State of the Art: Patterns in Ontology Engineering

Eva Blomqvist

ISSN 1404-0018
Research Report 04:8
State of the Art: Patterns in Ontology Engineering

Eva Blomqvist

Information Engineering Research Group
Department of Electronic and Computer Engineering
School of Engineering, Jönköping university
Jönköping, SWEDEN

ISSN 1404-0018
Research Report 04:8
Abstract

This report brings together three different areas, Ontology Learning, ontology reuse and patterns in Computer Science in general. These three areas may not seem to have much in common but the report aims to illustrate the potential of bringing them together and to outline research possibilities in the field. Patterns have been successfully applied as a means for facilitating reuse and managing complexity in many areas. So far not many pattern approaches have emerged in Ontology Engineering especially when considering patterns for use with Ontology Learning systems or patterns to facilitate reuse of ontologies. This report is concluded by a discussion about future research possibilities in the field. Among other things more exchange between Ontology Engineering and Software Engineering is suggested. Researchers should draw from already existing knowledge when creating ontology patterns. The most interesting applications of ontology patterns in the future are to further facilitate Ontology Learning, by for example using the patterns as construction templates, and to facilitate reuse of ontologies by using the patterns to search and sort ontology libraries.

Keywords

Ontologies, Ontology Engineering, Patterns, Ontology Learning, Ontology Reuse
# Table of Contents

1. **Introduction** ...................................................... 1
   1.1 Background and Motivation .................................. 1
   1.2 Aim ................................................................. 2
   1.3 Method ............................................................ 2
   1.4 Delimitation ..................................................... 2
   1.5 Disposition ...................................................... 3

2. **Basic Concepts** .................................................. 4
   2.1 What is an Ontology? ........................................... 4
   2.2 Ontology Usage Areas .......................................... 8
   2.3 How are Ontologies Constructed Today? ..................... 9
      2.3.1 Combining Ontologies .................................... 10
   2.4 Reuse .............................................................. 11
   2.5 What is a Pattern? ............................................. 13

3. **Ontology Learning** ............................................... 15
   3.1 Semi-automatic Approaches ................................... 15
      3.1.1 Basic Methods ............................................. 15
      3.1.2 SOAT ....................................................... 17
      3.1.3 Text-To-Onto ............................................... 18
      3.1.4 OntoLearn ................................................ 19
      3.1.5 TextStorm .................................................. 20
      3.1.6 ASIUM ..................................................... 21
      3.1.7 Adaptiva ................................................... 22
      3.1.8 SYNDIKATE ............................................... 23
   3.2 Ontology Enrichment ........................................... 24
   3.3 Ontology Learning Summary .................................. 24

4. **Ontology Reuse** .................................................. 25
   4.1 Generic Components ........................................... 26
   4.2 Modularisation .................................................. 27
   4.3 Ontology Matching, Integration and Merging ............. 28
   4.4 Ontology Reuse Summary ...................................... 30

5. **Patterns** .......................................................... 31
   5.1 Pattern Usage Areas .......................................... 31
5.1.1 Software Patterns ......................................................... 31
5.1.2 Data Model Patterns ................................................... 36
5.1.3 Linguistic Patterns ...................................................... 37
5.1.4 Semantic Patterns ...................................................... 37
5.1.5 Knowledge Patterns .................................................... 38
5.1.6 Pattern Recognition .................................................... 40

5.2 Pattern Levels with Applications to Ontologies .................. 41
5.2.1 Syntactic Patterns ...................................................... 41
5.2.2 Semantic Patterns ...................................................... 42
5.2.3 Design Patterns ......................................................... 43
5.2.4 Architecture Patterns .................................................. 44
5.2.5 Application Patterns ................................................... 45

5.3 Patterns Summary .......................................................... 46

6 Outlook and Summary .................................................... 48
6.1 Future Research Possibilities ........................................... 48
6.2 Summary ................................................................. 50

References ................................................................. 51
1 Introduction

This report brings together three different areas, Ontology Learning, ontology reuse and patterns in Computer Science in general. These three areas may not seem to have much in common but this report aims to illustrate the potential of bringing them together and as a conclusion to outline research possibilities in the field.

1.1 Background and Motivation

Patterns are a thoroughly researched and accepted part of Software Engineering. The idea of using patterns originated in architecture and was later adopted by many other fields, such as Computer Science. The most well-known patterns in Computer Science today are Software Patterns, in the form of for example Analysis Patterns, Architecture Patterns and Design Patterns. These patterns aim at helping software developers reuse well-tested solutions and handle the complexity of software design.

To motivate the use of ontologies (and patterns for their construction) another area first needs to be considered, namely Information Logistics. Information Logistics is about getting the right information at the right time at the right place and through the right channel. This is so far a quite unexplored area but an area that is getting more and more attention every day, as we are all flooded with more and more information. To facilitate solutions in Information Logistics some definition of the application scenario of interest is needed, some description of what is meant by different concepts and their relations, in order to be able to interpret the need for information and structure information originating from different sources. This shared understanding is commonly obtained by using ontologies.

Information Logistics is important in many different areas of the community, and of course also in the business world. Large companies often have structured information management systems and procedures, smaller companies might not, and with the radical increase in information availability today this poses a problem. Information Logistics is thereby especially important for Small and Medium Sized Enterprises (SMEs), which is the general target audience of our research.

Ontologies are apparently a very important part of an Information Logistics system, but how can such an ontology be constructed? How can it be maintained and maybe reused in other application cases? So far, ontology construction has been an all together manual process. There exist several methodologies and tools for ontology construction, but the work is done manually by an Ontology Engineer. The process thereby requires several experts, both Ontology Engineers and domain experts (who are familiar with the concepts to be described by the ontology). This creates problems, especially in small-scale application cases where these resources may not be present.
The latest trend in ontology construction is therefore to automate parts of the Ontology Engineering process and try to eliminate the need for Ontology Engineers, to let the domain experts build their own ontologies. The area of Semi-automatic Ontology Construction has lately been labelled Ontology Learning since most of the approaches have drawn heavily on Machine Learning techniques.

Another way to help ease the process of Ontology Engineering is to reuse parts of pre-existing ontologies when constructing new ones. This is not a very well explored area either, but an area that most researchers agree is very important. Together with Semi-automatic Ontology Construction it could provide means of reducing the need for Ontology Engineers.

The next step in the process of facilitating and automating the construction of ontologies may be the use of patterns. This development has been seen in many other areas, like Software Engineering as stated in the beginning of this section, and it would surely be useful in the ontology field, too. There have been some attempts at using patterns for ontologies but no general effort has been made. This document will show how the Ontology Engineering area could benefit from more exchange of methods and experiences with other parts of Computer Science, mainly Software Engineering and Software Patterns, and the result at the end of this report is a discussion of interesting research possibilities in the area.

1.2 Aim

This document aims at showing what has been done so far in the research areas of Ontology Learning, ontology reuse and also selected parts of the pattern communities in Computer Science. The goal is to identify areas and ideas for further research in the different fields and also in the combination of them.

1.3 Method

The work is based on a literature study of research papers and books in the different fields. Some brief testing of existing systems has also been conducted to better understand what the state of research is today.

1.4 Delimitation

Ontologies are used in many different areas and can be defined in many different ways, as will be explained further in chapter 2. This report will focus on Enterprise Ontologies for structuring information and for use in Enterprise Applications, especially in small-scale application contexts. The term ontology is defined according to chapter 2.1. There are other forms of ontologies for other purposes and in other fields
than Computer Science but these will not be considered in this report. For this rea-
son, when discussing Ontology Learning and ontology reuse, several approaches and
systems have been left out. These systems deal for example with specific biological
applications of ontologies, ontologies for teaching meta-data, personal ontologies or
agreeing on ontologies within multi-agent systems.

When discussing patterns, only patterns used in Computer Science are consid-
ered, and the discussed pattern categories (usage areas) are chosen either because of
their importance and significance in general or by their specific significance to the
topic of this report. Thereby many pattern categories and applications have surely
been omitted, mainly because they fall outside the scope of this report.

1.5 Disposition

The following chapter discusses the basics, what an ontology is, why we want to use
it and what methods and tools are available today. Also, the basic ideas of reuse and
patterns are described. In chapter 3 the different approaches and systems existing in
the field of Ontology Learning are presented in brief. Chapter 4 deals with the field
of ontology reuse and chapter 5 discusses patterns in different contexts. Finally, in
chapter 6, areas for future research are identified and a short summary is presented
as a conclusion of this report.
2 Basic Concepts

In this chapter some basic concepts are presented and the differing definitions that exist, concerning most of the terms, are discussed.

2.1 What is an Ontology?

There are numerous definitions of the term ontology in the literature, both recent and older definitions. To start from the beginning, the classical definition of ontology is:

*The branch of metaphysics that deal with the nature of being.*

This definition is not very useful in the Computer Science area, so in this field other definitions have emerged. Here is a ”small” collection of some of the most common definitions:

- An ontology defines the basic terms and relations comprising the vocabulary of a topic area, as well as the rules for combining terms and relations to define extensions to the vocabulary [66].
- An ontology is an explicit specification of a conceptualisation [31].
- Ontology as a specification of a conceptualisation, Ontology as a philosophical discipline, Ontology as an informal conceptual system (knowledge level), Ontology as a formal semantic account (knowledge level), Ontology as a representation of a conceptual system via a logical theory (symbolic level), Ontology as the vocabulary used by a logical theory (symbolic level), Ontology as a (meta-level) specification of a logical theory (symbolic level) [34].
- An ontology is a formal description of entities and their properties. For any given ontology, the goal is to agree upon a shared terminology and set of constraints on the objects in the ontology [32].
- Ontology is a theory of vocabulary or concepts used for building artificial systems [64].
- An ontology is a hierarchically structured set of terms for describing a domain that can be used as a skeletal foundation for a knowledge base [96].
- An ontology provides the means for describing explicitly the conceptualisation behind the knowledge represented in a knowledge base [2].
- An ontology is a formal, explicit specification of a shared conceptualisation [92].
• 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 language using such a vocabulary are constrained by its ontological commitment. An ontology indirectly reflects this commitment by approximating these intended models [33].

• Ontologies are content theories about the sorts of objects, properties of objects, and relations between objects that are possible in a specified domain of knowledge [8].

• An IS ontology is an axiomatic theory made explicit by means of a specific formal language. The IS ontology is designed for at least one specific and practical application. Consequently, it depicts the structure of a specific domain of objects, and it accounts for the intended meaning of a formal vocabulary or protocols that are employed by the agents of the domain under investigation [105].

• Ontologies are (meta)data schemas, providing a controlled vocabulary of concepts, each with an explicitly defined and machine processable semantics [54].

• A (core) ontology is a tuple \( C := (C, \text{is}_a, R, \sigma) \), where \( C \) is a set whose elements are called concepts, \( \text{is}_a \) is a partial order on \( C \) (i.e., a binary relation \( \text{is}_a \subseteq C \times C \) which is reflexive, transitive and antisymmetric), and \( R \) is a set whose elements are called relation names (or relations for short), and \( \sigma : R \rightarrow C^+ \) is a function which assigns to each relation name its arity [93].

• An ontology is a 4-tuple \( \Omega := (C, \text{is}_a, R, \sigma) \), where \( C \) is a set we call concepts, \( \text{is}_a \) is a partial order relation on \( C \), \( R \) is a set of relation names and \( \sigma : R \rightarrow \wp(C \times C) \) is a function [17].

• ... systematic, computer-oriented representation of the world. Researchers often refer to such a world model as an ontology [62].

• An ontology structure is a 5-tuple \( O := \{C, R, H^C, \text{rel}, A^O\} \), consisting of
  - two disjoint sets \( C \) and \( R \) whose elements are called concepts and relations respectively.
  - a concept hierarchy \( H^C : H^C \) is a directed relation \( H^C \subseteq C \times C \) which is called concept hierarchy or taxonomy. \( H^C(C_1, C_2) \) means that \( C_1 \) is a subconcept of \( C_2 \).
  - a function \( \text{rel} : R \rightarrow C \times C \), that relates concepts non-taxonomically\(^4\). The function \( \text{dom} : R \rightarrow C \) with \( \text{dom}(R) := \Pi_1(\text{rel}(R)) \) gives the domain of \( R \), and \( \text{range} : R \rightarrow C \) with \( \text{range}(R) := \Pi_2(\text{rel}(R)) \) give its range. For \( \text{rel}(R) = (C_1, C_2) \) one may also write \( R(C_1, C_2) \).
- a set of ontology axioms $A^O$, expressed in an appropriate logical language, e.g. first order logic.

4 In this generic definition one does not distinguish between relations and attributes.

A lexicon for the ontology structure $O := \{C, R, H^C, \text{rel}, A^O\}$ is a 4-tuple $L := \{L^C, L^R, F, G\}$ consisting of

- two sets $L^C$ and $L^R$, whose elements are called lexical entries for concepts and relations, respectively.
- two relations $F \subseteq L^C \times C$ and $G \subseteq L^R \times R$ called references for concepts and relations, respectively. Based on $F$, let for $\ell \in L^C$, $F(\ell) = \{c \in C \mid (\ell, c) \in F\}$ and for $F^{-1}(c) = \{\ell \in L^C \mid (\ell, c) \in F\}$. $G$ and $G^{-1}$ are defined analogously.

In general, one lexical entry may refer to several concepts or relations and one concept or relation may be referred by several lexical entries. An ontology structure with lexicon is a pair $(O, L)$, where $O$ is an ontology structure and $L$ is a lexicon [52].

The above definitions range from very informal to very precise, using mathematical notation for the definitions, but give a good idea of the diversity in opinion of what an ontology really is. The definitions in [66], [31], [64], [2], [33], [105] [54] and [62] are too general and imprecise to be useful in a technical context and the definition in [34] is more an explanation that ontologies can be viewed on different levels of abstraction than an actual definition of an ontology.

The other definitions have some good parts but none covers all aspects of an ontology. From the definition in [32] it can be derived that ontologies have entities, relations and constraints and aim to create a shared terminology. In [96] it can be noted that an ontology is hierarchically structured and can be used to create a knowledge base. In [92] it is stated that it has to be machine readable, contain concepts, properties, relations, functions, axioms and constraints. It also has to be a consensual description of some phenomenon in the world. From the definition in [8] it can be derived that it has to contain concepts, properties and relations and that it corresponds to a specific domain of knowledge. If these definitions are combined it could be phrased as follows:

An ontology is a hierarchically structured set of concepts describing a specific domain of knowledge, that can be used to create a knowledge base. An ontology contains concepts, a subsumption hierarchy, arbitrary relations between concepts, and axioms. It may also contain other constraints and functions.
The above definition is a general description of the concept “ontology”, but if one wants to use the definition in a more technical context it is better to use a mathematical notation. The definitions in [93] and [17] are not very well explained and does not contain anything about for example attributes or axioms, but the definition in [52] on the other hand is well defined, precise and easy to understand. With some slight modifications it will be used in this report as the definition of an ontology:

An ontology structure is a 5-tuple $O := \{C, R, H^C, rel, A^O\}$, consisting of

- two disjoint sets $C$ and $R$ whose elements are called concepts and relations respectively.
- a concept hierarchy $H^C$: $H^C$ is a directed relation $H^C \subseteq C \times C$ which is called concept hierarchy or taxonomy. $H^C(C_1, C_2)$ means that $C_1$ is a subconcept of $C_2$.
- a function $rel : R \rightarrow C \times C$, that relates concepts non-taxonomically (note that this also includes attributes). The function $\text{dom}(R) := \Pi_1(\text{rel}(R))$ gives the domain of $R$, and $\text{range}(R) := \Pi_2(\text{rel}(R))$ give its range. For $\text{rel}(R) = (C_1, C_2)$ one may also write $R(C_1, C_2)$.
- a set of ontology axioms $A^O$, expressed in an appropriate logical language.

A lexicon for the ontology structure $O := \{C, R, H^C, rel, A^O\}$ is a 4-tuple $L := \{L^C, L^R, F, G\}$ consisting of

- two sets $L^C$ and $L^R$, whose elements are called lexical entries for concepts and relations, respectively.
- two relations $F \subseteq L^C \times C$ and $G \subseteq L^R \times R$ called references for concepts and relations, respectively. Based on $F$, let for $\ell \in L^C$, $F(\ell) = \{c \in C \mid (\ell, c) \in F\}$ and for $F^{-1}(c) = \{\ell \in L^C \mid (\ell, c) \in F\}$. $G$ and $G^{-1}$ are defined analogously.

One lexical entry may refer to several concepts or relations and one concept or relation may be referred by several lexical entries. An ontology is a pair $(O, L)$, where $O$ is an ontology structure and $L$ is a lexicon.

Finally a slight remark on the word “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 report ontologies will be discussed according to the definition above, so the word will be used with a small o and the appropriate inflected form.
2.2 Ontology Usage Areas

Ontologies can be used for many different purposes, like (see for example [57]):

- Providing a controlled vocabulary.
- Site organisation and navigation (browsing and comparative and customised search) for example on the web.
- Providing an "umbrella" structure from which to extend content.
- Word sense disambiguation.
- Consistency checking, validation and verification.
- Completion and correction of insufficient information.
- Interoperability support.
- Configuration support.
- Providing possibilities to exploit generalisation/specialisation information.

Ontologies can also be used in a variety of applications, ranging from Autonomous Agents to Web Portals, Corporate Intranets or Information Logistics systems. This variety influences the functions needed to be implemented in an ontology, for example sometimes a simple taxonomy might be enough but in other cases more advanced features, like axioms, constraints and reasoning capabilities, are needed. In the context of this report, the focus will be on Enterprise Ontologies mainly used in Information Logistics systems, which for example can mean providing a structure for content or supporting interoperability of knowledge sources.

Another view is to study ontologies at three different levels, according to [98]. These are Terminological Ontologies, Information Ontologies and Knowledge Modelling Ontologies. Where Terminological Ontologies deal with structuring terminology, Information Ontologies deal with the structuring of information in general (like documents, databases or web pages) and Knowledge Modelling Ontologies deal with more complex tasks in Knowledge Management. In this report the focus area is small-scale application contexts in the area of organisation and structuring of information, interoperability and configuration support. In the categorisation of [98] this puts the focus mainly on the second level, Information Ontologies, but might also be extended to Knowledge Modelling Ontologies.

Ontologies can be classified from yet another point of view, their level or generality (see [33]), as in Figure 1. Top-level Ontologies describe general concepts like space and time, which are independent of domains and specific problems. These
ontologies can be shared by large communities. Domain Ontologies describe a vocabulary of a specific domain and are specialisations of Top-level Ontologies. Task Ontologies in turn describe the vocabulary of a generic task or activity and are also specialisations of Top-Level Ontologies. Finally, an Application Ontology is a specialisation of a Domain Ontology adapted for a specific application or task (see also [52]). In this categorisation of ontologies, the focus of this report lies on Domain and Application Ontologies, where the domain is governed by the enterprise in question and, in case of an Application Ontology, also a certain application within that enterprise (both these variants will in this report be denoted by the term Enterprise Ontology).

![Ontology Categories Diagram](image)

Figure 1: Different kinds of ontologies according to their level of generality. [33]

The term Knowledge Base is sometimes used as a synonym to the term ontology. The difference is that a Knowledge Base also contains instances of the concepts represented in the ontology, but since an ontology is often used to structure a Knowledge Base this is sometimes confused.

### 2.3 How are Ontologies Constructed Today?

Today the Ontology Engineering process is mainly a manual process, quite similar to the commonly adopted Software Engineering methodologies. Either the ontology is built from scratch in a cumbersome process of extracting important concepts and finding generalisations, specialisations, relations and axioms (for example with methodologies like in [23]). Another way is to build it almost completely out of existing knowledge sources, as suggested in [24] or [51].
There are several well-known environments for constructing ontologies, like OntoEdit, Protégé and OILEd etc. All these systems support manual Ontology Engineering but do not give any assistance to the Ontology Engineer aside from the usual help functions (see for example [21]). There are also environments for merging or integrating ontologies (see for example [21] and [44]) in order to reuse them, like Chimaera (see [58]) or PROMPT (see [70] and [71]), which gives some support in finding overlaps in ontologies and suggests to the user how to integrate them. Still, it is up to the Ontology Engineer to do the actual integration or merging.

2.3.1 Combining Ontologies

As stated above, many Ontology Engineering environments include capabilities for integrating or merging ontologies, but the terms in this area are about as confusing as the term ontology itself. There is no clear definition of what is meant by matching, mapping, alignment, integration, merging and fusion of ontologies. All these terms are also used in other areas than combining ontologies so the definitions should also concur with those fields, in order to avoid confusion.

According to some recent studies ([72], [75] and [45] for example) there are two different approaches in the area of combining ontologies. Either the aim is to combine two ontologies of the same subject area (domain), in which case the process is named merging, or the aim is to combine two ontologies from different subject areas (domains), in which case it is named integration. This is not a definition adopted by all researchers in the field but it is quite common. 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. Also in an integration process a new ontology is created, but here it is easy to see what parts come from the different source ontologies since they handle different subject areas.

Normally the result of combining ontologies is a new ontology, but not always. It can also be just the mappings between concepts and relations etc, in the different ontologies, like in [43]. Mapping could here be synonymous to matching. These mappings can then be a starting point for alignment, that is: 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 in [41], the ONION system in [63] or the OBSERVER system in [59]). From this point the process can then be taken one step further and a merging or integration can be performed.

There are further concepts that need to be clarified, for example the distinction between merging and fusion. Fusion can be seen as a specific way to merge ontologies, where the individual concepts loose or change their previous identity. That is, new concepts may be introduced and concepts may be modified, whereas in ontology merging in general the concepts from the source ontologies are kept intact and only rearranged to form a new ontology.
In [46] there is a nice overview with definitions of merging, integration, alignment, mapping and a few other concepts, which conform to the definitions given here, but it needs to be noted that not all researchers accept these definitions and there exist many variations.

2.4 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 the time and effort needed to perform it (see for example [95] or [7]). One way to facilitate reuse is by using patterns, as described in section 2.5, but there are many other aspects to reuse that are not very often discussed in a formal way. A thorough discussion about reuse can be found in [95], 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 obstacles that a reuse methodology has to overcome is first of all 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. There is also need for a motivation to share your ideas without getting any direct benefit from it.

In [95] the reuse process is divided into two different steps: ”design for reuse” and ”design by reuse”. 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 2.

What to reuse is also an interesting question because reuse can be exploited on different levels. For example, reuse in Software Engineering can, according to [95], be done at at least three stages (or levels):

- Requirements Reuse (for example models of the domain or generic models of the requirement specification).
- Design Reuse (for example models, data structures and algorithms).
- Software Component Reuse (for example reusable software classes or editable source code).

Reuse can also be divided into categories by looking at what the developer knows of the reusable objects (see [95]). 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 to the developer, he or she only has 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 developer and he or she 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, it will obviously not be possible in the case of knowledge reuse, such as reuse of ontologies. As stated in [95] chapter 2.1, in order to reuse knowledge it has first to be understood.
2.5 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 maybe from one application to another. This is something we all use in our daily lives and in our professions as well. We seldom invent completely new solutions, since problems often resemble other problems we have encountered before, we reuse parts of the solutions, we use old solutions as patterns. We also recognise patterns in our surroundings that can be useful.

More formalised patterns, that can be recognised and used by a whole community of people have been used in several fields, like Architecture, Economics and also Computer Science. The most popular patterns in Computer Science are Software Patterns. Ideas for these patterns came from the architecture field already in the 70’s and the probably most well-known book in the Software Pattern community is the book on Design Patterns written by Gamma et al in 1995 [28]. These are patterns describing common design solutions in Object Oriented Software Engineering. Generally this kind of patterns can assist on the following issues (see [7]):

- 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. In [7] it is also stated 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. These points have to be presented in a formalised way so that the patterns can be communicated to a community of people. Often the description of a Software Pattern consist of (see [7] and [28]):
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. The level of formality and how to express the different parts in the list may vary, but the structure could be the same. For example, how to illustrate the structure of a pattern may vary greatly between different kinds of patterns. Software Design Patterns are normally described using the Unified Modelling Language (UML) class diagrams, while the structure of a Knowledge Pattern (see chapter 5.1.5) may be just a natural language text.

The previous discussion in this section deals with patterns used for construction. Another kind of patterns are patterns used to discover regularities in some way, in the field of Pattern Recognition. In this case it is the patterns themselves that are the product of the process, or at least what the patterns in turn can tell the researcher about the object of study. This can be things like Image Recognition in Computer Graphics, Signature Recognition in Computer Security, discovery of toxic compounds in a database of chemical substances or discovery of substructures, for rendering the code more efficient in software implementations etc.
3 Ontology Learning

Semi-automatic Ontology Construction, or Ontology Learning, is a quite new field in ontology research. Many researchers have realised that building ontologies from scratch, and by hand so to speak, is too resource demanding and too time consuming. In most cases there are already different knowledge sources that can be incorporated in the Ontology Engineering process. Such existing knowledge sources can be documents, databases, taxonomies, web sites, applications and other things. The question is how to extract the knowledge incorporated in these sources automatically, or at least semi-automatically, and reformulate it into an ontology.

To solve this problem the researchers in the field have come up with varying solutions. The name Ontology Learning refers to the fact that most solutions build on some learning mechanism similar to those in the area of Machine Learning, in the field of Artificial Intelligence. The approaches have some differences in choice of solutions and both in choice of input and output, but mainly the differences lie in the different aims. Some are aimed at linguistical ontologies only, structuring a language with ontologies, while others are more aimed at general domain ontologies. It is the latter kind that is most interesting in the scope of this report.

3.1 Semi-automatic Approaches

The majority of these approaches build on user intervention together with some form of Data Mining techniques and Machine Learning, like Pattern Recognition. In the next section, some common base concepts for these systems are described and then in the following sections the specific systems are discussed.

3.1.1 Basic Methods

Many of the semi-automatic approaches rely on basic techniques that have been developed for other purposes, in other research fields. The techniques have often originated in the Data Mining or Text Mining areas, or for other parts of the systems in Machine Learning. Some of the most important basic ideas are presented here in brief.

Term Extraction

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 specify that the texts has to be from only one specific domain others deal with a more general corpus of texts, but almost all start with extracting terms
according to their frequency of occurrence in the texts. Some use algorithms (see for example [65]) to filter out words that are common in all texts, and thus not domain specific, concepts that are only used in one single document (and not in the whole corpus) and of course terms that are simply not frequent enough to be of importance. Another approach to term extraction is to use preexisting Linguistic Patterns to extract terms that are part of important linguistic constructs (see for example [102]).

**Word Classification**

A task that is common to all ontology extraction approaches is the task of extracting a taxonomy for the ontology. There are several approaches to classifying words, and some can also be used to extract more arbitrary relations. One system that attempts to classify words according to their context is SVETLAN (see [12]). The input is texts automatically divided into so called Thematic Units, that is 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. The nouns are also weighted to eliminate nouns from classes where they are not very relevant. In [27] a similar method is used. Sentences are parsed, syntactic dependencies extracted, which can then be translated into semantic relations using predefined Interpretation Rules.

Another such system is Camille (see [101]) 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 actually arrange the terms into a hierarchy an approach using a Galois Lattice (or concept lattice) is proposed in [85]. This approach takes an unstructured set of terms as input, together with attributes and attribute values that are attached to the terms, which might have been automatically extracted from texts. The terms are arranged in a hierarchy according to these attributes and their values and finally the lattice is pruned to minimize redundancy in the hierarchy. This kind of arranging of concepts can also be done using a clustering algorithms, like in [18], which will be discussed further later on.

These approaches, and others similar to them, are used in most systems presented later on in this document. In some suggested systems the user is required to validate the hierarchy during the process while other approaches are more or less automatic.

**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 important approaches is to use Co-occurrence Theory or Association Rule Mining to derive possible relations in an ontology. This has been discussed for example in [86], [15], [39] and [9].
The basic idea of Co-occurrence Theory (or Collocation Theory) is that if two concepts often occur near each other they probably have some relationship. Some approaches, like [39], only look at co-occurrence in the same sentence but it could also be in the same textual environment in general, for example the same document. Association Rules are just an extension of Co-occurrence Theory, then the co-occurrences are formulated as rules. For example a rule could be that if a document contains the terms \( t_1, \ldots, t_i \) then it also tends to contain the terms \( t_{i+1}, \ldots, t_n \) (see [9]). These rules can then be used to extract more relations while at the same time also updating the rules themselves.

3.1.2 SOAT

The first general system to be discussed is named SOAT (see [102]). This system is one of the most automated ontology construction environments. Using templates of part-of-speech tagged phrase structures, in Chinese, the process of collecting concepts and relations from texts is performed. For this a text corpus from the domain is needed and a seed word, for example the name of the domain. Ideally the output is a prototype of the domain ontology, only to be validated by a human user, but the extraction rules might need to be updated during the process and that can only be done by the user.

Four different relations between concepts are extracted; category, synonym, attribute and event. Category roughly corresponds to a taxonomy. Event is the actions that can be associated with a concept, in [102] the authors explain this by saying 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 keyword, 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 is that it is highly automated and apparently very efficient (see [102]). Disadvantages are that it requires very heavy preparations and adaptations, in the form of constructing correct extraction rules. It is also quite static in that it can only extract predefined types of relations between concepts, types that can be detected with the extraction rules.

### 3.1.3 Text-To-Onto

A fairly elaborate approach is the Text-To-Onto system developed at the University of Karlsruhe (see for example [53], [56], [54], [52] and [55]). 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 system is based on the following parts:

1. Data Import and Processing
2. Natural Language Processing System
3. Algorithm Library
4. Result Presentation
5. Ontology Engineering Environment

The idea of the first part, Data Import and Processing, is to be able to use almost anything as input to the system. There may be existing knowledge sources like databases, web pages, documents or even already existing ontologies present. All these sources are then pre-processed into an appropriate format, and possible structure information is retrieved. The resulting texts are then used as text corpus for the rest of the process. The next step, which is Natural Language Processing, uses a shallow text processor which performs different levels of parsing on the texts.

The result from the parsing process is used in the third step by the algorithms in the Algorithm Library. For the task of Ontology Extraction there are two kinds of algorithms included in the 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 lexical 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 almost like Ontology Extraction 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 OntoEdit, which lets the user manually edit the extracted ontology.

3.1.4 OntoLearn

The OntoLearn system, from the Universitá di Roma La Sapienza, uses a slightly different approach (see [65]). This approach builds on the general linguistic ontology WordNet (see [61]), for English language, and the SemCor knowledge base (containing tagged sentences), to interpret terms found in the extraction process. The resulting ontology is a pruned and refined part of WordNet.

The approach is much 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 also part of a bigger architecture, for editing, validating, 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. 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. A term may be frequent but only in a specific document, then it is still not representative for the domain.
The second phase performs Semantic Interpretation which in turn is composed of 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 is included (up to a certain distance in the graph). Also 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 finally taxonomic relations are 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 are integrated, either with an existing domain ontology or with WordNet, which is then pruned of irrelevant concepts.

This approach is highly specialised towards language applications and one of its major drawbacks is that many things 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 many pre-existing structures, like WordNet and SemCor, which then are language dependent. An advantage is that it is highly automated and performs most tasks without any user intervention.

3.1.5 TextStorm

A similar application that also uses the top-level ontology WordNet is the TextStorm system presented in [73]. Another system, Clouds, is also presented in the same paper, it continues where TextStorm leaves off 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 a relation between two terms, extracted from a sentence. The synsets of WordNet are used to avoid extracting the same concept (only denoted by a different word) several times. The output of TextStorm is not a finished ontology, or concept map as the authors choose to call it, but only these predicates. It is the Clouds system that does the actual building of the map, or 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.
3.1.6 ASIUM

The system ASIUM (see [19] and [18]) is quite similar to the previously described OntoLearn system in the respect that this is also a system directed at linguistic applications and the system uses linguistic patterns and Machine Learning, but it does not rely on any predefined structures such as WordNet. The specific novelty of this approach is the conceptual clustering algorithm used to generate the structure of the ontology. In addition to this system another system has also been suggested, the Mo’K workbench (see [4]), 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 can be said to be performed repeatedly and in parallel):

1. Extraction of Instantiated Syntactic Frames
2. Frame Learning and Clustering
3. Validation

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

Then a learning algorithm is applied, with the basic assumption that "words occurring together after the same preposition and with the same verbs represent the same concept" [19]. Frame instances are extracted and new frames are learned at the same time. A frequency measure is also used to determine which 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. Then different clustering algorithms can be applied and different similarity measures for the clustering algorithms may 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", that is: 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, the idea of frames with verbs and prepositions may not work as well for all languages, for example. Another weakness is the output, which in this case is restricted to a
"pyramidal" tree, only consisting of a taxonomy basically. Also the relevance of the similarity measure can be questioned. Advantages are that the user gets to validate 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 should produce a better result.

3.1.7 Adaptiva

The Adaptiva system in [6], is yet another system based on linguistic patterns and Machine Learning, but with a more iterative and cooperative approach than the previously described systems. In [6] the authors explicitly consider what the system is to be used for and who the users are, and states 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 then divided into two stages which each consists 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 probably 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.
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 nodes, relabel nodes and relations and ask the learner to find all relations between two given nodes.

One of the stated advantages with this methodology 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 indeed is a good point. Other advantages are 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 could 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.

3.1.8 SYNDIKATE

This system (see [37], [35] and [36]) is more directed towards evolution and maintenance of ontologies, and other knowledge sources, than to construct an ontology from scratch. The system also has several uses, not only ontology construction, but grammar learning and other tasks connected to Natural Language Processing.

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. Then different quality labels are attached to the hypothesis. One is Linguistic Quality, which is a measure of how well the hypothesis conform 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 in [35], 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 the only approach present that uses 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 too resource demanding. 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 than a finished ontology.
Ontology Enrichment

Ontology Enrichment is actually Ontology Learning, only with a special precondition: it demands an ontology to already be present at the beginning. Some of the already discussed systems could also be used for this purpose, so these areas are actually just two views of the same problem.

One specific Ontology Enrichment approach is proposed in [17] 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 authors then try 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 Metric and an Information Content Metric, is described in [89].

Ontology Learning Summary

All these approaches build on roughly the same foundations, in Natural Language Processing, Data Mining and Machine Learning. Some approaches have specific and novel algorithms but mainly it is the same basic techniques. Most approaches consider concepts and taxonomic relations in the extraction process, some consider arbitrary relations, but none considers more advanced parts, like axioms or constraints. 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 instead to what extent a user is assisted in this task.

A characteristic of all the approaches is that they only extract single concepts and binary relations, only a few consider arbitrary relations and none considers axioms or constraints. All of the approaches also consider the concepts and relations one by one, some try to extract a taxonomy or arbitrary relations, but mostly the user has to validate single concepts and relations and not suggestions of finished ontology parts. Some of the systems use patterns in the extraction process but those patterns are only linguistic patterns for extracting certain types of relations for example.


4 Ontology Reuse

Reuse is commonly seen as a self-evident way of improving your work, both because it means using existing and well-tested solutions and because it generally shortens the development time and cost, whatever the application. In Ontology Engineering there are two different viewpoints. The first states that ontologies are so domain and application specific that almost nothing can ever be reused for another application case, so integration of existing ontologies is not a part of the development methodology (see for example [94] and [97]). The other extreme states that Ontology Engineering is almost impossible to do "from scratch" and therefore the ontology construction process is always just a process of combining already existing parts of ontologies and other knowledge sources (see for example [24] and [51]). Most researchers would put themselves somewhere in the middle of this scale where the idea is to find existing ontologies that might fit the application case and then combine, adapt and extend them, but also to build parts of the ontology "from scratch" if needed.

There are several problems in this area. Although many researchers agree that this is a very important activity in Ontology Engineering, almost no research has been done on how to actually perform the process, very much like what is said to be the problems of reuse in Software Engineering (in [95]). How does one find the appropriate ontologies to reuse? Even with ontology libraries present today, 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 somewhat discussed and also different methodologies for alignment, merging, integration etc on the concept level exist (see for example [80], [60] and [75]).

The first two questions though (how to find and to choose the right parts), have so far almost not been touched at all when it comes to ontologies. These issues also pose a problem in other areas, but there some solutions have also been proposed, for example as described in [95], chapter 1.10, concerning Software Engineering. There might also be possibilities to draw analogies from an area close to ontology reuse, reuse of Problem-solving Methods, where for example [20] has made an effort in this direction. By using approaches for ontology matching to search an ontology or an ontology library (similarly to the match-making in [3], the conceptual graph matching of [104], semantic matching in [103] for example, or what is suggested in general terms in [95]).

First of all there is a quite old approach, developed in 1995 (see [76]), which states methods and shows an implementation (the KARO system) of a method 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, which are generally domain-independent but might not be totally task-independent
(therefore the adaptation). It is this assumption that renders the approach more or less irrelevant, since the focus of this report is on Domain and Application Ontologies and not Top-level Ontologies. Despite this, a few good general points are stated in the article, that also concur with the recommendations for Software Engineering in [95]:

- 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.

Another similar paper is [99] which aims to show that ontologies are highly reusable within one domain, but this assumes very high-level ontologies which are again not very useful within the scope of this document. The continuation of this chapter presents some of the few existing approaches to ontology reuse.

### 4.1 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 section 5) in for example Software Engineering. In the area of ontology reuse so far there has been only a few attempts in this direction. One idea, using a library of generic concepts, for building ontologies, is presented in [1]. 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 [1].

This idea is similar to the many attempts at creating general top-level ontologies, like WordNet (see [61]), CYC (see [50]) and SUMO (see [67] and [74]) for example. It is also similar to the attempts involving Semantic Patterns for ontologies described further in section 5.1.4. The problem is that so far these ”general concepts” has either been too generic, like top-level ontologies, or too simple, like meta-language primitives. Nothing exists in-between.
4.2 Modularisation

The idea of modularisation has been fruitful for example in the Software Engineering field, and might be viewed as a step towards applying patterns for reuse. This could also be a step towards easier reuse of parts of ontologies, just like in reuse of software components (see for example [95]). The WonderWeb initiative have been working in this area, developing a methodology for ontology modularisation (see [90]). This methodology is mainly adapted for ontologies expressed with Description Logics, for example the ontology language OWL. The author states three types of requirements that should hold for ontology modules:

1. Loose Coupling, which means that often nothing can be assumed about different modules, they might have very little in common (like concepts or 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 using three techniques in [90]:

1. View-based Mappings, which means that the different modules are connected through conjunctive queries, the extension of a concept in one module can be said to be equivalent to the answer to a conjunctive query to another module. This is also used for arbitrary relations between concepts.

2. Interface Compilation, which means that in order to provide self-containment using these View-based Mappings the result of the query is then computed off-line and added as an axiom in the querying module, thus enabling local reasoning.

3. Change Detection and Automated Update, 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.
An advantage of this approach seems to be that the modules really are self-contained and this approach might work very well for ontologies modelled in Description Logics and used for reasoning. The approach is very specific however, and might not be feasible when dealing with ontologies implemented using other types of representations and there is 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.

Another modularisation approach can be found in [78], also only concerning ontologies represented in Description Logics. The author states an extensive set of rules on how to develop and represent an ontology to achieve explicitness and modularity. Here the modules are distinguished by the notion that all differentiating notions of the taxonomy of one module must be of the same sort, for example 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.

Both these approaches restrict the Ontology Engineer substantially, but despite all this ontology modularisation could be one step towards more structured ontology reuse. There are still questions however; the modules must be defined more precisely and also described for other types of ontologies, that might not be implemented in Description Logics and not used for reasoning tasks.

4.3 Ontology Matching, Integration and Merging

This is one of the areas in ontology reuse that is receiving quite a lot of attention by researchers today. It involves (possibly language dependent) techniques, often implemented as a part of an Ontology Engineering environment, to find similarities between ontologies, to find matches between them and also to possibly align, integrate or merge them. There are manual methodologies stating how to perform matching, alignment and merging, as in [75] and [51], but these are as cumbersome and resource demanding as manual methodologies for Ontology Engineering and thereby not very interesting in the focus of this report. The interesting methods are the semi-automatic approaches that, like the Ontology Learning systems, assist the user in performing these tasks.
One such system is GLUE that uses Machine Learning algorithms to find the most probable matches for concepts in two ontologies (see [16]). The system uses several similarity measures and several learning algorithms to perform the matching process, but it only considers single concept matchings between the ontologies. FCA-Merge is another approach, presented in [93]. The approach is based on Formal Concept Analysis and is, as the name states, aimed at merging ontologies not only to find mappings. The main idea is to use a concept lattice, together with instances of the concepts present in texts, to extract formal contexts for the ontologies, determine similarities and then to assist the user of the system in building a merged ontology. Another approach resting on Formal Concept Analysis is presented in [47].

IF-Map (see [43]) is yet another approach, this time based on Information-flow Theory, which generates (with assistance of the user) a merged ontology. A similarly well-founded approach is described in [81]. Here the author argues that the duality in knowledge sharing makes other proposed attempts of ontology alignment insufficient. The mathematical foundations for his theories are Chu spaces, which he states will allow a reliable modelling of knowledge sharing scenarios. This is a more abstract approach, but does not consider anything but single concepts and the hierarchy connecting them. Several other approaches exist, but most of them have in common that they only match individual concepts and do not really take into account the structure of the ontology or the axioms or constraints that might exist.

Two systems, that in some sense do take more into account, is the Chimaera system (see [58]) and the PROMPT-suite for Protégé (see [70], [71] and [69]). In Chimaera the merging or integration process is done in cooperation with the user. Name resolution lists are presented to the user to decide which concepts are to be merged or connected in some way, also taxonomy resolution is performed by looking at the taxonomic structures of the ontologies. PROMPT uses a similar approach, with the difference that the Anchor-PROMPT algorithm also takes into account that the intermediate concepts in a hierarchy, where two concepts already have been matched, also tend to have their corresponding concepts in the other ontology. This matching is mainly done by using heuristics.

There are numerous approaches to automate the matching, alignment, integration or merging of ontologies, but most of them have so far only taken into account the concepts and single taxonomic relations. Some also deal with attributes and more advanced structural properties of the ontology, but it is still very much up to the user to build the final ontology. The closest any approach comes to matching whole parts of ontologies is the approach of the Anchor-PROMPT algorithm, but this is still not very elaborate. None uses graph pattern-matching to find similarities in ontologies.
Approaches that do use this kind of pattern-matching are more related to Information Retrieval than ontologies (see for example [3], [104] and [103]). There are several approaches using so called Semantic Matching, or Conceptual Graph Matching, that takes into account larger parts of a graph structure. These approaches might, in the future, be adapted to ontology reuse, in order to search ontology libraries to find ontologies, and parts of ontologies, for reuse.

4.4 Ontology Reuse Summary

On the lowest level, concerning ontology languages, there are several approaches for translating and connecting ontologies, for example some using meta-languages. On the concept level there also exist approaches to compare and combine ontologies. The problems with these approaches are that very few of them take into account more than the labels of the concepts, and possibly relations, present in the ontologies. They do for example not compare the structure of relations, attributes, axioms and constraints. Most do not consider larger parts of the ontology structure than single relations, even though the structure in itself might say a lot about the ontology.

Another important part of reuse is to find ontologies, or parts of ontologies, to reuse. Today there exist many ontology libraries, containing various ontologies in various formats, and also generic Top-level Ontologies that can be reused. Still, it is up to the Ontology Engineer to decide which ontologies to consider and what parts to reuse, maybe simply by looking at the ontologies. Often there is no way of searching the libraries, except maybe by querying the name of an ontology, and similarly there seldom exist any description or specification of the ontologies, so to understand it the Ontology Engineer has to look at it graphically and study its concepts and relations.

Although some attempts at modularisation of ontologies have been made they are only very specific attempts, involving ontologies in Description Logics used for reasoning. There is no more general effort made and no specifications of what a module should or should not contain and how to adapt modules at the time of reuse. ontology reuse has to be established more firmly as a part of Ontology Engineering, and most of all it needs to be studied and specified more formally.
5 Patterns

There are two different views of patterns. The first kind of patterns aims at reuse of knowledge or readily implemented parts. The patterns are the means for reuse and aims at facilitating a construction process, like Software Engineering or architectural planning of a building. The second kind represents pattern-like structures discovered in existing implementations etc, through Pattern Recognition techniques. In the latter case the patterns themselves are the goal, finding regularities in graph structures for example, or finding common code usage to render the code more efficient.

When considering the ontology area, the first kind of patterns might be adapted to fit ontologies and be used in the same way there, for Ontology Engineering. Also the second kind might be transferred to ontologies and the techniques could be used to discover ontology similarities or possibly reusable parts in existing ontologies. To get that far though, first a study of the existing pattern communities in Computer Science is needed.

In order to study the existing communities within Computer Science they first have to be structured and categorised somehow. One way to divide them is to look at the different application areas for their patterns, and what the practitioners themselves have chosen as names of their patterns. Patterns are used in areas like Software Engineering, Data and Knowledge Modelling, Programming Languages, general Linguistics and Graph Theory (in turn used for Bioinformatics, Computer Graphics and Computer Security etc). Patterns can also be defined at different levels of abstraction. These two different categorisations are described further in the following sections.

5.1 Pattern Usage Areas

Patterns have, as stated earlier, been adopted by many areas of Computer Science and neighbouring sciences. In this section the usage areas are divided into groups with similar usage, or naming, of patterns and pattern categories. The list is not complete but may be seen as examples of pattern usage today.

5.1.1 Software Patterns

The most well-known kind of patterns in Computer Science today are no doubt Software Patterns. No structured software development project using an Object-oriented paradigm is conducted without the use of some kind of patterns. Patterns vouch for reusability, product quality and managing complexity of the system developed. These patterns are often divided into different kinds, according to when, in the development process, they are used and what level of granularity they operate on. The most common categories are:
• Analysis Patterns (see [25], [22] and [48])
• Architecture Patterns (see [82], [26], [30] and [7])
• Design Patterns (see [28], [7] and [5])
• Programming Language Idioms (see [7])

Analysis Patterns are used in the very beginning of the Software Engineering process. These patterns reflect things like conceptual structures of business processes, for different business domains for example, 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 of course, but also for all records where there is need to keep track of not only the current value but also details of each change that affects this value (see [25]). The pattern could be illustrated using UML notation, see Figure 3, where the entries record each change. Similar to Fowler’s Analysis Patterns in [25] are the ones suggested by Fernandez and Yuan in [22].

![Figure 3: The Account Analysis Pattern. [25]](image_url)

At another level of abstraction 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 in [48]. An example is the Joint Venture Pattern which represents the situation when several enterprises join together to perform a specific task which is coordinated by a joint management. An illustration of this pattern can be seen in Figure 4 (the authors of [48] use 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).

Architecture Patterns show overall structuring principles for software systems, how to divide them into subsystems, the responsibilities of the subsystems and their relations (see [7]). Even though the authors of [30] name their patterns Analysis Patterns they present detailed descriptions of software architecture, similar to those in [82], and can thereby also be classified as Architecture Patterns.
An example of an Architecture Pattern is the Layered Architecture Pattern, as described in [82]. This pattern illustrates a type of architecture, or model, suitable for systems that involve distinct services that can be classified hierarchically, see Figure 5. Usually the interaction between layers are procedure calls. The layers are then further decomposed using other patterns.

A slightly different kind of Architecture Patterns is presented in [26], by Fowler. These are not as high-level as the Architectural Patterns presented by Shaw in [82], but most are not as detailed as the Design Patterns of [28]. They resemble more a further development of Fowlers own Analysis Patterns in [25]. The patterns are specific for building Enterprise Applications.
An example of this kind of pattern is the Domain Model Pattern (see [26]), 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 6.

![Domain Model Pattern Diagram](image)

**Figure 6: The Domain Model Pattern. [26]**

When further developing the design 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 or paradigms (see [7]). 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 in Figure 7. There have also emerged Design Patterns specific to certain software domains, such as the Game Design Patterns in [5], describing certain interactions common when designing Computer Games.
Finally, Programming Language Idioms are language specific patterns describing ways to implement certain aspects of components, or interactions between them, using language specific features (see [7]). An example, also found in [7], is an idiom for implementing the Singleton Design Pattern in C++. The purpose of this pattern is to ensure that only one instance of a class exists at run-time. The idiom is depicted in Figure 8.

Figure 7: The Observer Design Pattern. [28]

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

Figure 8: Singleton Idiom in C++ from [7].
5.1.2 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-tested solutions to help modellers be more effective and increase the quality of the models. Data Modelling can of course also be a part of Software Engineering, so these two areas are strongly interconnected.

Data Model Patterns are presented in [38] by David Hay. 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, 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 draw from the presented Data Model Patterns, so in this respect they are quite similar to the previously described Architecture Patterns.

An example from [38] 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 9. Hay’s patterns have also been further developed in for example [84] and [83].

![Figure 9: Products and Services Data Model Pattern. [38]](image-url)
5.1.3 Linguistic Patterns

Linguistic Patterns are language dependent patterns that represent the structure of the language. Different constructs in a grammar could be such patterns. Linguistic Patterns are used in parsers, translators and also in systems for Semi-automatic Ontology Construction like in [102], [56], [19] and [35]. For example, in [19] linguistic patterns like:

\[
<\text{verb}>(<(\text{preposition})\text{|}\text{function})<\text{headword})
\]

are used to extract Instantiated Syntactic Frames. Such an extracted frame could look like this:

\[
<\text{totravel}> <\text{subject}> <\text{Bart}> <\text{by}> <\text{boat}>,
\]

(extracted from the sentence "Bart travels by boat").

5.1.4 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 away representation structure or 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 away hardware representation) for example, but let us restrict this term to denote those abstractions that are actually used as patterns.

Then one example of such patterns could be meta-languages. Meta-languages also abstract away the specifics of the actual languages, which can be programming languages, ontology languages etc. There are of course a large number of examples of such pattern areas but this is actually one area where there also exist Ontology Patterns (see [91] and [87]). In [87] these Ontology Patterns are named Semantic Patterns and aim to abstract away the specifics of ontology representation languages. In [91] the additional problem of language expressiveness is discussed, the Ontology Patterns need to be at least as expressive as the Ontology Languages they are used with.

An example of a Semantic Pattern, found in [87], 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 (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 like RDF (Resource Description Framework) everything is defined globally and relations exist independently of the concepts they belong to. In the notation of [87] this pattern, or meta-language primitive, is written

\[
LOCAL\text{INVERSE}(r_1,c_1,r_2,c_2),
\]

with \(r_1, r_2\) denoting binary relations and \(c_1, c_2\) their ranges.
5.1.5 Knowledge Patterns

The term Knowledge Pattern denotes different things depending on the community. In the more business related parts of the Knowledge Management community, a Knowledge Pattern could mean how to structure the management of knowledge within a company, by for example appointing teams of knowledge leaders or harvesting knowledge from workers (see [11]).

In the more technically oriented areas of Knowledge Management, the term Knowledge Pattern denotes patterns for building Knowledge Bases (see [10]) or reusing Problem-solving Methods (see [77]). The Knowledge Patterns in [10] are quite specific and are more related to Semantic Patterns than for example to Architecture Patterns. An example of one part of such a pattern can be seen in Figure 10 (expressed using Prolog notation). This part, or axiom, represents the concept of how free space in a container can be calculated. The whole 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.

\[
\text{free\_space(Container, } F \text{)} : = \\
\text{isa(Container, container),} \\
\text{capacity(container, } C \text{),} \\
\text{occupied\_space(Container, } O \text{),} \\
F \text{ is } C - O.
\]

Figure 10: The Free Space Axiom in the Container Knowledge Pattern. [10]

The Knowledge Patterns in [77] on the other hand lie somewhere between Software Architecture Patterns and Design Patterns, but developed 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 11.

Another category of Knowledge Patterns can be found in [95]. 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. The differences lie mainly in the scope of the approaches. [95] is a more theoretically well-founded approach and also covers more aspects of reuse, for example 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 12. 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 11: The Heuristic Decision Tree Pattern, selection of a problem area. [77]

Figure 12: The Object Inventory OSM. [95]
5.1.6 Pattern Recognition

This kind of patterns mainly belong to the second category mentioned in the beginning of this chapter. That is, 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 Recognition field. To this field could also be classified parts of the area described in section 5.1.3, concerning Linguistic Patterns. To find the Linguistic Patterns, Pattern Recognition techniques could be used. This is also true for some cases of Programming Language Idioms and Semantic Patterns.

The idea of the Pattern Recognition field is to find regularities, and thereby patterns. Whether these patterns then are used for some other purpose, like comparison to other patterns, Machine Learning, text parsing etc or if the mere fact that they have been found is enough, is very 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 or Computer Security areas. When it comes to chemical compounds, finding frequent substructures can mean finding toxic substances or simply determining similarities of compounds.

Examples of such subgraph discovery algorithms can be found in [42], [49] and [40]. 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 for example the discovered patterns or a compressed version of the original graph, where the patterns where used for the compression process).

Ontologies can also be used for pattern recognition tasks, this is mainly done in the Information Retrieval (IR) area. 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 (see [104]) means to exploit not only the exact keyword entered for the search but also the intended meaning of that keyword. This is one thing that can be done using ontologies. To then perform Semantic Search or other forms of IR many different approaches have developed, for example graph-matching algorithms to find ”semantic matches” (like in [3] and [103]).

5.2 Pattern Levels with Applications to Ontologies

Another way of structuring different patterns is to divided them by their level of abstraction. There may be many ways to characterise the level of abstraction but here five levels have been chosen, which roughly correspond to the levels that can for example be found among Software Patterns.

5.2.1 Syntactic Patterns

A Syntactic Pattern is a re-occurring combination of representation symbols. In this category the following previously discussed patterns can be placed:

- Linguistic Patterns (as described in section 5.1.3).
- Programming Language Idioms (as described in section 5.1.1).

When considering this level of patterns for ontologies it could correspond to both items on the list above. Ontologies are implemented using a representation language, like RDF, DAML+OIL or OWL for example. These representation languages have their own common constructs, or to put it in Software Engineering terms: their own Programming Language Idioms. Such idioms, or Ontology Design Patterns as some researchers choose to call them, are the subject of a W3C (World Wide Web Consortium) task force, working within the Semantic Web Best Practices and Deployment Working Group. The group tries to develop patterns for common constructs in OWL (see [100]). An illustration of such a pattern representing one way to express n-ary relations in OWL, found in [68], can be seen in Figure 13. The illustration shows an implementation of this pattern, using the example sentence ”Anne has lung cancer with high probability”. Other types of Syntactic Patterns are today mainly connected to ontologies in the respect that they can be used to extract ontologies from other representations, which may be viewed as a kind of translation of one representation into another.
5.2.2 Semantic Patterns

A Semantic Pattern is a commonly represented concept but independent of the representation language. The meaning of a certain idiom belonging to the Syntactic Pattern level could in turn constitute a Semantic Pattern. In this category the following patterns can be placed:

- Meta-languages in general.
- Modelling languages, like UML, describing representation independent modelling primitives.
- Semantic Patterns for ontologies (as described in section 5.1.4).
- The Knowledge Patterns of Clark et al in [10] (as described in section 5.1.5).

In the area of ontologies, this kind of patterns has already been suggested, for example in [91] and [87]. The aim is to create a logic meta-language for ontologies by providing a full-covering catalogue of Semantic Patterns. The example pattern Locally Inverse Relation, as considered already in section 5.1.4, can be described more in detail as in Figure 14 (see also [87] for an even more detailed description). When instead considering the ontology as a graph of concepts and relations, Semantic Patterns simply describe the individual objects (concepts, relations, axioms, constraints, functions etc).
Pattern Name | Locally Inverse Relation  
Intent | Allows to define inverses the scope of which is restricted to particular concepts.  
Also Known As | Restricted inverse relation  
Motivation | Often the definition of global inverses is too general and yields overly general inferences, with unwanted consequences. For this reason this pattern allows the definition of inverse properties restricting their domain and range concepts. This notion of locality is naturally given in OO systems, where properties are defined locally in classes. It is not given in RDF(S) where properties are first-class citizens and exist independent of classes.  
Structure |  
Namespace | http://ontobroker.aifb.uni-karlsruhe.de/schema/LocalInverse.rdf  
Epistemic Primitive(s) | LOCALINVERSE  
Signature | LOCALINVERSE(r₁, c₁, r₂, c₂) with r₁, r₂ denoting binary relations and c₁, c₂ denoting their corresponding ranges.  
Known Uses | http://ka2portal.aifb.uni-karlsruhe.de  
Related Patterns | The pattern “globally inverse relation” subsumes “locally inverse relation” when applied to the same relations.  
Constraints | C₁, in \[LOCALINVERSE(ISSHOWN, THEATRE, HOST, MOVIE), \] \[DOMAIN(ISSHOWN, MOVIE), \] \[RANGE(ISSHOWN, THEATRE), \] \[DOMAIN(HOST, THEATRE), R ange(HOST, EVENT), \] \[TYPE(Lassie, MOVIE), HOST(Schauburg,Lassie) \]  
C₂, out \[IS GIV EN(Lassie,Schauburg) \]  
C₃, pout \[TYPE(Schauburg,THEATRE) \]  
C₄, nout \[IS GIV EN(Lassie,Schauburg), TYPE(Lassie,PLAY) \]  

Figure 14: The Semantic Pattern of Locally Inverse Relations. [87]

5.2.3 Design Patterns

A Design Pattern is a small set of Semantic Patterns combined so that they form a commonly used, and reusable, combination (for construction). To this category of patterns belong:

- Analysis and Design Patterns in Software Engineering (including the Architecture Patterns in [26] but excluding the Analysis Patterns in [48], as described in section 5.1.1).
- Selected parts of the Data Model Pattern area (as described in section 5.1.2).

In the ontology context this would be a small connected collection of Semantic Patterns for ontologies. That is, a few connected primitives like concepts, relations, axioms and constraints. If an ontology is represented as a graph the implementation of such a pattern would constitute a small subgraph, with its accompanying axioms, constraints and other functions. Very similar to Software Design Patterns are the Ontology Design Patterns that actually already exist, developed by the Laboratory of Applied Ontology [29] and more generally in Molecular Biology [79]. These are
very generic and express things like encapsulation of concept definitions, how to keep track of updates between different partitions of an ontology or how to structure a terminological hierarchy. A somewhat simplified example of such a pattern, from [29], is depicted in Figure 15 (again using UML notation).

![Figure 15: The Dolce Descriptions and Situations Pattern. [29]](image)

**5.2.4 Architecture Patterns**

An Architecture Pattern describes the overall structure of the object in question, whether it is software, ontologies or buildings. Architecture Patterns show how to arrange the implemented Design Patterns, in order to complete the overall goal of the building process. To this category could be classified:

- Software Architecture Patterns (including the Analysis Patterns in [48], as described in section 5.1.1).
- The Knowledge Patterns of [77] (as described in section 5.1.5).
- Selected parts of the Data Model Pattern area (as described in section 5.1.2).
When considering ontologies, this level of patterns concerns the overall structure of an ontology and would for ontologies most likely resemble the Knowledge Patterns for Problem-solving Methods described in [77]. An ontology might be constructed as a tree of categories, the tree structure could be an example of an Architecture Pattern for ontologies. Other such patterns could be related to the idea of modularisation presented in section 4.2, especially the kind of modules and structures described in [78], where each module has its own specific structural properties.

An example of a potential Ontology Architecture Pattern could be an adaptation of one of Shaw’s Software Architecture Patterns, the Implicit Invocation Architectural Pattern in [82], together with the idea of ontology modules (as in [90]). If the objects or processes in Shaw’s pattern are replaced by ontology modules, this could be an Ontology Architecture Pattern, see Figure 16. The modules communicate through queries and there is a central structure tracking changes in the modules (not represented in the figure), as described in section 4.2.

![Figure 16: A possible Ontology Architecture Pattern.](image)

### 5.2.5 Application Patterns

Application Patterns are even more abstract patterns, concerning the purpose, overall scope and usage of an object. These patterns might not be very useful in a specific construction context, instead they are more concerned with the activities before and after the actual construction. They resemble mainly the following previously discussed pattern categories:

- The Knowledge Patterns in [11] (as described in section 5.1.5).
- Some of the more abstract Architecture Patterns in [82] (as described in section 5.1.1).
In the ontology area such Application Patterns could describe how to use an ontology for structuring search at a web site or how to use ontologies to describe the hotel booking domain. Application Patterns can describe the domain and task the ontology is intended for, how it should be applied and the interfaces it provides to other ontologies or systems. Another example of such a pattern could be how to use ontologies for decision-making support in a virtual enterprise, for which a model is described in [51] (see Figure 17).

Figure 17: A model of ontology usage. [51]

5.3 Patterns Summary

As can be noted, patterns are used in many different areas of Computer Science and with great success. When it comes to ontologies not much effort has so far been spent on studying patterns of different kinds. The categorisation presented in the previous section gives an overview of the patterns in Ontology Engineering, which can be depicted as in Figure 18. The separation of the levels can be seen as a difference in granularity. The pattern approaches that do exist are mainly present at the lower abstraction levels. The suggested patterns are also quite diverse when it comes to description styles etc and most of them would not be suitable for use in combination with an Ontology Learning approach.
Figure 18: The suggested classification of ontology patterns.
6 Outlook and Summary

When it comes to ontologies many of the areas discussed in previous sections are not very well researched. The first aspect discussed here is where these "white spots" in current research are and some ideas for further research are presented. Finally, a short summary is provided.

6.1 Future Research Possibilities

The first area of discussion in this report was Ontology Learning. It is an area which is quite new but developing fast. So far the main focus has been on helping the user to extract important concepts and simple relations. Further development, according to many of the researchers in the area, will be to extract more complex relations and also more complex features of the ontology (like attributes, axioms and constraints).

Right now the expected input to most systems is plain text. Another future research area is to accept other input formats, like databases, and to take automatically advantage of the structure information contained in those sources. Most systems today do not construct the final ontology automatically, for example naming of relations is often required to be done by hand. These are also issues that might be solved in future systems.

In the area of ontology reuse there is also much to be done. This is not an as explicit research field as Ontology Learning, it is more seen as a bi-product of Ontology Engineering in general and thereby it might get a little bit overlooked at times. On the language level there are lots of approaches for translating ontologies, some using meta-languages for example, and on the concept level there are approaches to assist in comparing and combining ontologies. Very few of the systems take into account anything but the concepts of the ontologies though. Future development will be to compare relations, attributes, axioms (maybe by treating the axioms like objects, as suggested in [88]), constraints etc and also to consider larger parts of the ontology structure, than only single concepts and relations.

Another important part of reuse is to find ontologies, or parts of ontologies, to reuse. There are already today ontology libraries and Top-level Ontologies that can be reused. Still, it is up to the Ontology Engineer to decide which ontologies to consider and what parts to reuse, often simply by looking at the ontologies. Here a lot of assistance could be developed, like searching among ontologies, comparing ontologies etc, by for example using techniques like in [3], [104] and [103] (from the Information Retrieval field) or other approaches (maybe originating in mathematical graph-matching).
Although some attempts at modularisation of ontologies have been made they are only language specific attempts. It is a long way to go in order to have off-the-shelf ontology components ready and waiting. Today a problem with reusing an ontology can often be that the developer has no way of knowing what parts can be discarded and how the different parts depend on each other.

Patterns are, as noted in section 5, used in a wide variety of areas. Many of these areas are comparably old and well-researched fields, such as Software Patterns. Of course there are always improvements to be made, but no obvious "white spots" exist in the field. If the focus is instead on ontology patterns the field is almost not explored at all. There are only the Syntactic- and Semantic-level patterns of ontology meta-languages, the W3C’s Ontology Design Patterns, the Semantic Patterns in [10] and [87], and finally the very generic Ontology Design Patterns of [29] and [79]. Research on the higher abstraction levels is almost non-existing.

In an interesting article from 1999 (see [13]), Vladan Devedzic compares ontologies with Software Patterns. He argues that the two concepts have a great overlap and only differ in the details of their aim, generality and practical use. Both are intended for knowledge sharing and knowledge reuse, and both involve hierarchies of concepts, relationships, vocabularies and constraints. Ontologies as well as Software Patterns can be thought of as using an Object-oriented paradigm. A specific section of the paper describes the situation very well:

Referring to software patterns again, we can treat them as incomplete specifications of software engineering ontologies. They can also be part of other ontologies, and they can be sources of knowledge for developing other ontologies.

This relationship between ontologies and Software Patterns, and other patterns used in Computer Science, comes quite naturally when looking at the examples of section 5. Even the graphical representation of for example an Analysis Pattern resembles the graph representation of an ontology. Further research in the area of ontology patterns could be to explore these similarities on different levels of abstraction and to develop similar patterns for ontologies, or simply adapting and reusing some existing patterns from another area. In another paper (see [14]), Devedzic continues to explore the fact that Ontology Engineering and Software Engineering are very closely related. He gives several suggestions of areas (like development methodologies, patterns, component-based engineering etc) where the two areas would benefit from further exchange of methods and experiences.

The main aim of this report is to show how the three areas discussed earlier can be brought together and I think it is here that the most interesting aspects of future research can be found. If the concept of ontology patterns can be defined at different levels of abstraction then the continuation of the study could investigate how to apply this for Ontology Learning and ontology reuse.
The possibilities of ontology patterns in Ontology Learning are manifold. Design Patterns for ontologies could be used as building blocks by the Ontology Learning systems, either automatically or semi-automatically. By for example using whole patterns when enough parts of the pattern have been detected, or by suggesting possible patterns, that might fit the application case. Architecture Patterns for ontologies could then provide the overall structure of the ontology and an Application Pattern may provide certain constraints on the ontology, its interfaces and the construction process, as well as further directions on the usage of the ontology.

Reuse is another aspect that should be more explicitly incorporated in the Ontology Learning systems of the future. This could also be facilitated by patterns. The difficult question has so far been how to find parts to reuse, but this could be done by performing pattern-matching using the ontology patterns with the candidate ontologies (using techniques from for example the AIR toolset for reuse support in [95], or [3], [104] and [103] with techniques for Information Retrieval). For example by letting the user select a pattern that should be present in the ontology and then searching for those ontologies containing the pattern. In order to ease the users’ understanding, the ontologies could also be sorted according to what patterns they contain.

### 6.2 Summary

Based on the discussion in the previous section it can be noted that there are several fields in each different area that need further exploration. The in-depth development and research in the three different fields will not be the focus of future work. Instead the focus here is the intersection of the three areas; facilitating Ontology Learning and reuse with ontology patterns. Vladan Devedzic certainly has a point when he states, in [14], that Software Engineering and Ontology Engineering have much more in common than one might think, and that the areas would benefit from a closer cooperation.

As stated in the beginning of this report the general context of our research is small-scale application contexts and ontologies within enterprises. The process of building an ontology thereby must be so easy, or so automated, that it can be handled by anyone who is an expert of the domain in question but not an expert on ontologies. The aim is then to facilitate further automatisation of the Ontology Engineering process and increase the possibilities for reuse of ontologies. This is exactly what were considered as the most interesting research possibilities in the previous section, and also where ontology patterns on different levels of abstraction would be most needed. The patterns can then be used as templates in the construction process and as search and sorting criteria within ontology libraries. Another important issue is that knowledge should be reused and adopted from other areas, a closer cooperation with for example Software Engineering would be desirable.
References


[28] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.


