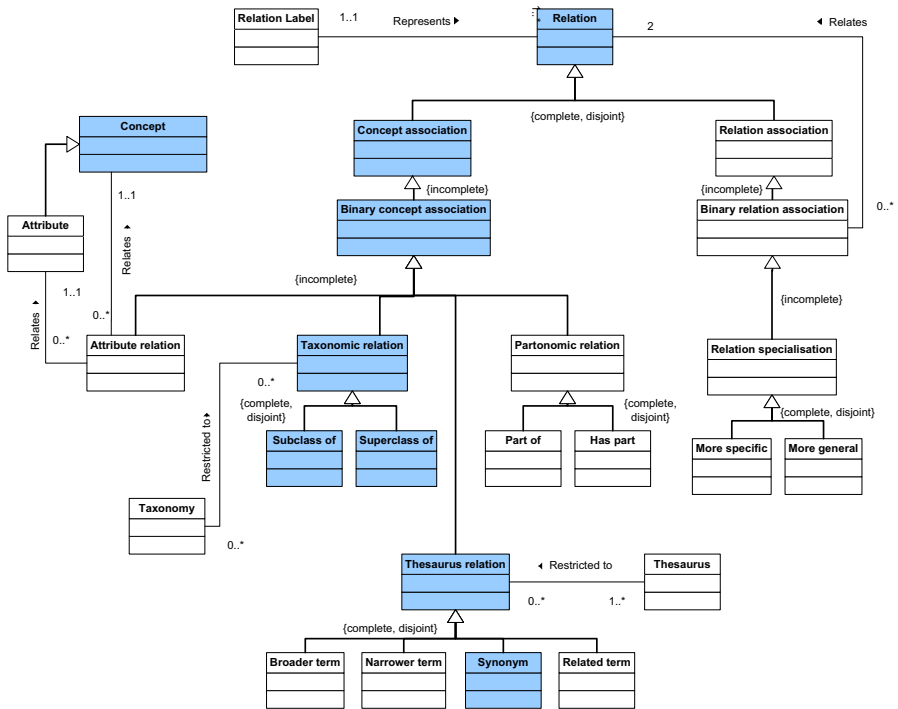


Figure A.4: Metamodel describing our view of ontological relations.

Appendix B

Pattern catalogues

As described previously in this thesis an initial pattern catalogue was developed based on knowledge reused from other fields, such as data modelling. For completeness we here repeat the list of patterns and their sources in Table B.1. Table B.2 shows the list of patterns in the extended catalogue used for the evaluations of OntoCase. In Table B.3 some details are given concerning the main topic and domain of each of the patterns, and the size in terms of number of concepts. The patterns in this last catalogue are all represented in OWL, while in the initial catalogue the patterns were represented in F-logic. Two patterns from the original catalogue were not translated into OWL, due to difficulties in modelling the features originally translated from their data model counterparts. The patterns could have been remodelled, but this was not done due to the intention to keep the patterns as close to the original sources as possible. Additionally, one pattern was not translated and included in the extended catalogue since it was a pattern originally translated from an OWL ontology into F-logic, and it did not make sense to translate it back again. The patterns added from the ODP portal actually already covered this pattern, since they were extracted from the same source, a top-level ontology.

Table B.1: Patterns and their original sources.

Pattern name	Source
Actions	Analysis pattern [69]
Analysis and modelling	Goal structure [199]
Communication event	Data model [180]
DOLCE Descriptions and Situations	Top-level ontology [80]
Employee and department	Data model [180]
Engineering change	Data model [179]
Information acquisition	Goal structure [199]
Organisation	Data model [180]
Parts	Data model [179]
Party	Data model [180]
Party relationships	Data model [180]
Party roles	Data model [180]
Person	Data model [180]
Planning and scheduling	Goal structure [199]
Positions	Data model [180]
Product	Analysis pattern [69]
Product associations	Data model [180]
Product categories	Data model [180]
Product features	Data model [180]
Requirements	Data model [180]
Requirements analysis	Goal structure [199]
System	Cognitive pattern taxonomy [82]
System analysis	Cognitive pattern taxonomy [82]
System synthesis	Cognitive pattern taxonomy [82]
Validate and test	Goal structure [199]
Work effort	Data model [180]

Table B.2: Patterns in the extended pattern catalogue and their sources.

Pattern name	Source
Actions	Analysis pattern [69]
Analysis and modelling	Goal structure [199]
Communication event	Data model [180]
Employee and department	Data model [180]
Engineering change	Data model [179]
Information acquisition	Goal structure [199]
Organisation	Data model [180]
Parts	Data model [179]
Party	Data model [180]
Person	Data model [180]
Planning and scheduling	Goal structure [199]
Positions	Data model [180]
Product	Analysis pattern [69]
Product associations	Data model [180]
Product categories	Data model [180]
Product features	Data model [180]
Requirements	Data model [180]
Requirements analysis	Goal structure [199]
System	Cognitive pattern taxonomy [82]
System analysis	Cognitive pattern taxonomy [82]
System synthesis	Cognitive pattern taxonomy [82]
Validate and test	Goal structure [199]
Work effort	Data model [180]
Time interval	proposed ODP [78]
Precedence	proposed ODP [78]
Participation	proposed ODP [78]
Information realisation	proposed ODP [78]
Description	proposed ODP [78]
Agent role	proposed ODP [78]
Classification	proposed ODP [78]
Collection entity	proposed ODP [78]
Constituency	proposed ODP [78]
Co-participation	proposed ODP [78]
GO top	proposed ODP [78]
Invoice	proposed ODP [78]
Metonymy 1 FAO	proposed ODP [78]
N-ary participation	proposed ODP [78]
Object role	proposed ODP [78]
Situation	proposed ODP [78]
Species v1.0 model	proposed ODP [78]
Task role	proposed ODP [78]
Types of entities	proposed ODP [78]

Table B.3: Pattern content

Name	Conc.	Domain	Topic
Actions	8	Organisation	Planning and follow-up of actions
Analysis and modelling	23	Product dev.	Problem analysis and solution modelling
Communication event	25	Organisation	Types and usage of communication
Employee and department	2	Organisation	Relations concerning employees
Engineering change	18	Product dev.	Changes to specification and realizations
Information acquisition	15	Product dev.	Acquiring information
Organisation	8	Organisation	Types of organisations
Parts	24	Product dev.	Products, part and their specifications
Party	14	Organisation	Types of persons and organisations
Person	1	Organisation	Attributes of a person
Planning and scheduling	21	Organisation	Constructing plans and schedules
Positions	7	Organisation	Positions filled by different parties
Product	9	Product dev.	Contracts and products
Product associations	10	Product dev.	Types of associations between products
Product categories	11	Product dev.	Categories and classifications
Product features	24	Product dev.	Categories of attributes and their features
Requirements	21	Product dev.	Types and relations of requirements
Requirements analysis	4	Product dev.	General parts of requirements analysis
System	5	Product dev.	General categories of systems
System analysis	16	Product dev.	Types of system analysis methods
System synthesis	11	Product dev.	Modes of synthesis and tasks involved
Validate and test	19	Product dev.	Tasks for validation and testing
Work effort	24	Product dev., Organisation	Work effort categories and purpose
Time interval	1	General	Intervals with start and end time
Precedence	1	General	Sequencing of entities
Participation	2	General	Objects participating in events
Information realisation	2	General	Semiotic view of information
Description	2	General	Concepts and their use in descriptions
Agent role	4	General	Agents taking different kinds of roles
Classification	1	General	Concepts classifying entities
Collection entity	1	General	Collections with members
Constituency	1	General	Entities made up of constituents
Co-participation	3	General	Several objects participating in an event
GO top	4	Biology	Top categories of the Gene Ontology (GO)
Invoice	11	Business	Transactions and concepts concerning invoicing
Metonymy 1 FAO	2	Agriculture	Species and commodities
N-ary participation	5	General	Several entities participating in a situation
Object role	3	General	Objects taking different roles
Situation	1	General	Situation as setting for entities
Species v1.0 model	5	Biology, Agriculture	Top categories for species
Task role	2	General	Tasks for specific roles
Types of entities	5	General	Categories for general entities

To give the reader a better intuition concerning the nature of the patterns, a set of selected pattern examples are presented in brief below. These are examples from the catalogue of patterns developed for the initial experiment, but here the versions translated into OWL are shown. The illustrations use a UML-notation, produced by the ontology editor TopBraid Composer*.

The first pattern example is the Actions pattern. An illustration of the pattern structure can be seen in Figure B.1. The pattern models different types of actions, with respect to their status, such as proposed actions, completed actions, and abandoned actions. Actions may also have a suspension, if the action has been suspended or aborted. Proposed actions form a plan. In order to show a detailed example of the syntactic representation of a content pattern the OWL XML-syntax of the Actions pattern is also displayed in Listing B.1.

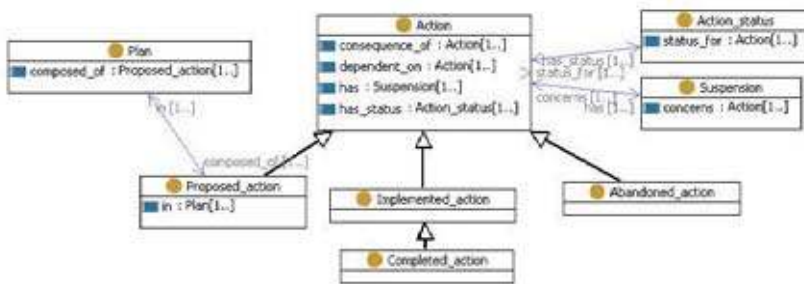


Figure B.1: The Actions pattern.

Listing B.1: The representation of the Actions pattern.

```
<?xml version="1.0" ?>
<rdf:RDF
  xmlns="http://www.eva.ing.hj.se/Actions#"
  xmlns:oxml="http://schema.ontoprise.com/oxml/core/2.1#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xml:base="http://www.eva.ing.hj.se/Actions">
  <owl:Ontology rdf:about=""/>
```

*<http://www.topquadrant.com/topbraid/composer/>

```

<owl:Class rdf:ID="Plan">
  <rdfs:label xml:lang="en">http://www.eva.ing.hj.se/Actions#
    Plan</rdfs:label>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom>
        <owl:Class rdf:ID="Proposed_action" />
      </owl:allValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="composed_of" />
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality rdf:datatype="http://www.w3.org
        /2001/XMLSchema#nonNegativeInteger"
        >1</owl:minCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#composed_of" />
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/
    owl#Thing" />
</owl:Class>
<owl:Class rdf:ID="Action_status">
  <rdfs:label xml:lang="en">http://www.eva.ing.hj.se/Actions#
    Action_status</rdfs:label>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="status_for" />
      </owl:onProperty>
      <owl:minCardinality rdf:datatype="http://www.w3.org
        /2001/XMLSchema#nonNegativeInteger"
        >1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/
    owl#Thing" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom>
        <owl:Class rdf:ID="Action" />
      </owl:allValuesFrom>
      <owl:onProperty>

```

```

    <owl:ObjectProperty rdf:about="#status_for" />
  </owl:onProperty>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Proposed_action">
  <rdfs:label xml:lang="en">http://www.eva.ing.hj.se/Actions#
    Proposed_action</rdfs:label>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Action" />
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality rdf:datatype="http://www.w3.org
        /2001/XMLSchema#nonNegativeInteger"
        >1</owl:minCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="in" />
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom rdf:resource="#Plan" />
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#in" />
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Abandoned_action">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Action" />
  </rdfs:subClassOf>
  <rdfs:label xml:lang="en">http://www.eva.ing.hj.se/Actions#
    Abandoned_action</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="Completed_action">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Implemented_action" />
  </rdfs:subClassOf>
  <rdfs:label xml:lang="en">http://www.eva.ing.hj.se/Actions#
    Completed_action</rdfs:label>
</owl:Class>
<owl:Class rdf:about="#Implemented_action">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Action" />

```

```

</rdfs:subClassOf>
<rdfs:label xml:lang="en">http://www.eva.ing.hj.se/Actions#
  Implemented action</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="Suspension">
  <rdfs:label xml:lang="en">http://www.eva.ing.hj.se/Actions#
    Suspension</rdfs:label>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom>
        <owl:Class rdf:about="#Action" />
      </owl:allValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="concerns" />
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#concerns" />
    </owl:onProperty>
    <owl:minCardinality rdf:datatype="http://www.w3.org
      /2001/XMLSchema#nonNegativeInteger"
      >1</owl:minCardinality>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/
  owl#Thing" />
</owl:Class>
<owl:Class rdf:about="#Action">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="consequence_of" />
      </owl:onProperty>
      <owl:minCardinality rdf:datatype="http://www.w3.org
        /2001/XMLSchema#nonNegativeInteger"
        >1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#consequence_of" />
    </owl:onProperty>
    <owl:allValuesFrom rdf:resource="#Action" />
  </owl:Restriction>

```



```

</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:ID="dependent_on" />
    </owl:onProperty>
    <owl:allValuesFrom rdf:resource="#Action" />
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:ID="has_status" />
    </owl:onProperty>
    <owl:minCardinality rdf:datatype="http://www.w3.org
      /2001/XMLSchema#nonNegativeInteger"
      >1</owl:minCardinality>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:allValuesFrom rdf:resource="#Action_status" />
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#has_status" />
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/
  owl#Thing" />
<rdfs:label xml:lang="en">http://www.eva.ing.hj.se/Actions#
  Action</rdfs:label>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:minCardinality rdf:datatype="http://www.w3.org
      /2001/XMLSchema#nonNegativeInteger"
      >1</owl:minCardinality>
    <owl:onProperty>
      <owl:ObjectProperty rdf:ID="has" />
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#has" />
    </owl:onProperty>
  </owl:Restriction>

```

```

    <owl:allValuesFrom rdf:resource="#Suspension" />
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#dependent_on" />
    </owl:onProperty>
    <owl:minCardinality rdf:datatype="http://www.w3.org
      /2001/XMLSchema#nonNegativeInteger"
      >1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:about="#dependent_on">
  <rdfs:domain rdf:resource="#Action" />
  <rdfs:range rdf:resource="#Action" />
  <rdfs:label xml:lang="en">http://www.eva.ing.hj.se/Actions#
    dependent_on</rdfs:label>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#in">
  <rdfs:range rdf:resource="#Plan" />
  <rdfs:domain rdf:resource="#Proposed_action" />
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#composed_of" />
  </owl:inverseOf>
  <rdfs:label xml:lang="en">http://www.eva.ing.hj.se/Actions#
    in</rdfs:label>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#concerns">
  <rdfs:range rdf:resource="#Action" />
  <rdfs:domain rdf:resource="#Suspension" />
  <rdfs:label xml:lang="en">http://www.eva.ing.hj.se/Actions#
    concerns</rdfs:label>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#has" />
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#consequence_of">
  <rdfs:domain rdf:resource="#Action" />
  <rdfs:range rdf:resource="#Action" />
  <rdfs:label xml:lang="en">http://www.eva.ing.hj.se/Actions#
    consequence_of</rdfs:label>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#composed_of">
  <rdfs:domain rdf:resource="#Plan" />
  <owl:inverseOf rdf:resource="#in" />

```

```

    <rdfs:range rdf:resource="#Proposed_action" />
    <rdfs:label xml:lang="en">http://www.eva.ing.hj.se/Actions#
      composed of</rdfs:label>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#has_status">
    <rdfs:range rdf:resource="#Action_status" />
    <rdfs:label xml:lang="en">http://www.eva.ing.hj.se/Actions#
      has status</rdfs:label>
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#status_for" />
    </owl:inverseOf>
    <rdfs:domain rdf:resource="#Action" />
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#status_for">
    <rdfs:domain rdf:resource="#Action_status" />
    <owl:inverseOf rdf:resource="#has_status" />
    <rdfs:range rdf:resource="#Action" />
    <rdfs:label xml:lang="en">http://www.eva.ing.hj.se/Actions#
      status for</rdfs:label>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#has">
    <rdfs:label xml:lang="en">http://www.eva.ing.hj.se/Actions#
      has</rdfs:label>
    <owl:inverseOf rdf:resource="#concerns" />
    <rdfs:range rdf:resource="#Suspension" />
    <rdfs:domain rdf:resource="#Action" />
  </owl:ObjectProperty>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 2.2, Build 339)
      http://protege.stanford.edu -->

```

The second pattern example is the Communication Event pattern. A partial illustration of the pattern structure can be seen in Figure B.2. In the figure some parts are hidden for increasing the readability. The pattern additionally includes small taxonomies of communication events, such as letter correspondence and phone communication, and communication even purposes, such as meetings, seminars, and enquiries. A communication event has a status and a mechanism type, which is the medium used for the communication. The parties involved have some kind of relationship, such as a seller-buyer relationship, where each has a specific role in the event, such as seller and buyer respectively. The roles are also connected to the purpose of the event, i.e. in the selling buying situation the purpose could be a sales follow-up.

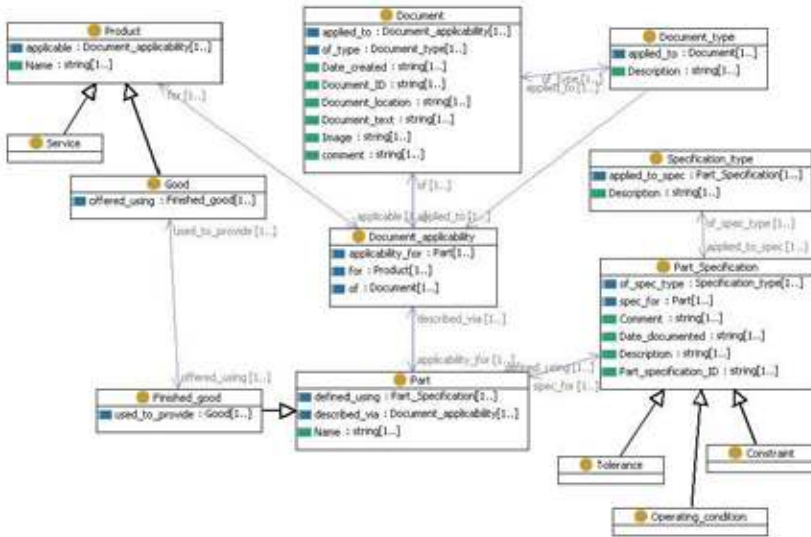


Figure B.3: The Parts pattern.

The above described patterns are examples of ontology content design patterns developed in our research. Below an additional example is given, to show the nature of the more general patterns present, for example in the ODP portal[†]. The pattern is the Agent role pattern, and can be seen in Figure B.5. The pattern is small, it includes only three concepts. Agents, that are types of objects, that in turn can be classified by certain roles. An example instantiation would be to specialise the 'agent' concept and add 'person' as a subconcept, then add 'parenting role' as a subconcept of 'role'. Such an instantiation can then be used to store information about the parenting roles, i.e. instances of the 'parenting role' concept such as 'father' and 'mother', of people, i.e. concrete instances of the 'person' concept.

[†]<http://ontologydesignpatterns.org/>

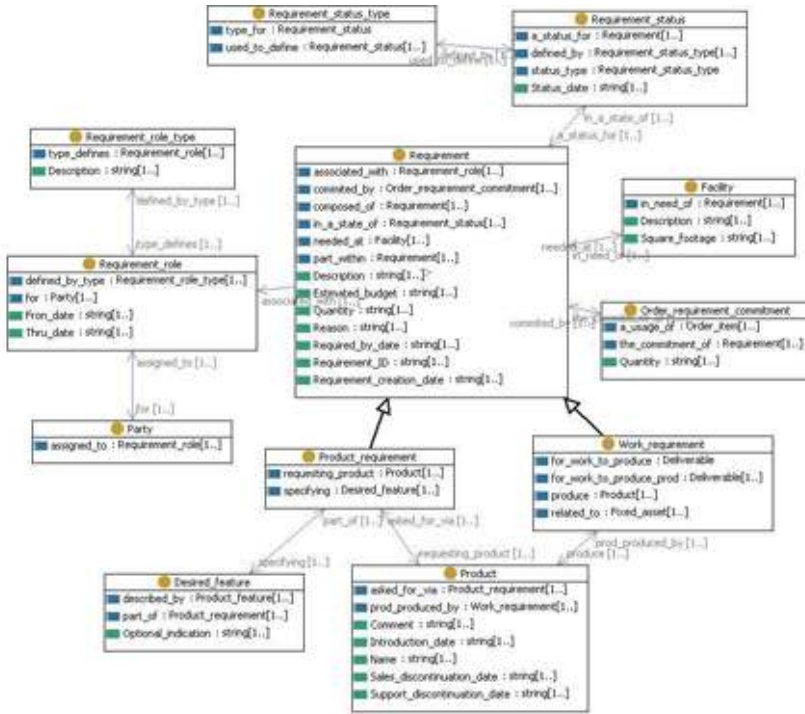


Figure B.4: The Requirements pattern.

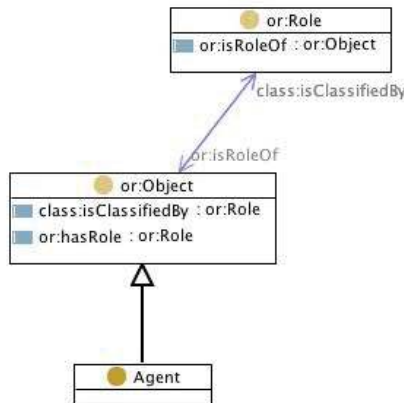


Figure B.5: The Agent role pattern.

List of Figures

1.1	Semi-automatic ontology construction and its activities.	8
1.2	The unexplored areas of OL.	13
1.3	The OntoCase framework.	17
2.1	The information logistics triangle, according to Sandkuhl [170].	27
2.2	An illustration of the semiotic triangle.	30
2.3	An example of two class definitions in the OWL XML-syntax. .	37
2.4	Ontologies with respect to their level of generality. [89]	40
2.5	The Ontology Learning layers.	46
3.1	The reuse processes. [199]	65
3.2	The account pattern. [69]	84
3.3	The joint venture pattern. [115]	84
3.4	The layered architecture pattern. [176]	85
3.5	The domain model pattern. [70]	86
3.6	The observer design pattern. [73]	86
3.7	Singleton Idiom in C++ from Buschmann et al. [31].	87
3.8	Products and services data model pattern. [97]	88
3.9	The 'free space axiom' in the container knowledge pattern. [38]	91
3.10	The heuristic decision tree pattern. [165]	91
3.11	The object inventory OSM. [199]	92
3.12	The typology proposed by Presutti et al. [164].	96
3.13	The agent role content pattern. [164].	97
4.1	The research process with an approximate time line.	127
4.2	The OntoCase phases in focus.	135
5.1	Levels of abstraction and granularity of ontology patterns. . . .	153
5.2	Data model pattern describing organisations. [180]	158

5.3	Resulting ontology content design pattern.	158
5.4	The perceived level of difficulty of the modelling problem.	166
5.5	Trivial and inspiring patterns.	167
5.6	Overall perceived usefulness.	167
5.7	Using patterns, was the task solved faster, easier and better?	168
5.8	The extent of managing to solve problems within time limit.	168
5.9	The easiest task and the 'best' ontology.	169
5.10	The easiest task and the 'best' ontology.	169
6.1	The basic steps of the initial method.	175
6.2	Steps of the ontology building, for each accepted pattern.	176
6.3	The structure of the experiment.	177
6.4	Top-level concepts of the aut. constructed ontology.	180
6.5	A part of the resulting ontology.	181
6.6	Top-level concepts of the man. constructed ontology.	182
6.7	The top-level concepts and relations of the resulting ontology.	193
6.8	The Protégé ArtifactManager plug-in. [19]	194
7.1	The OntoCase framework.	200
7.2	The OntoCase approach.	204
7.3	A pattern covering communication event concepts.	206
7.4	A pattern covering concepts related to positions.	206
7.5	A pattern covering work efforts and requirements.	212
7.6	List of extracted terms with associated confidence values.	212
7.7	Heuristic for adding unmatched relations.	215
7.8	Heuristic for taxonomic relations.	216
7.9	Heuristic for downward propagation of relations.	216
7.10	Heuristic for preserving extracted structure.	216
7.11	Heuristic for preserving pattern top levels.	217
7.12	The use of WordNet hypernym chains.	221
7.13	The relation matching.	222
7.14	An ontology design pattern including the positions concept.	227
7.15	An extract from the terms list used.	227
7.16	The concepts and properties of the input ontology.	233
7.17	The concepts and properties of the output ontology.	234
A.1	The relation between knowledge bases, ontologies and logic.	324
A.2	Metamodel describing our view of ontology concepts.	325
A.3	Hierarchies found in ontologies.	326
A.4	Metamodel describing our view of ontological relations.	327

B.1	The Actions pattern.	333
B.2	The Communication Event pattern.	340
B.3	The Parts pattern.	341
B.4	The Requirements pattern.	342
B.5	The Agent role pattern.	342

List of Tables

5.1	Mappings between data models and ontologies.	158
5.2	Chosen mappings between goal structures and ontologies.	159
5.3	Patterns and their original sources.	160
6.1	Comparison of structural characteristics.	184
6.2	Result of the OntoClean evaluation.	187
6.3	Results of the domain expert evaluation.	189
6.4	General characteristics	191
7.1	Precision and recall of text analysis for concept discovery.	210
7.2	String matching scores.	213
7.3	Example patterns	226
7.4	Ranking order (1-10) of the patterns (a-j).	229
8.1	Ranking values of the patterns in the SEMCO case.	240
8.2	General characteristics of the SEMCO ontologies.	242
8.3	Results of the OntoClean evaluation.	248
8.4	Number of concepts and relations collected in the sample.	254
8.5	General characteristics of JIBSNet ontology.	255
8.6	Result of the OntoClean evaluation of the JIBSNet ontologies.	258
8.7	Interpretation of assessment combinations.	261
8.8	Interpretation results.	262
8.9	Interpretation results concerning non-taxonomic relations.	263
8.10	The experiment input ontologies.	268
8.11	The enriched ontologies for the second part of the experiment.	268
8.12	The experiment output ontologies.	269
8.13	The evaluation results of the ontologies.	272
8.14	The number of patterns selected.	274

8.15	The patterns selected.	275
8.16	The number of completely and partially applicable patterns. . .	275
B.1	Patterns and their original sources.	330
B.2	Patterns in the extended pattern catalogue and their sources. .	331
B.3	Pattern content	332

Dissertations

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 processbeskrivningar 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 Fritzon:** 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 Reinfrank:** 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 Shahmehri:** 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 Kägedal:** 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 Inter-organisational 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 informa-

- tionsystem, 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 91-85643-97-2.
- 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 978-91-85715-65-7.
- 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 Comprehensive 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.

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 Wahlin:** Roadmap for Trend Detection and Assessment of Data Quality, 2008, ISBN: 978-91-7393-792-4

Linköping Studies in Information Science

- No 1 **Karin Axelsson:** Metodisk systemstrukturering - att skapa samstämmighet mellan informationssystemarkitektur 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, ISBN91-7373-736-4.
- 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.