Zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften (Dr. rer. pol.)
von der Fakultät für Wirtschaftswissenschaften
des Karlsruher Instituts für Technologie (KIT)
vorgelegte Dissertation von
Dipl.-Inform. Zdenko Vrandečić.

# Ontology Evaluation

Denny Vrandečić

**Tag der mündlichen Prüfung:** 8. June 2010
**Referent:** Prof. Dr. Rudi Studer
**Erster Koreferent:** Prof. James A. Hendler, PhD.
**Zweiter Koreferent:** Prof. Dr. Christof Weinhardt
**Vorsitzende der Prüfungskommission:** Prof. Dr. Ute Werner

This document was created on June 10, 2010

*Koji su dozvolili da sanjam,*
*koji su omogućili da krenem,*
*koji su virovali da ću stići,*

*Vama.*

# Acknowledgements

Why does it take years to write a Ph.D.-Thesis?

Because of all the people who were at my side.

Tonči Vrandečić, my father, who, through the stories told about him, made me dream that anything can be achieved.

Perica Vrandečić, my mother, who showed me the truth of this dream.

Rudi Studer, my *Doktorvater*, who is simply the best supervisor one can hope for.

Martina Nöth, for carefully watching that I actually write. Elena Simperl, for giving the most valuable advises and for kicking ass. Anupriya Ankolekar, for reminding me that it is just a thesis. Uta Lösch, for significant help. Miriam Fernandez, for translating obscure Spanish texts. Markus Krötzsch, for an intractable amount of reasons. York Sure, for being a mentor who never restrained, but always helped. Sebastian Rudolph and Aldo Gangemi, for conceptualizing my conceptualization of conceptualizations. Stephan Grimm for his autoepistemic understanding. Yaron Koren, for his support for SMW while Markus and I spent more time on our theses.

Christoph Tempich, Peter Haase, Jim Hendler, Pascal Hitzler, Frank van Harmelen, Enrico Motta, Sofia Pinto, Marta Sabou, and Gus Schreiber, for their advise.

All the people in the Rudiverse, past and present, for making me feel more like being with friends than being at work. Especially I want to thank those, who became friends over the years, and will remain so after our time in Semantic Karlsruhe ends.

All my co-authors – a thousand thanks! All the people I worked with in the research projects and co-organizers of events that helped sharpening the ideas presented here.

The European Union for the projects SEKT and ACTIVE, and for creating a new Europe. Vulcan Inc. for Project Halo. They funded my time in Karlsruhe.

All those giants which shoulders I am balancing on, for carrying my weight, which surely is not the easiest of all tasks. All the anonymous reviewers, who made sharp comments to our papers, that lead to improvements of the presented work

My sister and my other good friends, who inspired me, either with their work, their ideas, their discussions, their presence, or most importantly with their friendship and love, giving me encouragement and reality checks whenever I needed them most.

Because of them, it took years to write this thesis.

Without them, it would have taken decades – if not forever.

# Abstract

Ontologies are a pillar of the emerging Semantic Web. They capture background knowledge by providing relevant terms and the formal relations between them, so that they can be used in a machine-processable way, and thus enable automatic aggregation and the proactive use and serendipitous reuse of distributed data sources. Ontologies on the Semantic Web will come from a vast variety of different sources, spanning institutions and persons aiming for different goals and quality criteria.

**Ontology evaluation** is the task of measuring the quality of an ontology. It enables us to answer the following main question:

*How to assess the quality of an ontology for the Web?*

Ontology evaluation is essential for a wide adoption of ontologies, both in the Semantic Web and in other semantically enabled technologies. We regard the three following scenarios as relevant for ontology evaluation:

- Mistakes and omissions in ontologies can lead to the inability of applications to achieve the full potential of exchanged data. Good ontologies lead directly to a higher degree of reuse of data and a better cooperation over the boundaries of applications and domains.
- People constructing an ontology need a way to evaluate their results and possibly to guide the construction process and any refinement steps. This will make the ontology engineers feel more confident about their results, and thus encourage them to share their results with the community and reuse the work of others for their own purposes.
- Local changes in collaborative ontology engineering may effect the work of others. Ontology evaluation technologies allow to automatically check if constraints and requirements are fulfilled, in order to automatically reveal plausibility problems, and thus to decrease maintenance costs of such ontologies dramatically.

In this thesis a theoretical framework and several methods breathing life into the framework are presented. The application to the above scenarios is explored, and the theoretical foundations are thoroughly grounded in the practical usage of the emerging Semantic Web. We implemented and evaluated a number of the methods. The results of these evaluations are presented, indicating the usefulness of the overall framework.

# Short Table of Contents


Acknowledgements 5
Abstract 7

**I Foundations 11**
1 Introduction 13
2 Terminology and Preliminaries 23
3 Framework 37

**II Aspects 63**
4 Vocabulary 65
5 Syntax 83
6 Structure 99
7 Semantics 127
8 Representation 143
9 Context 151

**III Application 165**
10 Collaborative ontology evaluation in Semantic MediaWiki 167
11 Related work 185
12 Conclusions 197

**IV Appendix 201**
List of Methods 203
List of Tables 205
List of Figures 207
Bibliography 209
Full Table of Contents 230

# Part I

# Foundations

1 Introduction 13

2 Terminology and Preliminaries 23

3 Framework 37

## Chapter 1

# Introduction

> What I mean (and everybody else means) by the word 'quality' cannot be broken down into subjects and predicates. This is not because Quality is so mysterious but because Quality is so simple, immediate and direct.
>
> *(Robert M. Pirsig, b. 1928, Zen and the Art of Motorcycle Maintenance (Pirsig, 1984))*

The **Semantic Web** (Berners-Lee *et al.*, 2001), also known as the **Web of Data**, is an extension of the hypertext Web. It enables the exchange and integration of data over the Web, in order to achieve the cooperation of humans and machines on a novel, world-wide scale.

**Ontologies** are used in order to specify the knowledge that is exchanged and shared between the different systems, and within the systems by the various components. Ontologies define the formal semantics of the terms used for describing data, and the relations between these terms. They provide an *"explicit specification of a conceptualization"* (Gruber, 1995). Ontologies ensure that the meaning of the data that is exchanged between and within systems is consistent and shared – both by computers (expressed by formal models) and humans (as given by their conceptualization). Ontologies enable all participants *'to speak a common language'*.

Ontologies, like all engineering artifacts, need a thorough **evaluation**. But the evaluation of ontologies poses a number of unique challenges: due to the declarative

nature of ontologies developers cannot just compile and run them like most other software artifacts. They are data that has to be shared between different components and used for potentially different tasks. Within the context of the Semantic Web, ontologies may often be used in ways not expected by the original creators of the ontology. Ontologies rather enable a serendipitous reuse and integration of heterogeneous data sources. Such goals are difficult to test in advance.

This thesis discusses the evaluation of Web ontologies, i.e. ontologies specified in one of the standard Web ontology languages (RDF(S) (Klyne and Carroll, 2004) and the different flavors of OWL (Smith *et al.*, 2004; Grau *et al.*, 2008)) and published on the Web, so that they can be used and extended in ways not expected by the creators of the ontology, outside of a central control mechanism. Some of the results of this thesis will also apply to other ontology languages, and also for ontologies within a closed environment. In turn, many problems discussed in earlier work on ontology evaluation do not apply in the context of Web ontologies: since the properties of the ontology language with regards to monotonicity, expressivity, and other features are known, they need not to be evaluated for each ontology anymore. This thesis will focus on domain- and task-independent automatic evaluations. That does not mean that the ontology has to be domain-independent or generic, but rather the evaluation method itself is. We will discuss other types of evaluations in Chapter 11.

This chapter contains introductory material by providing the motivation for this thesis (Section 1.1), giving a short preview of the contributions (Section 1.2), and offering a readers' guide for the rest of the thesis (Section 1.3). It closes with an overview of related previous work by the author (Section 1.4).

## 1.1 Motivation

Ontologies play a central role in the emerging Semantic Web. They capture background knowledge by providing relevant concepts and the relations between them. Their role is to provide formal semantics to terms, so that they can be used in a machine processable way. Ontologies allow us to share and formalize conceptualizations, and thus to enable humans and machines to readily understand the meaning of data that is being exchanged. This enables the automatic aggregation and the proactive use and serendipitous reuse of distributed data sources, thus creating an environment where agents and applications can cooperate for the benefit of the user on a hitherto unexperienced level (Berners-Lee *et al.*, 2001).

This section provides three preliminary arguments for the importance of ontology evaluation. The arguments are about (i) the advantages of better ontologies (Section 1.1.1), (ii) increasing the availability and thus reusability of ontologies (Section 1.1.2), and (iii) the lower maintenance costs of collaboratively created knowledge bases (Section 1.1.3).

### 1.1.1 Advantages of better ontologies

Ontologies are engineering artifacts, and as such they need to be evaluated like all other engineering artifacts. The central role of ontologies in the Semantic Web makes ontology evaluation an important and worthwhile task: mistakes and omissions in ontologies can lead to applications not realizing the full potential of exchanging data. Good ontologies lead directly to a higher degree of reuse and a better cooperation over the boundaries of applications and domains.

To just name a few examples of disadvantages of low quality ontologies: readability of an ontology may suffer if the vocabulary or the syntax contains errors. Reasoners may be unable to infer answers in case of inconsistent semantics. Underspecified ontologies hinder automatic mapping approaches. On the other hand, high quality ontologies can be easier reused (since their semantics can be mapped with a higher confidence to a known and grounded ontology within the reading system), can be more readily featured in an existing application (since good user interface elements can be automatically constructed in order to work with the ontology), and will easier discover and also actively omit data errors (since the ontology constrains possible data interpretations).

### 1.1.2 Increasing ontology availability

Ontologies on the Semantic Web are coming from a vast variety of different sources, spanning institutions and persons aiming for different quality criteria. Ontology evaluation techniques help to consolidate these quality criteria and make them explicit. This encourages the publication of ontologies and thus increases the number of available ontologies.

This argument has two sides: on the one hand, more ontologies will be published because the ontology engineers are more confident with releasing their work. On the other hand, ontologies can be automatically assessed about their quality and thus can be reused easier, since the confidence of the reusers in the quality of these ontologies increases. Even though this does not raise the actual number of ontologies accessible by a user, it does raise the number of ontologies that will be reused. This fosters cooperation and reuse on the Semantic Web and increases directly its impact and lowers its costs.

For anectodal support we refer the reader to the SEKT case studies (Sure *et al.*, 2007): in one of the case studies, legal experts were trained to engineer an ontology of their knowledge domain. They felt consistently that they were doing it wrong, but since it all *"looks a lot like common sense"* making mistakes made them feel ignorant and thus uncomfortable. Ontology evaluation tools are supposed to discover some of these mistakes, and thus to raise the confidence of the engineers in their own work – and thus to publish them more readily.

### 1.1.3 Lower maintenance costs

Decentralized collaborative ontology creation requires a high independence of tasks. If local changes lead to wide-reaching effects, then the users should be able to understand these effects. Otherwise they will invariably effect the work of many others, which will need a complex system of measures and counter-measures. The prime example is the world's currently biggest knowledge base, Wikipedia.

Changes within the knowledge contained by Wikipedia require to be constantly tracked and checked. Currently, thousands of users constantly monitor and approve these changes. This is inefficient, since many of these changes could be tracked automatically using ontology evaluation technologies. They allow to automatically check if certain constraints and requirements are fulfilled. This allows maintainers to quickly and automatically reveal plausibility problems, and thus to decrease maintenance and development costs of such ontologies dramatically.

## 1.2 Contribution

The contribution of this thesis is threefold: we (i) introduce a framework for ontology evaluation, (ii) we organize existing work in ontology evaluation within this framework and fill missing spots, and finally (iii) we implement the theoretical results in practical systems to make the results of this thesis accessible to the user.

### 1.2.1 A framework for ontology evaluation

Terminology and content in ontology evaluation research has been fairly diverse. Chapter 3 contains a survey of literature, and consolidates the terminology used. It identifies and defines a concise set of eight ontology quality criteria and ontology aspects that can be evaluated (Section 3.6):

- Accuracy
- Adaptability
- Clarity
- Completeness
- Computational efficiency
- Conciseness
- Consistency
- Organizational fitness

We identify and describe in detail the following aspects of ontologies (Part II):

- Vocabulary
- Syntax
- Structure
- Semantics
- Representation
- Context

### 1.2.2 Methods for ontology evaluation

We surveyed the existing literature on ontology evaluation for evaluation methods and organized them according to the introduced framework. For each method we describe it within the context of its respective aspect.

Furthermore, with the help of the framework we identified blind spots and missing methods. We introduce a number of novel evaluation methods to address these gaps. Among them are:

- Schema validation (Section 5.3)
- Pattern discovery using SPARQL (Section 6.2)
- Normalization (Section 7.1)
- Metric stability (Section 7.2)
- Representational misfit (Section 8.1)
- Unit testing (Section 9.1)

### 1.2.3 Implementation

In order to allow the theoretical results of this thesis to be used, we implemented a number of introduced methods within Semantic MediaWiki, an extension for the popular MediaWiki wiki engine. Semantic MediaWiki has become the most popular semantic wiki engine by far, used by several hundreds installations worldwide and nurturing an active open source developer community. Semantic MediaWiki was extended to allow for the collaborative evaluation of the knowledge created within the wiki, and thus lower the costs for maintaining such a knowledge base dramatically (Chapter 10).

## 1.3 Readers' guide

This thesis presents a conceptual framework and its implementations for defining and assessing the quality of an ontology for the Web. In this section we will offer an overview of the whole thesis so that readers may quickly navigate to pieces of particular interest to them. The whole thesis is written in such a way that it enables the understanding of the topic in one pass.

In the examples in this thesis, whenever a QName is used (see Section 5.2), we assume the standard namespace declarations given in Table 1.1. Often we omit namespaces for brevity and legibility.

| Prefix | Namespace |
|---|---|
| rdf | http://www.w3.org/1999/02/22-rdf-syntax-ns# |
| rdfs | http://www.w3.org/2000/01/rdf-schema# |
| xsd or xs | http://www.w3.org/2001/XMLSchema# |
| owl | http://www.w3.org/2002/07/owl# |
| skos | http://www.w3.org/2008/05/skos# |
| foaf | http://xmlns.com/foaf/0.1/ |
| dc | http://purl.org/dc/elements/1.1/ |
| swrc | http://swrc.ontoware.org/ontology# |
| dbpedia | http://dbpedia.org/resource/ |
| dolce | http://www.loa-cnr.it/ontologies/DOLCE-Lite.owl# |
| proton | http://proton.semanticweb.org/2005/04/protont# |
| sw | http://semanticweb.org/id/ |
| swp | http://semanticweb.org/id/Property-3A |
| swc | http://semanticweb.org/id/Category-3A |
| swivt | http://semantic-mediawiki.org/swivt/1.0# |
| aifb | http://www.aifb.kit.edu/id/ |

Table 1.1: Namespace declaration in this thesis

The **appendix** contains further navigational help: a bibliography of referenced works, lists of methods, tables and figures, an index of all relevant terms, and a full table of contents.

The content of this thesis is divided in three parts.

### 1.3.1 Foundations

The foundations describe necessary preliminaries for understanding the rest of the thesis and offers a framework for the whole field of ontology evaluation that is later filled with more details in the following chapters. This first chapter introduces the

topic of the thesis and gives a rough **overview** of the contribution and how the whole thesis is structured.

Chapter 2 gives the **terminology and preliminaries** needed to understand the rest of the work. Whereas most of the terms may be familiar to a user knowledgeable about Semantic Web technology, this chapter offers a concise definition of these terms within this thesis. Readers may skip this chapter first, and decide to use it for reference as needed. The index in the appendix can help with finding relevant sections.

Chapter 3 introduces the theoretical **framework** for ontology evaluation that is the main theoretical contribution of this thesis. It describes criteria for ontology evaluation, aspects that can be evaluated, and how the criteria, aspects, and measures can be used in order to achieve defined goals by the user.

### 1.3.2 Aspects

Part II of this thesis describe the six different aspects of ontology evaluation as defined by the evaluation framework. Each chapter in this part is dedicated to one of the aspects.

Chapter 4 deals with the **vocabulary** of an ontology. The vocabulary of an ontology is the set of all names in that ontology, be it URI references or literals, i.e. a value with a datatype or a language identifier. This aspect deals with the different choices with regards to the used URIs or literals.

Chapter 5 is about the ontology **syntax**. Web ontologies can be described in a number of different surface syntaxes such as RDF/XML, N-Triples, OWL Abstract Syntax, the Manchester Syntax, or many else. Often the syntactic description within a certain syntax can differ widely. This aspect is about the different serializations in the various syntaxes.

Chapter 6 evaluates the **structure** of an ontology. A Web ontology describes an RDF graph. The structure of an ontology is this graph. The structure can vary highly even when describing the same meaning.

Chapter 7 examines how the **semantics** of an ontology can be evaluated. A consistent ontology describes a non-empty, usually infinite set of possible models. The semantics of an ontology are the common characteristics of all these models.

Chapter 8 takes a look at the aspect of **representation**. This aspect captures the relation between the structure and the semantics. Representational aspects regard how the explicit structure captures and defines the intended meaning, being the formal semantics of a description logics theory or some other specification of meaning.

Chapter 9 finally regards how the **context** of an ontology can be used for evaluation. This aspect is about the features of the ontology when compared with other artifacts in its environment, which may be, e.g. an application using the ontology, a data source about the domain, or formalized requirements towards the ontology in form of competency questions.

### 1.3.3 Application

The last part describes practical implications and implementations of the work given in the previous part, compares it to other work in the area, and offers conclusions on the results.

Chapter 10 describes **Semantic MediaWiki**, an extension to the MediaWiki wiki engine, that allows for the massive collaborative creation and maintenance of ontologies. It discusses how collaborative approaches towards ontology evaluation are implemented within Semantic MediaWiki.

Chapter 11 surveys **related approaches** and how they relate to the presented framework. Whereas most of the related approaches have been already included in the description of their respective aspect, some of them do not fit into the overall framework presented in this thesis. In this chapter we analyze the reasons for that.

Chapter 12 finally summarizes the **results** and compares them to the motivation presented in this chapter. We collect and comment on research questions that remain open, and outline the expected future work and impact of the research topic.

## 1.4 Relation to previous publications

Most of the content in this thesis has been published previously. Here we present the relation to other publications of the author, in order to show which parts of the content have been already peer-reviewed. Most of the publications add further details to the given topic that has not been repeated in this thesis, either because of space constraints or because that part of the work has been performed by one of the co-authors. On the other hand, all the content in this thesis has been updated to adhere to a common framework and to OWL2, which has been released only recently.

The outline of this thesis was previously published in a much shorter form in (Vrandečić, 2009b). This includes the framework in Chapter 3, especially the criteria selection in Section 3.6 and the definition of aspects in Section 3.8, which provides a structure for the whole thesis. The meta-model presented in Section 3.2 has been previously published in (Vrandečić *et al.*, 2006c).

A number of publications present experiences and thoughts which have informed the whole thesis. In (Cregan *et al.*, 2005) and (Mochol *et al.*, 2008) we present experiences from ontology engineering, issues with then current tools, and problems with the OWL semantics (which in turn have informed the methods presented in Section 7.2 on stable metrics, Section 7.3 on completeness, and Section 9.2 on using ontologies with higher expressivity for consistency checking and query answering). Also experiences with distributed ontology engineering especially using the DILIGENT methodology has been gained in the EU project SEKT and published in (Vrandečić *et al.*, 2005), (Vrandečić *et al.*, 2006b), (Sure *et al.*, 2007), (Casanovas *et al.*, 2007), and (Casanovas *et al.*, 2005). These experiences had guided us in the the development of Semantic

MediaWiki (Chapter 10), unit testing for ontologies (Section 9.1) and the overall need to develop automatic evaluation methods that also care about the seemingly simpler aspects such as vocabulary (Chapter 4) or syntax (Chapter 5).

We also have organized two workshops on the topic of ontology evaluation (EON2006 at WWW2006 in Banff, Canada, and EON2007 at ISWC2007 in Seoul, South Korea) that have contributed heavily to an understanding of the topic of the thesis. The proceedings are available in (Vrandečić *et al.*, 2006a) and (Vrandečić *et al.*, 2007a), respectively.

An earlier version of Chapter 4 was published in (Vrandečić, 2009a).

The idea of RDF syntax normalization in order to enable XML validation in Section 5.3 has been previously published in (Vrandečić *et al.*, 2009).

The criticism of existing metrics presented in Section 6.1 was first raised in (Vrandečić and Sure, 2007). The idea of explicating and detecting patterns with macros described in Section 6.2 was introduced in (Vrandečić, 2005) and extended in (Vrandečić and Gangemi, 2006). In this thesis we have further expanded the idea considerably and added some evaluation based on a significant ontology corpus. Section 6.3 on AEON has been previously presented in (Völker *et al.*, 2005) and (Völker *et al.*, 2008). There the machine learning aspects of AEON and its evaluation is also described.

The notions of normalization as presented in Section 7.1 and of stable metrics presented in Section 7.2 have both been previously published in (Vrandečić and Sure, 2007). The description of normalization in (Vrandečić and Sure, 2007) contained some formal errors though. They have been corrected in (Vrandečić *et al.*, 2007b). Section 7.1 presents the corrected version.

Ontological metrics as described in Section 8.1 have been first published in (Vrandečić and Sure, 2007).

Unit testing as presented in Section 9.1 was introduced in (Vrandečić and Gangemi, 2006). The idea of extending ontologies with stronger axiomatizations as presented in Section 9.2 was exemplified in (Lösch *et al.*, 2009) and (Völker *et al.*, 2007). The idea of extending ontologies with logic program rules builds on the line of work of (Grosof *et al.*, 2003) and especially (Motik, 2006) and was implemented as tools in (Motik *et al.*, 2005) and evaluated in (Hitzler and Vrandečić, 2005) and (Krötzsch *et al.*, 2006a), especially for efficiency.

Semantic MediaWikia (SMW) has been first presented in (Krötzsch *et al.*, 2005) and then in (Völkel *et al.*, 2006). The further development of SMW has been documented in (Krötzsch *et al.*, 2006b) and (Krötzsch *et al.*, 2007c). Chapter 10 provides an updated introduction to the whole system. The ideas expanded and implemented in Section 10.4 have been first published in (Vrandečić, 2009c). Introducing further reasoning expressivity to SMW in order to evaluate the content was published in (Vrandečić and Krötzsch, 2006) and (Krötzsch *et al.*, 2007b).

## Chapter 2

# Terminology and Preliminaries

> All good counsel begins in the same way; a man should know what he is advising about, or his counsel will all come to nought.
>
> *(Socrates, 469 BC–399 BC, Phaedrus (Plato, 370 BC))*

This chapter defines the terminology used in this thesis. Unlike a glossary, the terms will be given in a logical, or pedagogical order, meant to be read in order of presentation. Some of the following terms have a wider meaning outside of this thesis, and sometimes that wider meaning is mentioned. But in order to enable a succinct discussion we will widely disregard these additional meanings. Within this chapter, words written in **bold** are the words defined in that paragraph. Words written in SMALL CAPITALS are references to the given definitions.

## 2.1 Ontologies

An **ontology** is a (possibly NAMED) set of AXIOMS. Axioms are stated in an ONTOLOGY LANGUAGE. If all axioms of an ontology are stated in the same ontology language, then the ontology as a whole is in that ontology language.

An **ontology language** defines which language constructs (i.e. which types of axioms) can be used in an ontology in that language. The ontology language also defines the formal semantics of that language. A big number of ontology languages have been suggested in the last twenty years, such as Ontolingua (Farquhar *et al.*, 1996), F-logic (Kifer *et al.*, 1995), or plain first order logic.

**Web ontologies** are ONTOLOGIES that are written in one of the standardized SEMANTIC WEB ONTOLOGY LANGUAGES. Within this thesis we regard only Web ontologies, i.e. other ontologies using ontology languages do not necessarily have the same properties and thus may not be evaluable with the methods presented here.

As of writing of this thesis, the **Semantic Web ontology languages** are RDF, RDFS (jointly called RDF(S)),and OWL. OWL is available in a number of profiles with specific properties (Grau *et al.*, 2008). All these languages are standardized by the World Wide Web Consortium (W3C), a public standards body overseeing standards relevant to the development of the Web.

According to the Semantic Web ontology languages, ontologies do not include only terminological knowledge – definitions of the terms used to describe data, and the formal relations between these terms – but may also include the knowledge bases themselves, i.e. terms describing individuals and ground facts asserting the state of affairs between these individuals. Even though such knowledge bases are often not regarded as being ontologies (see (Obrst *et al.*, 2007) for an example), for the remainder of this thesis we follow the OWL standard and regard ontologies as artifacts encompassing both the terminological as well as the assertional knowledge.

An **ontology document** is a particular serialization of an ONTOLOGY. As such, it is an information resource, usually a file, and thus an artifact that can be processed by a machine. WEB ONTOLOGIES may be serialized in one of the many W3C standards for ontology serialization, i.e. RDF/XML (Beckett, 2004), OWL Abstract Syntax (Patel-Schneider *et al.*, 2004), OWL XML presentation syntax (Hori *et al.*, 2003), N3 (Berners-Lee, 2006), or OWL Functional Syntax (Motik *et al.*, 2009b). There are several further serializations, which can be translated from and to the other set of serializations, for example the KAON2 ontology serialization, the Manchester Syntax (Horridge *et al.*, 2006), or a memory only JAVA internal presentation. In Chapter 5 we will discuss the aspect of serializations and syntax more deeply. An infinite number of different ontology documents can describe the same ontology.

WEB ONTOLOGIES are often represented with an **RDF graph** (Klyne and Carroll, 2004), and in turn the RDF graph is serialized in an RDF DOCUMENT. This is the case when using the RDF/XML or the N3 serializations. The RDF graph in turn has to be interpreted to arrive at the actual ontology. This additional step was introduced by the W3C standards in order to achieve a form of interoperability between the different layers in the so called *Semantic Web Layer Cake* (first introduced in (Berners-Lee *et al.*, 2001) and updated several times since then, see e.g. (Berners-Lee *et al.*, 2006b)). Every ontology can be represented by an RDF graph, but not every ontology is necessarily represented by an RDF graph. Whereas the interpretation of an RDF graph as an ontology always yields the same ontology, there is no canonical representation of an ontology as an RDF graph, i.e. there can be many RDF graphs representing the same ontology. This stems from the fact that the normative translation of ontologies to RDF graphs is not injective, as described in Section 4.1 of the OWL Language Semantics

and Abstract Syntax document (Patel-Schneider *et al.*, 2004).

WEB ONTOLOGIES are also often serialized as XML files (Bray *et al.*, 2008) or else given by an XML infoset (Cowan and Tobin, 2004). An XML file is a particular serialization of an XML infoset, but XML infosets could also be serialized in other ways (e.g. as binary XML (ISO 24824, 2007)). XML infosets in turn may either express the ontology directly (as, for example, in the OWL/XML presentation syntax (Hori *et al.*, 2003)) or express the RDF GRAPH that in turn expresses the ontology. This shows that ontologies can be expressed in a big number of ways. They can be expressed via RDF or XML or both or neither.

**Ontology elements** are both AXIOMS and ONTOLOGY ENTITIES. In Section 2.2 we describe the available types of axioms, followed by the types of entities in Section 2.3.

## 2.2 Axioms

An **axiom** is the smallest unit of knowledge within an ontology. It can be either a TERMINOLOGICAL AXIOM, a FACT, or an ANNOTATION. Terminological axioms are either CLASS AXIOMS or PROPERTY AXIOMS. An axiom defines formal relations between ONTOLOGY ENTITIES or their NAMES.

### 2.2.1 Facts

A **fact** or **individual axiom** is either an INSTANTIATION, a RELATION, an ATTRIBUTE, or an INDIVIDUAL (IN)EQUALITY.

An **instantiation** or **class instantiation** has the form[1]

`ClassAssertion(`$C$ $a$`)`

with $C$ being a CLASS EXPRESSION and $a$ being an INDIVIDUAL NAME. This is the same as stating that $a$ has the type $C$. Semantically that means that the INDIVIDUAL that has the NAME $a$ is in the extension of the set described by $C$.

A **(positive) relation** or **object property instance** has the form

`PropertyAssertion(`$R$ $a$ $b$`)`

with $a$ and $b$ being INDIVIDUAL NAMES and $R$ being an OBJECT PROPERTY EXPRESSION. Informally it means that the property $R$ pointing from $a$ to $b$ holds – e.g. saying that *Germany* has the *capital Berlin*. In the example, *Germany* and *Berlin* are individual names, and *capital* is the name of the property that holds between them. The actual instantiation of this property is thus called the relation. Semantically it means that the tuple $(a, b)$ is in the extension of the set $R$.

OWL2 introduces **negative relations**, i.e. the direct possibility to state that a certain relation is *not* the case. This uses the following form:

[1] For more on the syntax and semantics of the axioms throughout this thesis, see Section 2.4.

`NegativePropertyAssertion(`$R$ $a$ $b$`)`
This means that the tuple $(a, b)$ is not in the extension of the set $R$. Semantically, this was already possible to be indirectly stated in OWL DL by using the following statement:
`SubClassOf(OneOf(`$a$`) AllValuesFrom(`$R$ `ComplementOf(OneOf(`$b$`))))`
It is easy to see that the new syntax is far easier to understand.

An **attribute** or **datatype property instance** uses almost the same form as a RELATION:
`PropertyAssertion(`$R$ $a$ $v$`)`
with $a$ being an INDIVIDUAL NAME, $R$ being a DATATYPE PROPERTY EXPRESSION and $v$ being a LITERAL. A **negative attribute** uses the following form respectively:
`NegativePropertyAssertion(`$R$ $a$ $v$`)`

**Individual equality** is an axiom stating that two (or more) names refer to the same individual, i.e. that the names are synonyms. An **individual inequality** on the other hand makes explicit that the names do not refer to the same individual. Ontology languages with the *unique name assumption* assume that two different names always (or by default) refer to two different individuals. In OWL, this is not the case: OWL does not hold to the unique name assumption, and thus OWL does not make any assumptions about equality or inequality of two individuals referred to by different names. The syntax for these axioms is as follows (for $i \geq 2$):
`SameIndividual(`$a_1$ $a_2$ ... $a_i$`)`
`DifferentIndividuals(`$a_1$ $a_2$ ... $a_i$`)`

### 2.2.2 Class axioms

A **terminological axiom** is either a CLASS AXIOM or a PROPERTY AXIOM.

A **class axiom** can either be a SUBSUMPTION, CLASS EQUIVALENCE, DISJOINT, or DISJOINT UNION.

A **subsumption** has the following form:
`SubClassOf(`$C$ $D$`)`
with $C$ (the subclass) and $D$ (the superclass) being CLASS EXPRESSIONS. This axiom states that every individual in the extension of $C$ also has to be in the extension of $D$. This means that individuals in $C$ are described as being individuals in $D$. For example,
`SubClassOf(`$Square$ $Polygon$`)`
describes squares as polygons. Subsumptions may be SIMPLE SUBSUMPTIONS, COMPLEX SUBSUMPTIONS, or DESCRIPTIONS.

In a **simple subsumption** both the subclass and the superclass are CLASS NAMES instead of more complex CLASS EXPRESSIONS. Simple subsumptions form the backbone of class hierarchies.

In a **complex subsumption** both the subclass and the superclass are COMPLEX CLASS EXPRESSIONS. A complex subsumption thus sets an intricate restriction on the possible models of the ontology. Such restrictions may be rather hard to understand by users of the ontology.

In a **description**, either the subclass is a CLASS NAME and the superclass a COMPLEX CLASS EXPRESSION, or the other way around. This describes the named class, i.e. the complex class expression is a **condition** of the named class. If the named class is the subclass, then the complex class expression offers a **necessary condition** of the named class, i.e. each individual in the named class must also fit to the complex class expression. If the named class is the superclass, then the complex class expression offers a **sufficient condition** of the named class, i.e. each individual fitting to the complex class expression will also be an individual of the named class. Descriptions are among the most interesting axioms in an ontology, and they are the namesake of *description logics*.

A **class equivalence** is stated as follows (for $i \geq 2$):

`EquivalentClasses(`$C_1$ $C_2$ `...` $C_i$`)`

with $C_n$, $1 \leq n \leq i$, being CLASS EXPRESSIONS. Class equivalences are SIMPLE CLASS EQUIVALENCES, COMPLEX CLASS EQUIVALENCES, or DEFINITIONS.

In a **simple class equivalence**, both class expressions of the class equivalence axiom are CLASS NAMES. This is similar to a synonym, since it states that two names mean the same class, i.e. that they have the same extension.

In a **complex class equivalence** both classes are COMPLEX CLASS EXPRESSIONS, and thus the axiom defines an intricate condition on the possible models. Just like COMPLEX SUBSUMPTIONS such axioms and their implications may be hard to understand.

If any of the two classes in a class equivalence is a CLASS NAME and the other a COMPLEX CLASS EXPRESSION, then the axiom is a DEFINITION of the class name. A definition is the strongest statement about a class name, offering both a sufficient and necessary condition by means of the complex class description. Thus, a definition offers the complete meaning of a name by building on the meaning of the names used in the defining class expression. As an example, a mother can be completely described by the following axiom:

```
EquivalentClasses(Mother IntersectionOf(Woman
                                        SomeValuesFrom(child Thing)))
```

defining a `Mother` as a `Woman` with a `child`.

A **disjoint** is an axiom of the form (for $i \geq 2$):

`DisjointClasses(`$C_1$ $C_2$ `...` $C_i$`)`

with $C_n$, $1 \leq n \leq i$, being CLASS EXPRESSIONS. The axiom states that two classes have no common individuals. This type of axiom is syntactic sugar for the following axiom:

`SubClassOf(`$C_1$ `ComplementOf(`$C_2$ `...` $C_i$`))`
for all $C_n$, $2 \leq n \leq i$.

A **disjoint union** has the form (for $i \geq 2$):
`DisjointUnion(`$C$ $D_1$ $D_2$ `...` $D_i$`)`
stating that the class $C$ is a union of all classes $D_n$, $1 \leq n \leq i$, and at the same time the classes $D_n$, $1 \leq n \leq i$ are all mutually disjoint. A disjoint union is also called a **complete partition** or a **covering axiom**.

### 2.2.3 Property axioms

A **property axiom** describes formal semantics of properties. Unlike classes, properties can hardly be described or even defined with a single axiom. In other words, whereas an OWL ontology allows to classify individuals based on their descriptions, the same is not true for property instances. Property axioms can be used to define the formal semantics of properties, but this is hardly ever expressive enough to define a property. This is because the only PROPERTY EXPRESSIONS are INVERSE PROPERTY and PROPERTY CHAINS.

The available property axioms either define (i) the relation between two properties, (ii) their domain or range, (iii) their type, (iv) or keys for individuals. The formal semantics of all these axioms are given in Table 2.1 on page 34.

Relations between properties are **subproperties** (e.g. *mother* as a subproperty of *parent*), **equivalent properties** (e.g. *mother* as an equivalent property to *mom*), **disjoint properties** (e.g. *mother* and *father* have to be disjoint sets), and **inverse properties** (e.g. *child* is inverse to *parent*).

A **domain definition** defines the class an individual belongs to, if that property points *from* it. A **range definition** defines the class an individual belongs to, if that property points *to* it. E.g. the property *wife* would connect a groom with his bride, and thus have the domain *Man* and the range *Woman* (based on a conservative conceptualization not accounting for same-sex marriages). Note that domains and ranges are not constraints, i.e. the system will usually not detect inconsistencies if the related individuals do not belong to the specified class, a behaviour often anticipated by programmers since it resembles the usage of signatures in procedure definitions. In order to do so, the ontology requires either sufficient DISJOINTS between the classes (see Section 9.2.1) or we need to add the ability to formulate domains and ranges as constraints (see Section 9.2.4).

A number of axioms can be used to declare specific formal types of properties. These are **functional**, **inverse functional**, **reflexive**, **irreflexive**, **symmetric**, **asymmetric**, and **transitive** property declarations. The formal semantics of all these properties are given in Table 2.1.

Finally, property axiom can define **keys** over one or more properties. Keys, just as INVERSE FUNCTIONAL PROPERTIES are important to infer the identity of individuals,

and thus play a crucial role in merging data from heterogeneous sources.

### 2.2.4 Annotations

An **annotation** connects an ELEMENT by an ANNOTATION PROPERTY with an ANNOTATION VALUE. ELEMENTS can be either ENTITIES, ONTOLOGIES, or AXIOMS. An annotation has no impact on the DL semantics, but adds further information about the elements.

The most widely deployed annotation is `rdf:label`. It connects an element with a human-readable label. We will investigate this in Section 4.2.3 in detail.

Annotations can express further metadata about the data itself, e.g. who stated a specific axiom, when was a class introduced, which properties are deprecated, etc. Many of these annotations can be used for evaluations, and throughout this thesis we will frequently see examples of such usage. Before the introduction of the punning mechanism in OWL2 (see Section 4.1.5) annotations were also required to make statements about classes or properties. For example, the AEON approach for analyzing class hierarchies was only possible by expressing the meta-properties with annotations (see Section 6.3). Since punning was introduced it is allowed to make statements about classes directly, which is much more powerful. We have used this new feature in Section 6.3.

**Ontology annotations** add metadata about the whole ontology, e.g. stating the author of the ontology, the version of the ontology, pointing to previous versions, stating compatibility with previous versions, etc. The OWL standard already defines a number of these ontology annotations, but allows for more to be added.

## 2.3 Entities

An **ontology enitity** may be an INDIVIDUAL, a CLASS, a PROPERTY, or an ONTOLOGY. Since OWL2, the NAMES referencing these entities do not have to be disjoint anymore, i.e. one and the same name may point to both an individual and a class. Consider the following example where `Father` is the name for a property (connecting a person to its father), a class (of all fathers), and an individual (as an instance of the class role, which in return can be used to query for all roles a person has or can have).

```
ClassAssertion(Familyrole Father)
PropertyDomain(Father Person)
PropertyRange(Father Man)
SubPropertyOf(Father Parent)
InverseProperties(Parent Child)
EquivalentClass(Father SomeValuesFrom(Child Person))
EquivalentClass(Father HasValue(Role Father))
```

### 2.3.1 Individuals

**Individuals** can be given by their NAME or as an ANONYMOUS INDIVIDUAL. An individual can be any entity with an identity (otherwise it would not be possible to identify that entity with an identifier).

An **anonymous individual** does not have a **URI** but provides only a LOCAL NAME instead. This means that the individual can not be identified directly from outside of the given ontology, but only through indirect means like INVERSE FUNCTIONAL PROPERTIES, KEYS, or NOMINALS. We will discuss anonymous individuals in Section 4.3 in more detail.

### 2.3.2 Classes

A **class** is a set of individuals. A class is given by a **class expression**. A CLASS EXPRESSION may either be a CLASS NAME or a COMPLEX CLASS DESCRIPTION. A **class name** is simply the NAME, i.e. a URI, of a **class**. Class names do not carry any further formal information about the class.

A **complex class expression** defines a CLASS with the help of other ENTITIES of the ONTOLOGY. In order to create these expressions, a number of constructs can be used. The formal semantics and exact syntax of all these constructs are given in Table 2.2. The constructs are set operations or restrictions.

The available set operations are **intersections**, **unions**, **complements**, and **nominals**. A NOMINAL defines the extension of a class by listing all instances explicitly.

The available restrictions are the **existential restriction** on a property (i.e. a class of all instances where the property exists), the **universal restriction** (on a property $P$ and a class $C$, constructing a class where the instances have all their $P$ property values be instances of $C$), **unqualifed number restriction**, the **qualified number restriction**, and the **self-restriction** (on a property $P$, stating that an instance has to be connected to itself via $P$).

As we can see, classes can be expressed with a rich variety of constructs, whereas the same does not hold for individuals and properties.

### 2.3.3 Properties

**Properties** are given by a PROPERTY EXPRESSION. Most often, a **property expression** is just a **property name**. The only **complex property expressions** are INVERSE PROPERTIES and PROPERTY CHAINS.

An **inverse property** is the PROPERTY EXPRESSION that is used when the subject and the object exchange their place in a property instantiation. For an example, `child` is the inverse property of `parent`. Instead of giving the inverse property the PROPERTY NAME `parent`, we could have used the PROPERTY EXPRESSION `InverseOf(child)` instead.

A **property chain** is the PROPERTY EXPRESSION that connects several PROPERTY EXPRESSIONS in a chain, e.g. the property `uncle` can be described as a superproperty of the chaining of the properties `parent` and `brother` by using the following axiom:

```
SubPropertyOf(PropertyChain(parent brother) uncle)
```

Note that this is not a definition of `uncle` (since `uncle` may also be the chaining of the properties `parent`, `sister`, and `husband`). Since there are no boolean operators on properties (i.e. property unions, intersections, and complements) we cannot actually define `uncle`.

PROPERTIES can be either OBJECT PROPERTIES or DATA PROPERTIES. **Object properties** connect two individuals with each other. **Data properties** connect an individual with a DATA VALUE.

A **data value** is not represented by a URI but rather by a **literal**, the syntactic representation of a concrete value. The mapping between the LITERAL and the DATA VALUE is given by a DATATYPE MAP. For example, the typed literal `"4"^^xsd:int` is mapped to the number 4. More on literals will be discussed in Section 4.2.

### 2.3.4 Ontologies

An ONTOLOGY can be either a NAMED or an UNNAMED ONTOLOGY. ONTOLOGIES can also be regarded as ONTOLOGY ENTITIES and can have AXIOMS to describe them, especially with ANNOTATION AXIOMS (e.g. to state the authoring institution or version information).

A **named ontology** is an ontology that explicitly states its name inside the ontology. Within this ontology, and especially in external ontologies this ontology can now be referred to by name.

An **unnamed ontology** is an ONTOLOGY that has no NAME given explicitly within the ontology. If a LOCATION is available, the location may be used instead of the name in this case. An ontology's name is also always the LOCAL NAME which is represented by the empty string, which should be interpreted as *this ontology*. For example, in RDF/XML serialization the following statement would say that *this ontology* was created on January 1 2010. We can see that instead of the name in the first line there is just an empty string, enclosed by double quotes.

```
<owl:Ontology rdf:about="">
  <dc:created>2010-01-01</dc:created>
</owl:Ontology>
```

## 2.4 Semantics

Throughout this thesis we are using the OWL2 Functional Syntax (Motik *et al.*, 2009b) for serializing axioms and thus ontologies. We believe that this is the most understand-

able OWL2 syntax curently available. OWL2 Functional Syntax is easy to read and nevertheless concise, and unlike DL syntax it actually reflects not only the semantics of the axioms but often also their intention. To give an example: a domain declaration in Functional Syntax is written as

```
PropertyDomain( mother Female )
```

whereas in DL syntax the same statement would be

$\exists$`mother`.$\top \sqsubseteq$ `Female`

Although the DL syntax is more concise, the intention of a domain declaration is easier to see from the Functional Syntax. RDF based syntaxes such as N3 on the other hand become very unwieldy and need to deal with many artifacts introduced to the fact that complex axioms need to be broken down in several triples (see the example below).

Table 2.1 describes all OWL axiom types, their direct set semantics, and their translation to RDF. The table is abbreviated: for all axiom types marked with *, it contains only the version with two (resp. three in the case of the `DisjointUnion` axiom type) parameters, even though the parameter list can be arbitrarily long. This often complicates the RDF graph enormously.

To give one example: the `DisjointProperties` axiom type is given in Table 2.1 with two possible parameters, $R$ and $S$. This can be expressed in RDF with a single triple:

```
R owl:propertyDisjointWith S .
```

But the axiom type can use an arbitrary number of parameters, e.g.

`DisjointProperties(`$R\ S\ T$`)`

stating that all the given properties are mutually disjoint, i.e.

$$(R \sqcap S) \sqcup (R \sqcap T) \sqcup (S \sqcap T) \equiv \bot$$

Translating this axiom to RDF yields a much more complicated graph than the single triple above:

```
_:x1 rdf:type owl:AllDisjointProperties .
_:x1 owl:members _:x2 .
_:x2 rdf:first R .
_:x2 rdf:rest _:x3 .
```

```
_:x3 rdf:first S .
_:x3 rdf:rest _:x4 .
_:x4 rdf:first T .
_:x4 rdf:rest rdf:nil .
```

Table 2.2 is abbreviated in the same way. The `IntersectionOf`, `UnionOf`, `OneOf`, and `PropertyChain` expression types can all accomodate more than the given number of parameters. The Table is also abbreviated as it considers only object properties. Datatype properties are built in an analogous way, but further allow for **facets**. Facets allow to constrain the range of data values, i.e. one may restrict the age for adults to be bigger than or equal to eighteen years.

Table 2.1 and Table 2.2 are compiled from (Motik *et al.*, 2009b; Motik *et al.*, 2009a; Patel-Schneider and Motik, 2009). For further detail, a primer, and normative declarations of all entities, the reader should consult the standards.

| **Functional syntax** | **Set semantics** | **RDF-Graph (N3)** |
|---|---|---|
| `ClassAssertion`($C$ $a$) | $a \in C$ | $a$ `rdf:type` $C$. |
| `PropertyAssertion`($R$ $a$ $b$) | $(a,b) \in R$ | $a$ $R$ $b$. |
| `NegativePropertyAssertion` ($R$ $a$ $b$) | $(a,b) \notin R$ | `_:x rdf:type owl:NegativePropertyAssertion.` `_:x owl:sourceIndividual` $a$. `_:x owl:assertionProperty` $R$. `_:x owl:targetIndividual` $b$. |
| `SameIndividual`($a$ $b$)* | $a = b$ | $a$ `owl:sameAs` $b$. |
| `DifferentIndividuals`($a$ $b$)* | $a \neq b$ | $a$ `owl:differentFrom` $b$. |
| `SubClassOf`($C$ $D$) | $C \subseteq D$ | $C$ `rdfs:subClassOf` $D$. |
| `EquivalentClasses`($C$ $D$)* | $C \equiv D$ | $C$ `owl:equivalentClass` $D$. |
| `DisjointClasses`($C$ $D$)* | $(C \cap D) \equiv \bot$ | $C$ `owl:disjointwith` $D$. |
| `DisjointUnion`($C$ $D$ $E$)* | $C \equiv (D \cup E)$ $(D \cap E) \equiv \bot$ | $C$ `owl:disjointUnionOf _:x.` `_:x rdf:first` $D$. `_:x rdf:rest _:y.` `_:y rdf:first` $E$. `_:y rdf:rest rdf:nil.` |
| `SubPropertyOf`($R$ $S$) | $R \subseteq S$ | $R$ `rdfs:subPropertyOf` $S$. |
| `EquivalentProperties`($R$ $S$)* | $R \equiv S$ | $R$ `owl:equivalentProperty` $S$. |
| `DisjointProperties`($R$ $S$)* | $(R \cap S) = \bot$ | $R$ `owl:propertyDisjointWith` $S$. |
| `InverseProperties`($R$ $S$) | $(a,b) \in R \leftrightarrow (b,a) \in S$ | $R$ `owl:inverseOf` $S$. |
| `PropertyDomain`($R$ $C$) | $(a,b) \in R \rightarrow a \in C$ | $R$ `rdfs:domain` $C$. |
| `PropertyRange`($R$ $C$) | $(a,b) \in R \rightarrow b \in C$ | $R$ `rdfs:range` $C$. |
| `FunctionalProperty`($R$) | $(a,b) \in R \wedge (a,c) \in R$ $\rightarrow b = c$ | $R$ `rdf:type owl:FunctionalProperty.` |
| `InverseFunctionalProperty`($R$) | $(a,c) \in R \wedge (b,c) \in R$ $\rightarrow a = b$ | $R$ `rdf:type owl:InverseFunctionalProperty.` |
| `ReflexiveProperty`($R$) | $a \in \top \rightarrow (a,a) \in R$ | $R$ `rdf:type owl:ReflexiveProperty.` |
| `IrreflexiveProperty`($R$) | $a \in \top \rightarrow (a,a) \notin R$ | $R$ `rdf:type owl:IrreflexiveProperty.` |
| `SymmetricProperty`($R$) | $(a,b) \in R \leftrightarrow (b,a) \in R$ | $R$ `rdf:type owl:SymmetricProperty.` |
| `AsymmetricProperty`($R$) | $(a,b) \in R \rightarrow (b,a) \notin R$ | $R$ `rdf:type owl:AsymmetricProperty.` |
| `TransitiveProperty`($R$) | $(a,b) \in R \wedge (b,c) \in R$ $\rightarrow (a,c) \in R$ | $R$ `rdf:type owl:TransitiveProperty.` |
| `HasKey`($C$ $R$ $S$)* | $(a,c) \in R \wedge (b,c) \in R$ $\wedge (a,d) \in S \wedge (b,d) \in S$ $\rightarrow a = b$ | $C$ `owl:hasKey _:x.` `_:x rdf:first` $R$. `_:x rdf:rest _:y.` `_:y rdf:first` $S$. `_:y rdf:rest rdf:nil.` |

Table 2.1: Semantics of OWL axioms. Axiom types noted with * may hold more than the given parameters.

| Functional syntax | Set semantics | RDF-Graph (N3) |
|---|---|---|
| IntersectionOf($C$ $D$)* | $C \cap D$ | _:x owl:intersectionOf _:y.<br>_:y rdf:first $C$.<br>_:y rdf:rest _:z.<br>_:z rdf:first $D$.<br>_:y rdf:rest rdf:nil. |
| UnionOf($C$ $D$)* | $C \cup D$ | _:x owl:unionOf _:y.<br>_:y rdf:first $C$.<br>_:y rdf:rest _:z.<br>_:z rdf:first $D$.<br>_:y rdf:rest rdf:nil. |
| ComplementOf($C$) | $\neg C$ | _:x owl:complementOf $C$. |
| OneOf($a$)* | $\{a\}$ | _:x owl:oneOf _:y.<br>_:y rdf:first $a$.<br>_:y rdf:rest rdf:nil. |
| SomeValuesFrom($R$ $C$) | $\{x \mid \exists((x,y) \in R \wedge y \in C)\}$ | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty $R$.<br>_:x owl:someValuesFrom $C$. |
| AllValuesFrom($R$ $C$) | $\{x \mid \forall(x,y) \in R \rightarrow y \in C\}$ | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty $R$.<br>_:x owl:allValuesFrom $C$. |
| HasValue($R$ $a$) | $\{x \mid \exists(x,a) \in R\}$ | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty $R$.<br>_:x owl:hasValue $a$. |
| HasSelf($R$) | $\{x \mid \exists(x,x) \in R\}$ | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty $R$.<br>_:x owl:hasSelf *true*. |
| MinCardinality($n$ $R$) | $\{x \mid \#\{y \mid (x,y) \in R\} \geq n\}$ | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty $R$.<br>_:x owl:minCardinality $n$. |
| MaxCardinality($n$ $R$) | $\{x \mid \#\{y \mid (x,y) \in R\} \leq n\}$ | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty $R$.<br>_:x owl:maxCardinality $n$. |
| ExactCardinality($n$ $R$) | $\{x \mid \#\{y \mid (x,y) \in R\} = n\}$ | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty $R$.<br>_:x owl:cardinality $n$. |
| MinCardinality($n$ $R$ $C$) | $\{x \mid \#\{y \mid (x,y) \in R \wedge y \in C\} \geq n\}$ | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty $R$.<br>_:x owl:onClass $C$.<br>_:x owl:minQualifiedCardinality $n$. |
| MaxCardinality($n$ $R$ $C$) | $\{x \mid \#\{y \mid (x,y) \in R \wedge y \in C\} \leq n\}$ | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty $R$.<br>_:x owl:onClass $C$.<br>_:x owl:maxQualifiedCardinality $n$. |
| ExactCardinality($n$ $R$ $C$) | $\{x \mid \#\{y \mid (x,y) \in R \wedge y \in C\} = n\}$ | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty $R$.<br>_:x owl:onClass $C$.<br>_:x owl:qualifiedCardinality $n$. |
| PropertyChain($R$ $S$)* | $\{(a,b) \mid \exists(a,x) \in R \wedge (x,b) \in S\}$ | _:x owl:propertyChain _:y.<br>_:y rdf:first $R$.<br>_:y rdf:rest _:z.<br>_:z rdf:first $S$.<br>_:z rdf:rest rdf:nil. |

Table 2.2: Semantics of OWL expressions using object properties (datatypes properties are analogous). Expression types with * may hold more parameters.

## Chapter 3

# Framework

> A definition is the starting point of a dispute, not the settlement.
>
> *(Neil Postman, 1931–2003, Language Education in a Knowledge Context (Postman, 1980))*

We introduce a framework for ontology evaluation. The rest of this thesis will be built on the framework in this chapter. First, we give an informal overview of the whole framework in Section 3.1, introducing the relevant terms and their connections. Section 3.2 describes an ontology of ontology evaluation and related concepts, formally specifying the framework presented earlier. Based on that specification, we define different types of ontologies in Section 3.3. We then outline the limits of this work in Section 3.4. The concepts of the ontology are finally discussed in more detail in the following sections: conceptualizations (Section 3.5), quality criteria (Section 3.6), evaluation methods (Section 3.7), and ontology aspects (Section 3.8).

## 3.1 Overview

The following framework is inspired by the semiotic meta-ontology $O^2$ and the ontology of ontology evaluation and selection *oQual* (Gangemi *et al.*, 2006b). We disagree on a few premises which will be named explicitly in Section 11.1. Nevertheless, we try to remain as close to $O^2$ and *oQual* as reasonable.

An **ontology** (i) specifies a conceptualization, (ii) consists of a set of axioms, (iii) is expressed by an ontology document, and (iv) constraints the construction of models satisfying the ontology. In Section 3.5 we discuss the conceptualizations and their

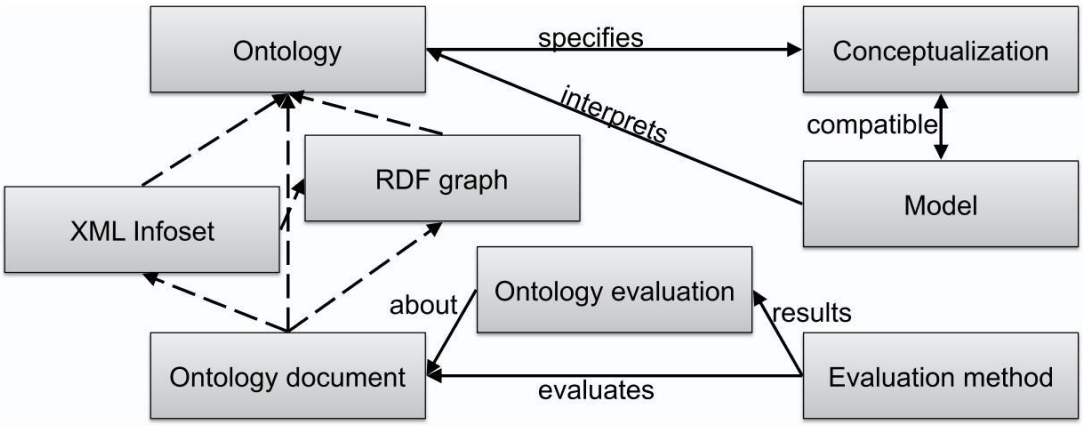


Figure 3.1: Framework for ontology evaluation. The slashed arrow represents the *expresses* relation.

relation to ontologies in detail. The structural definition of an ontology as a set of axioms was given in Section 2.2. The serialization or expression of an ontology as an ontology document was described in Section 2.1. Constraining the models is done by the semantics of an ontology as given in Section 2.4.

Ontologies are not artifacts in a narrow sense, but are expressed by ontology documents which in turn are artifacts. Whenever one speaks about ontologies as artifacts they mean ontology documents. Evaluation methods are descriptions of procedures that assess a specific quality of an ontology. Since methods cannot asses an ontology directly (since they are not artifacts), methods directly always evaluate ontology documents. Only indirectly it is possible for an evaluation method to assess an ontology (i.e. by assessing the ontology document that expresses the ontology). Figure 3.1 shows that in a bit more detail by describing the different levels an ontology document may express: either an XML Infoset, an RDF graph, or the ontology directly. These nuances were already discussed in Section 2.1.

Figure 3.1 summarizes the relations between models and conceptualizations, ontologies, ontology documents, and evaluation methods.

An **ontology evaluation** *may* be expressed by an ontology, which has the advantage that the result can be reused with the very same tools that we use with the ontologies anyway. It enables us to integrate the results of several different methods and thus to build complex evaluations out of a number of simple evaluations.

To give an example: the result of a method such as *calculating the ratio between the normalized and unnormalized depth* (described in Section 8.2) may be represented as a simple fact in an *ontology profile*:

```
PropertyAssertion(normalDepthRatio Ontology1 "2.25"^^xsd:decimal)
```

Another method may calculate the ratio between instantiated and uninstantiated classes in an ontology, and result in a second fact:

```
PropertyAssertion(instancedClassRatio Ontology1 "1.0"^^xsd:decimal)
```



Since the results are both expressed in OWL, we can easily combine the two facts and then create a new class of ontologies based on this metadata about ontologies:

```
EquivalentClasses(FluffyOntology}
   IntersectionOf(HasValue(normalDepthRatio "1.0"^^xsd:decimal)
                  HasValue(instancedClassRatio "1.0"^^xsd:decimal)))
```

(Hartmann *et al.*, 2005) introduced the *Ontology Metadata Vocabulary* (OMV) to describe such kind of ontology metadata such as results of measurements, its design policy, or how it is being used by others.

## 3.2 Meta-ontology

The framework presented in Section 3.1 and the terminology presented in Chapter 2 is specified as a meta-ontology (Vrandečić *et al.*, 2006c). We call it a meta-ontology since it is an ontology about ontologies. In Section 3.2.1 we show one example of a reified axiom in order to understand how the meta-ontology works. Section 3.2.2 discusses reifying entities. In Section 3.2.3 we will show the advantages of a meta-ontology within this thesis. In Section 3.3 we will then demonstrate how ontologies can be classified with the help of the meta-ontology.

### 3.2.1 Reifying ontologies

A major part of the meta-ontology provides the vocabulary to reify an ontology. This means that we create new entities to represent the axioms and entities in an ontology. Here we will demonstrate how the reification works based on a single axiom type, subsumption.

As shown in Section 2.2.2, the syntax for subsumptions is the following:

```
SubClassOf(o:Cat o:Pet)
```

The reification first creates a new individual to represent the axiom (`m:Axiom1` in the first line), then reifies the mentioned class names (second and third lines, creating `m:Cat` and `m:Pet`), connects the reified terms to the terms in the original axiom (see Section 3.2.2 for more details on reifying terms), and connects them to the individual representing the axiom. The last line adds a property directly relating the reified sub- and superclass, which helps with formulating some analyses (e.g. as given by the AEON approach in Section 6.3).

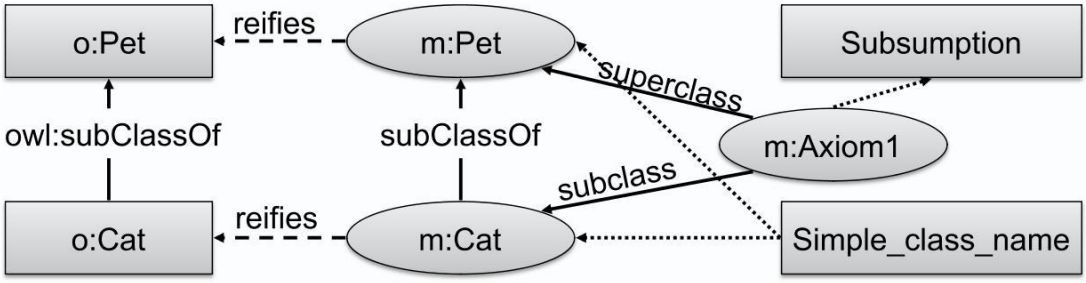


Figure 3.2: A subsumption axiom (on the left) and its reification. Dotted lines represent instantiation, slashed lines annotations, circles individuals, and squares classes.

```
ClassAssertion(meta:Subsumption m:Axiom1)
ClassAssertion(meta:Simple_class_name m:Cat)
ClassAssertion(meta:Simple_class_name m:Pet)
EntityAnnotation(o:Cat Annotation(meta:reifies m:Cat))
EntityAnnotation(o:Pet Annotation(meta:reifies m:Pet))
PropertyAssertion(meta:subclass m:Axiom1 m:Cat)
PropertyAssertion(meta:superclass m:Axiom1 m:Pet)
PropertyAssertion(meta:subClassOf m:Cat m:Pet)
```

The namespaces used in this example are `meta` for the meta-ontology itself, `m` for the reifying ontology, and `o` for the ontology that is being reified. Figure 3.2 illustrates the given example.

Every axiom is also explicitly connected to the ontology it is part of, and also the number of axioms is defined. This allows us to close the ontology and thus to classify it (see the classification example in Section 3.3.4). Furthermore, the ontology is also defined to be about all the terms used in the ontology.

```
ClassAssertion(meta:Ontology m:Ontology1)
PropertyAssertion(meta:axiom m:Ontology1 m:Axiom1)
ClassAssertion(ExactCardinality(1 meta:axiom) m:Ontology1)
PropertyAssertion(meta:about m:Ontology1 m:Cat)
PropertyAssertion(meta:about m:Ontology1 m:Pet)
```

The meta-ontology includes further axioms about the terms regarding subsumptions.

```
FunctionalProperty(meta:subClass)
PropertyDomain(meta:subClass meta:Subsumption)
```

```
PropertyRange(meta:subClass meta:Class)
FunctionalProperty(meta:superClass)
PropertyDomain(meta:subClass meta:Subsumption)
PropertyRange(meta:subClass meta:Class)
SubPropertyOf(PropertyChain(InverseOf(meta:subClass) meta:superClass)
              meta:subClassOf)
```

These axioms determine further formal relations between the terms of the meta-ontology. These decrease the risk of adding erroneous axioms without becoming inconsistent (i.e. the risk of adding errors that are not detected).

The property `meta:subClassOf` only regards explicitly stated subsumption. In order to expressed inferred subsumption a property `meta:inferredSubClassOf` exists.

### 3.2.2 Reifying URIs

The reified ontology does not make statements about the classes and individuals of the domain, but about the names used in the ontology. So, for example, even if the ontology claims that two terms are synonymous

`SameIndividual(o:Denny o:Zdenko)`

the reified individuals representing these individuals are *not* the same, since they are different names.

`DifferentIndividuals(m:Denny m:Zdenko)`

This means, whereas `o:Denny` represents the author of this thesis, `m:Denny` represents the name of the author of this thesis (in this case the URI `o:Denny`).

Even though it would be easy to create new URIs for every reified axiom and entity by simply creating new random URIs, it makes sense to generate the reified URIs for the entities based on their actual URIs. We use an invertible function by simply concatenating an URI-encoded (i.e. escaped) URI to a fixed namespace. This allows us to set up a Web service that returns information about the given reified URI at the fixed namespace.

### 3.2.3 Advantages of a meta-ontology

Throughout this thesis, we will use reified ontologies in order to express the methods succinctly and unambiguously. The following gives an overview of some of the uses the reified ontology can be put to.

- Section 3.3 shows how the meta-ontology can be used to classify the ontology itself and thus to make information about the ontology explicit.
- SPARQL queries can be used to discover the application of ontology patterns or anti-patterns (see Section 6.2 for examples).

- There exists no standard OWL query language and SPARQL can not be used to query for the existence of OWL axioms (see Section 6.2 for details). Based on a reified meta-ontology it is possible to use SPARQL for queries against the axiom structure of the ontology (instead merely the RDF graph structure).
- Meta-properties and constraints on meta-properties can be directly expressed and reasoned over. The AEON approach uses the OntoClean methodology (Guarino and Welty, 2002) in order to first state the meta-properties (e.g. if a class is rigid or not) and then to check the constraints automatically (see Section 6.3 for details).
- Additional axioms can be added to check if the ontology satisfies specific constraints, for example, it is easy to check if a subsumption axiom is ever used to express the subsumption of a class by itself by adding that `meta:subClassOf` is irreflexive and checking the resulting ontology for consistency.

## 3.3 Types of ontologies

### 3.3.1 Terminological ontology

A **terminological ontology** is an ONTOLOGY that consists only of TERMINOLOGICAL AXIOMS and ANNOTATIONS. This is introduced in order to account for the often found understanding that an ONTOLOGY indeed is a set of only TERMINOLOGICAL AXIOMS. This disagrees with the W3C definition of the term ontology, where an ontology may also include FACTS or even be constituted only of facts. In DL terms, a terminological ontology consists only of a TBox, i.e. terminological knowledge.

In terms of the meta-ontology, we define a terminological ontology as follows:

```
EquivalentClasses(Terminological_ontology
   IntersectionOf(Ontology
                  AllValuesFrom(axiom
                                UnionOf(Terminological_axiom Annotation))))
```

This distinction is a fairly common one, even though all FACTS can be expressed as TERMINOLOGICAL AXIOMS (using NOMINALS) thus making this distinction irrelevant. We proof this by offering terminological axioms that could replace each type of fact:

*Proof.* A **fact** is either an INSTANTIATION, a RELATION, an ATTRIBUTE, or an INDIVIDUAL (IN)EQUALITY. An INSTANTIATION is expressed as
`ClassAssertion(`$C$ $a$`)`
Written as a terminological axiom the same meaning is conveyed like this:
`SubClassOf(OneOf(`$a$`)` $C$`)`

Relations and attributes can be either positive or negative.
Positive relations and positive attributes are expressed as
`PropertyAssertion(`$R$ $a$ $b$`)`
As a terminological axiom it can be written as
`SubClassOf(OneOf(`$a$`) HasValue(`$R$ $b$`))`
Negative relations or negative attributes are expressed as:
`NegativePropertyAssertion(`$R$ $a$ $b$`)`
This can be restated the following way:
`SubClassOf(OneOf(`$a$`) ComplementOf(HasValue(`$R$ $b$`)))`
An individual equality is expressed as:
`SameIndividual(`$a$ $b$`)`
As a terminological axiom this can be written as:
`EquivalentClasses(OneOf(`$a$`) OneOf(`$b$`))`
Finally, individual inequality is expressed as:
`DifferentIndividuals(`$a$ $b$`)`
This can be transformed to:
`DisjointClasses(OneOf(`$a$`) OneOf(`$b$`))` □

### 3.3.2 Knowledge base

A **knowledge base** is an ontology that consists only of facts and annotations. In DL terms, a knowledge base only has an ABox, i.e. contains only assertional knowledge.

```
EquivalentClasses(Knowledge_base
   IntersectionOf(Ontology
                  AllValuesFrom(axiom
                                UnionOf(Fact Annotation))))
```

A knowledge base **instantiates** an ontology if the facts within the knowledge base use property names and class names introduced in that ontology. Often, knowledge bases instantiate several ontologies. An **exclusive ontology instantiation** uses only property names and class names from a single ontology. Often such instantiating ontologies are named by the ontology, e.g. a **FOAF file** is an often exclusive ontology instantiation of the FOAF ontology (Brickley and Miller, 2005).

```
EquivalentClasses(FOAF_file
   IntersectionOf(Ontology
                  AllValuesFrom(axiom UnionOf(
                    Annotation
                    IntersectionOf(Relation
                                   HasValue(property FOAF_property))
```

```
                    IntersectionOf(Attribute
                                   HasValue(property FOAF_property))
                    IntersectionOf(Instantiation
                                   HasValue(class FOAF_class))))))
EuivalentClasses(FOAF_property
   IntersectionOf(Property HasValue(InverseOf(about) FOAF)))
EuivalentClasses(FOAF_class
   IntersectionOf(Class HasValue(InverseOf(about) FOAF)))
```

Note that ontologies are not partitioned into terminological ontologies and knowledge bases. Many ontologies on the Web will contain both terminological axioms and facts, and thus not belong to one or the other. Ontologies that include both facts and terminological axioms are called **populated ontologies**. We can further classify them into populated proper ontologies or populated taxonomies, based on the types of included terminological axioms.

### 3.3.3 Semantic spectrum

The **semantic spectrum** defines one dimension of ontologies, ranging from the most simple and least expressive to the most complex and most precise ones. It was first presented at an invited panel at the AAAI 1999 in Austin, Texas, then published in (Smith and Welty, 2001) and refined in (McGuiness, 2003; Uschold and Gruninger, 2004).

We aggregate the types of ontologies they report on as the following five, ordered by increasing complexity (see Figure 3.3 for a common visualization):

- catalogs / sets of IDs
- glossaries / sets of term definitions
- thesauri / sets of informal is-a relations
- formal taxonomies / sets of formal is-a relations
- proper ontologies / sets of general logical constraints

We name the most expressive type of ontologies *proper ontologies* rather than *formal ontologies* as in (Uschold and Gruninger, 2004) since we regard an ontology as being formal by definition (Gruber, 1995).

In order to appropriately evaluate an ontology, we have to first determine its type. The type of an ontology determines which evaluation methods can be useful for the ontology, and which make no sense. There are two ways to determine the type of the ontology: prescriptive and descriptive. **Prescriptive determination** is given by the

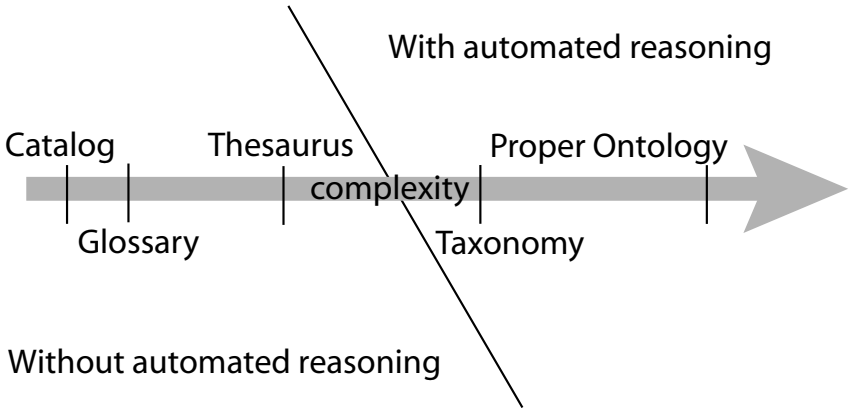


Figure 3.3: The semantic spectrum for ontologies.

ontology authors, i.e. they define the task that the ontology should fulfill and based on that the type of ontology that is required. **Descriptive determination** is given by examining an ontology and say what type of ontology is actually given.

Based on the meta-ontology described in Section 3.2 we can actually classify an ontology automatically since we can define the above types in the meta-ontology.

A **catalog** is an ontology that consists only of label annotations. This means that a catalog is just a set of URIs with human readable labels.

```
EquivalentClasses(Catalog
   IntersectionOf(Ontology
                  AllValuesFrom(axiom Label_annotation)))
```

In the simplified case, `Label_annotation` is defined as an annotation instantiating the `rdfs:label` annotation property (`Label` is the reification of `rdfs:label`).

```
EquivalentClasses(Label_annotation
   IntersectionOf(Annotation
                  HasValue(annotation_property Label)))
```

If we want to include not only instantiations of the `rdfs:label` property but also of its subproperties, e.g. `skos:prefLabel` or `skos:altLabel`, we need to redefine `Label_annotation` to also include its subproperties.

```
EquivalentClasses(Label_annotation
   IntersectionOf(Annotation
                  AllValuesFrom(annotation_property Label_property)))
EquivalentClasses(Label_property Has_Value(inferredSuperProperty Label))
```

Due to OWL's open world semantics this definition is much harder to reason with, since the reification of the ontology we want to classify needs to include sufficient axioms to make other models impossible. We will discuss possible solutions of that in Section 3.3.4.

A **glossary** is an ontology that only has annotations. This way, only human readable, informal definitions of the terms can be given.

```
EquivalentClasses(Glossary
   IntersectionOf(Ontology
                  AllValuesFrom(axiom Annotation)))
```

A **thesaurus** is an ontology that, besides annotations, also allows instantiations of classes and properties from the SKOS ontology (Miles and Bechhofer, 2009). SKOS (Simple Knowledge Organization System) is an ontology that allows to define thesauri with a number of predefined relations between terms, such as `skos:narrower` or `skos:broader`.

```
EquivalentClasses(Thesaurus
   IntersectionOf(Ontology
                  AllValuesFrom(axiom UnionOf(
                    Annotation
                    IntersectionOf(Relation
                                   HasValue(property SKOS_property))
                    IntersectionOf(Attribute
                                   HasValue(property SKOS_property))
                    IntersectionOf(Instantiation
                                   HasValue(class SKOS_class))))))
EuivalentClasses(SKOS_property
   IntersectionOf(Property HasValue(InverseOf(about) SKOS)))
EuivalentClasses(SKOS_class
   IntersectionOf(Class HasValue(InverseOf(about) SKOS)))
```

Catalogs and glossaries do not provide the means to allow any inferences (they are simply not expressive enough). Glossaries allow a very limited number of inferences, due to domain and range axioms and inverse and transitive properties.

A **taxonomy** or **class hierarchy** is an ontology that consists only of SIMPLE SUBSUMPTIONS, FACTS, and ANNOTATIONS.

```
EquivalentClasses(Taxonomy
   IntersectionOf(Ontology
                  AllValuesFrom(axiom
                                UnionOf(Simple_subsumption
```

```
                                    Fact
                                    Annotation))))
```

A **proper ontology** finally allows for all possible axioms, as defined in Section 2.2. In the semantic spectrum, each type of ontologies subsumes the simpler types, i.e.

$$\textit{Catalog} \subset \textit{Glossary} \subset \textit{Thesaurus} \subset \textit{Taxonomy} \subset \textit{Ontology}$$

This means that every glossary is also a taxonomy (though obviously a degenerated taxonomy, since the depth of the class hierarchy is 0), etc.

### 3.3.4 Classification example

This Section gives an illustrative example how an ontology may be classified. Afterwards we discuss problems of such classification and possible solutions.

Given ontology `o`:

```
ClassAssertion(o:Human o:Socrates)
SubClassOf(o:Human o:Mortal)
```

The reified ontology `m` is the following:

```
ClassAssertion(Class m:Human)
ClassAssertion(Class m:Mortal)
ClassAssertion(Individual m:Socrates)
ClassAssertion(Instantiation m:a1)
PropertyAssertion(class m:a1 m:Human)
PropertyAssertion(instance m:a1 m:Socrates)
PropertyAssertion(type m:Socrates m:Human)
ClassAssertion(Subsumption m:a2)
PropertyAssertion(subclass m:a2 m:Human)
PropertyAssertion(superclass m:a2 m:Mortal)
PropertyAssertion(subClassOf m:Human m:Mortal)
ClassAssertion(Ontology m:o1)
ClassAssertion(ExactCardinality(2 meta:axiom) m:o1)
PropertyAssertion(about m:o1 m:Human)
PropertyAssertion(about m:o1 m:Mortal)
PropertyAssertion(about m:o1 m:Socrates)
DifferentIndividuals(m:Human m:Mortal m:Socrates m:o1)
```

Using this with the meta-ontology, a reasoner can infer that `m:o1` is a `Taxonomy`: it is stated that there are exactly two axioms in the ontology (i.e. the ontology is closed),

so it is allowed to make inferences from the universal quantifier in the definition of `Taxonomy`.

Many reasoners cannot deal well with cardinality constraints. KAON2 (Motik, 2006) requires a long time to classify an ontology starting with more than four axioms, whereas Pellet (Sirin *et al.*, 2007) and Fact++ (Tsarkov and Horrocks, 2006) start to break with ontologies having more than a few dozen axioms. Since ontologies may easily contain much bigger numbers of axioms, it may be preferable to write dedicated programs to check if an ontology is a taxonomy or not. These programs may assume certain structural conditions which the ontology reification has to adhere to. In this example, the dedicated classifier may assume that the ontology is always complete when reified and thus partially ignore the open world assumption.

## 3.4 Limits

Ontology evaluations are conducted on several different levels:

1. Ontologies can be evaluated by themselves.

2. Ontologies can be evaluated with some context.

3. Ontologies can be evaluated within an application. This is called **application based ontology evaluation** (Brank *et al.*, 2005).

4. Ontologies can be evaluated in the context of an application and a task. This is called **task based ontology evaluation** (Porzel and Malaka, 2004).

In this thesis we will restrict ourselves to the first two possibilities. Note that each of the above levels gains from evaluating the previous levels, i.e. every ontology evaluated within an application should have been evaluated by itself and with some context before that. Many types of errors are much easier discovered on the first and second level than in the much more complex environment of an application or a task. The majority of this thesis deals with the first task (Chapters 4–8), whereas the second point is dealt with in Chapter 9.

Ontology-based applications most often have certain requirements regarding the applied ontologies. For example, they may require that the data within the ontology is complete (e.g. a semantic birthday reminder application may require that all persons need to have at least their name and birthday stated), or they may have certain structural or semantic constraints (e.g. the class hierarchy must not be deeper than five levels). Such conditions can often be stated in a way that allows to use the evaluation methods within this thesis in order to ensure the application's requirements are satisfied.

Asunción Gómez-Pérez separates ontology evaluation into two tasks: ontology verification and ontology validation (Gómez-Pérez, 2004).

**Ontology verification** is the task of evaluating if the ontology has been built correctly. Verification checks the encoding of the specification. Errors such as circular class hierarchies, redundant axioms, inconsistent naming schemes etc. are detected by ontology verification. Verification confirms that the ontology has been built according to certain specified ontology quality criteria.

**Ontology validation** is the task of evaluating if the correct ontology has been built. Validation refers to whether the meaning of the definitions matches with the conceptualization the ontology is meant to specify. The goal is to show that the world model is compliant with the formal models.

Since this thesis mainly concentrates on methods that can be highly automatized we will limit ourselves to ontology verification. Chapter 11 will discuss further methods that lean more towards ontology validation. Often such ontology validation methods assume a simpler understanding of ontologies i.e. they assume that an ontology is more a formal description of a domain than the formal specification of a shared conceptualization. Section 3.5 will discuss in detail the ramifications of this difference.

Like applications, ontologies are used in many different domains. Whereas it is easily possible, and often sensible, to create domain-specific ontology evaluation methods, in this thesis we will restrict to domain-independent evaluations. This means that even though the examples presented throughout the thesis will use some domain, the applied methods will always be usable on any other domain as well.

To summarize, this thesis will concentrate on the domain-, task-, and application-independent verification of Web ontologies. The framework presented in this chapter, and thus the methods described throughout the following chapters, have to be regarded in light of this limitation. Also, the methods presented in Part II are not a complete list of possible evaluation methods. They should rather be regarded as a list of exemplary methods in order to illustrate the usage of the framework presented here. Further methods can easily be created and combined within the framework in order to meet the needs of the evaluator.

## 3.5 Conceptualizations

This section describes and formalizes how agents can achieve a shared conceptualization. The formalization is reminiscent of epistemic logics, but differs from them in two main points: first, epistemic logics are about propositions, whereas here we speak mainly about conceptualizations. Second, epistemic logics assume that propositions itself are shareable, whereas here we assume conceptualizations to be private to an agent. In this section we will resolve the discrepancy between shared and private conceptualizations.

A **rational agent** has thoughts and perceptions. In order to express, order, and sort its thoughts and perceptions the agent creates, modifies, and uses conceptualizations. A conceptualization is the agent's mental model of the domain, its understanding of the domain. We define $C_X(d)$ to be the conceptualization that agent $X$ has of domain $d$, $C_X$ being a function $C_X : \mathcal{D} \rightarrow \mathcal{C}_X$ with $\mathcal{D}$ being the set of all domains and $\mathcal{C}_X$ being the set of all conceptualizations of agent $X$.

A **shared conceptualization** represents the commonalities of two (or more) conceptualizations of different agents. We define the operator $\cap$ for creating a shared conceptualization, but note that this should not be understood as the set-theoretic intersection operator – conceptualizations probably are not sets. It is rather an operation done internally by the agent on two or more of his internal conceptualizations to derive the commonalities.

Extracting a shared conceptualization can only be done with conceptualizations of the same agent, i.e. $C_X(d) \cap C_Y(d)$ is undefined. This is because conceptualizations are always private (i.e. conceptualizations have no reality external to their agents and thus two agents can not have the *same* conceptualization). So in order to intersect the conceptualizations of two different agents, an agent needs to conceptualize the other agent's conceptualization (as stated above, anything can be conceptualized, in particular another conceptualization). By interacting and communicating with the other agents, each agent builds conceptualizations of the other agents. These conceptualized agents again are conceptualized with their own private conceptualizations. $C_X(C_Y(d))$ is the conceptualization $X$ has of the conceptualization $Y$ has of a domain $d$ (note that $C_X(C_Y(d))$ tells us more about $C_X(Y)$ and thus about $X$ than about $C_Y(d)$ or about $Y$). For simplicity, we assume that each agent's conceptualization of its own conceptualization is perfect, i.e. $C_X(C_X(d)) = C_X(d)$ (this is similar to the KK axiom in epistemic logics (Fagin *et al.*, 2003)). Figure 3.4 illustrates two agents and their conceptualization of a domain (in the example, a tree) and their conceptualizations of each other and their respective conceptualizations. Note that the agents conceptualizations do not overlap.

An agent can combine its conceptualizations to arrive at a shared conceptualization, i.e. $C_X(d) \cap C_X(C_Y(d))$ results in what $X$ considers to be the common understanding of $d$ between itself and $Y$. Regarding a whole group of $n$ agents we define the (private conceptualization of the) shared conceptualization $SC_X$ as follows (let $C_X$ be one of $C_{Y_i}$):

$$SC_X(d) = \bigcap_{i=1}^{n} C_X(C_{Y_i}(d))$$

Furthermore, $X$ assumes that $\forall i : C_X(SC_{Y_i}(d)) = SC_X(d)$, i.e. $X$ assumes that everybody in the group has the same shared conceptualization. This is true for all

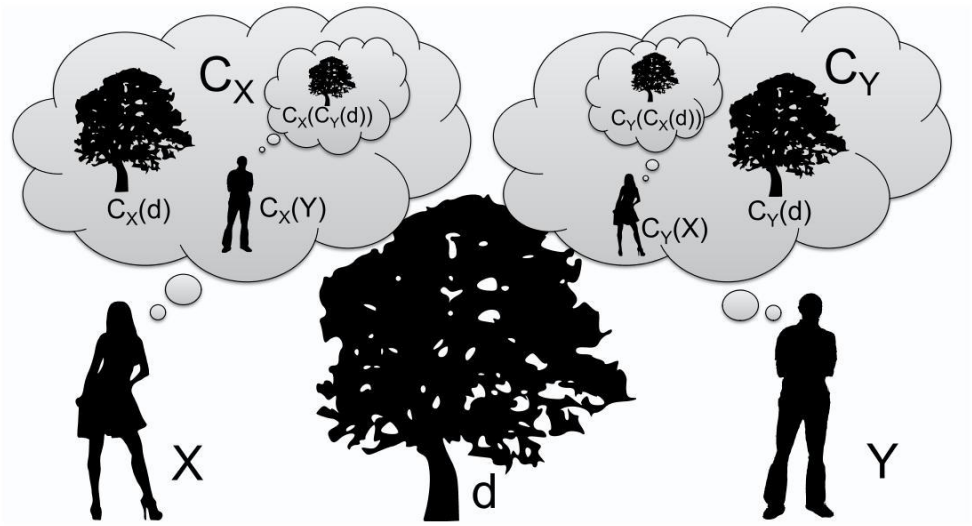


Figure 3.4: Two agents $X$ and $Y$ and their conceptualizations of domain $d$ (the tree), each other, and their respective conceptualizations.

members of the group, i.e. $\forall i, j : C_{X_i}(SC_{Y_j}(d)) = SC_{X_i}(d)$. In Figure 3.5 this is visualized by $C_X(X)$ and $C_X(Y)$ having the same shared conceptualization $SC_X(d)$.

An ontology $O$ is the specification (defined as the function $S$) of a conceptualization $C$, i.e. it is the result of the externalization of a conceptualization $O = S(C)$. For our discussion it is not important, how $S$ was performed (e.g. if it was created collaboratively, or with the help of (semi-)automatic agents, or in any other way).

The resulting ontology is a set of axioms that constrain the interpretations of the ontology. This has two aspects, depending on the interpreting agent: a **formal agent** (e.g. an application using a reasoning engine to apply the formal semantics) will have the possible logical models constrained, and based on these models it will be able to answer queries; a **rational agent** (e.g. a human understanding the ontology) is constrained in the possible mappings of terms of the ontology to elements of its own internal conceptualization (possibly changing or creating the conceptualization during the mapping). Figure 3.5 illustrates an agent $Z$ who internalizes ontology $O$ and thus builds its own conceptualization $C_Z$ of domain $d$ from its understanding of the ontology (i.e. $C_Z(d)$ is not build from the perceptions of $d$ by $Z$ but from $O$).

To give an example: if the property `married` is stated to be a functional property, a formal agent will only allow models to interpret the ontology where the set $R$ denoted by `married` fulfills the condition $(x, y) \in R \wedge (x, z) \in R \rightarrow y = z$. A rational agent in turn will map the term `married` to a monogamous concept of marriage, and will not use the term to refer to polygamous marriages.

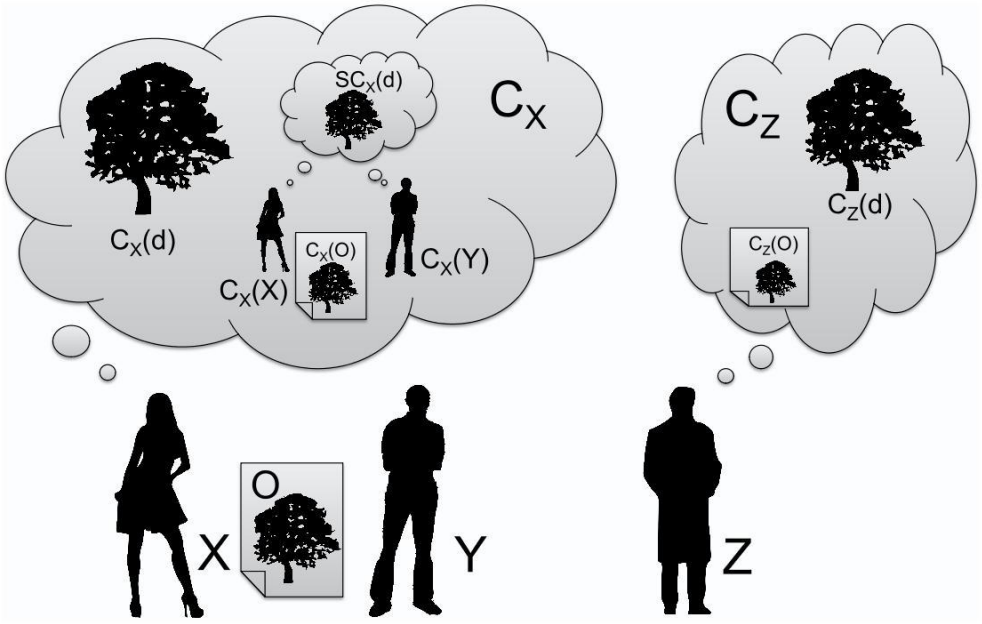


Figure 3.5: Three agents and an ontology. $Y$'s conceptualization is omitted for space reasons. $Z$ internalizes ontology $O$, thus connecting it to or creating its own conceptualization $C_Z$ of domain $d$, in this case, the tree.

In this case, the rational agent already had a concept that just needed to be mapped to a term from the ontology. In other cases the agent may need to first create the concept before being able to map to it (for example, let a *wide rectangle* be defined as a rectangle with the longer side being at least three times the size of the short side – in this case readers create a new concept in their mind, and then map the term *wide rectangle* to it).

So after the creation of the ontology $O$, each member $Y$ of the group can create a conceptualization of $O$, i.e. internalize it again. So a group member $Y_i$ gets the internal conceptualization $C_{Y_i}(O)$, and then compares it to its own understanding of the shared specification $SC_{Y_i}(d)$. Ideally, all members of the group will agree on the ontology, i.e. $\forall i : C_{Y_i}(O) = SC_{Y_i}(d)$

Note that creating an ontology is not the simple, straight-forward process that is presented in this section. Most of the conceptualizations will be in constant flux during the process. The communication in the group during the creation of the specification may change each member's own conceptualization of the domain, each member's conceptualization of each other member's conceptualization of the domain, each member's shared conceptualization of the domain, and in some cases even the domain itself.

## 3.6 Criteria

Ontology evaluation can regard a number of several different criteria. In this section we will list criteria from literature, aggregate them to form a coherent and succinct set, and discuss their applicability and relevance for Web ontologies. The goal of an evaluation is not to perform equally well for all these criteria – some of the criteria are even contradicting, such as *conciseness* and *completeness*. It is therefore the first task of the evaluator to choose the criteria relevant for the given evaluation and then the proper evaluation methods to assess how well the ontology meets these criteria.

We selected five important papers from literature, where each defined their own set of ontology quality criteria or principles for good ontologies (Gómez-Pérez, 2004; Gruber, 1995; Grüninger and Fox, 1995; Gangemi *et al.*, 2005; Obrst *et al.*, 2007). These quality criteria need to be regarded as desiderata, goals to guide the creation and evaluation of the ontology. None of them can be directly measured, and most of them cannot be perfectly achieved.

Asunción Gómez-Pérez lists the following criteria (Gómez-Pérez, 2004):

- **Consistency**: capturing both the logical consistency (i.e. no contradictions can be inferred) and the consistency between the formal and the informal descriptions (i.e. the comments and the formal descriptions match)
- **Completeness**: All the knowledge that is expected to be in the ontology is either explicitly stated or can be inferred from the ontology.
- **Conciseness**: if the ontology is free of any unnecessary, useless, or redundant axioms.
- **Expandability**: refers to the required effort to add new definitions without altering the already stated semantics.
- **Sensitiveness**: relates to how small changes in an axiom alter the semantics of the ontology.

Thomas Gruber defines the following criteria (Gruber, 1995):

- **Clarity**: An ontology should effectively communicate the intended meaning of defined terms. Definitions should be objective. When a definition can be stated in logical axioms, it should be. Where possible, a definition is preferred over a description. All entities should be documented with natural language
- **Coherence**: Inferred statements should be correct. At the least, the defining axioms should be logically consistent. Also, the natural language documentation should be coherent with the formal statements.

- **Extendibility**: An ontology should offer a conceptual foundation for a range of anticipated tasks, and the representation should be crafted so that one can extend and specialize the ontology monotonically. New terms can be introduced without the need to revise existing axioms.

- **Minimal encoding bias**: An encoding bias results when representation choices are made purely for the convenience of notation or implementation. Encoding bias should be minimized, because knowledge-sharing agents may be implemented with different libraries and representation styles.

- **Minimal ontological commitment**: The ontology should specify the weakest theory (i.e. allowing the most models) and defining only those terms that are essential to the communication of knowledge consistent with that theory.

Grüninger and Fox define a single criteria, **competency** (or, in extension, **completeness** if all the required competencies are fulfilled). In order to measure competency they introduce informal and formal **competency questions** (Grüninger and Fox, 1995).

Obrst *et al.* name the following criteria (Obrst *et al.*, 2007):

- **coverage** of a particular domain, and the richness, complexity, and granularity of that coverage

- **intelligibility** to human users and curators

- **validity and soundness**

- evaluation against the **specific use cases**, scenarios, requirements, applications, and data sources the ontology was developed to address

- **consistency**

- **completeness**

- the sort of **inferences** for which they can be used

- **adaptability** and *reusability* for wider purposes

- **mappability** to upper level or other ontologies

Gangemi *et al.* define the following criteria (Gangemi *et al.*, 2005):

- **Cognitive ergonomics**: this principle prospects an ontology that can be easily understood, manipulated, and exploited.

- **Transparency** (explicitness of organizing principles): this principle prospects an ontology that can be analyzed in detail, with a rich formalization of conceptual choices and motivations.
- **Computational integrity and efficiency**: this principle prospects an ontology that can be successfully/easily processed by a reasoner (inference engine, classifier, etc.).
- **Meta-level integrity**: this principle prospects an ontology that respects certain ordering criteria that are assumed as quality indicators.
- **Flexibility** (context-boundedness): this principle prospects an ontology that can be easily adapted to multiple views.
- **Compliance to expertise**: this principle prospects an ontology that is compliant to one or more users.
- **Compliance to procedures for extension, integration, adaptation, etc.**: this principle prospects an ontology that can be easily understood and manipulated for reuse and adaptation.
- **Generic accessibility** (computational as well as commercial): this principle prospects an ontology that can be easily accessed for effective application.
- **Organizational fitness**: this principle prospects an ontology that can be easily deployed within an organization, and that has a good coverage for that context.

We have taken analyzed the given criteria and summarized them into a concise set. Eight criteria result from this literature survey: **accuracy**, **adaptability**, **clarity**, **completeness**, **computational efficiency**, **conciseness**, **consistency**, and **organizational fitness**. All criteria given in the literature are subsumed by the this set. In the following, we define the criteria and how they map to the given criteria described above.

We have ignored evaluation criteria that deal with the underlying language used for describing the ontology instead of evaluating the ontology itself. Before OWL became widespread, a plethora of knowledge representation languages were actively used. For some ontologies, specific ontology languages were developed in order to specify that one ontology. Today, OWL is used for the vast majority of ontologies. Therefore we disregard criteria that are based on the ontology language, such as expressivity, decidability, complexity, etc.

One example is the criteria *expandability* from (Gómez-Pérez, 2004). It is defined as the required effort to add new definitions without altering the already stated semantics. Since OWL is a monotonic language it is not possible to retract any inferences that have already been made. Thus the monotonicity of OWL guarantees a certain kind

of expandability for all ontologies in OWL. A complete list of methods given in this thesis is available in the appendix.

### 3.6.1 Accuracy

**Accuracy** is a criteria that states if the axioms of the ontology comply to the knowledge of the stakeholders about the domain. A higher accuracy comes from correct definitions and descriptions of classes, properties, and individuals. Correctness in this case may mean compliance to defined "gold standards", be it other data sources, conceptualizations, or even reality ((Ceusters and Smith, 2006) introduces an approach to use reality as a benchmark, i.e. if the terms of the ontology capture the intended portions of reality). The axioms should constrain the possible interpretations of an ontology so that the resulting models are compatible with the conceptualizations of the users.

For example, all inferences of an ontology should be true. When stating that the `foaf:knows` property is a superproperty of a `married` property, then this axiom would only be accurate if indeed all married couples know their respective spouses. If we find counterexamples (for example, arranged prenatal marriages), then the ontology is inaccurate.

The following methods in this thesis can be used to measure this criteria: Method 3: Hash vs slash (Page 71), Method 13: Querying for anti-patterns (Page 114), Method 14: Analysis and Examples (Page 125), Method 18: Class / relation ratio (Page 146), Method 19: Formalized competency questions (Page 154), Method 20: Formalized competency questions (Page 155), Method 21: Affirming derived knowledge (Page 157), Method 22: Expressive consistency checks (Page 160), and Method 23: Consistency checking with rules (Page 161).

### 3.6.2 Adaptability

**Adaptability** measures how far the ontology anticipates its uses. An ontology should offer the conceptual foundation for a range of anticipated tasks (ideally, on the Web, it should also offer the foundation for tasks not anticipated before). It should be possible to extend and specialize the ontology monotonically, i.e. without the need to remove axioms (note that in OWL, semantic monotonicity is given by syntactic monotonicity, i.e. in order to retract inferences explicit stated axioms need to be retracted). An ontology should react predictably and intuitively to small changes in the axioms. It should allow for methodologies for extension, integration, and adaptation, i.e. include required meta-data. New tools and unexpected situations should be able to use the ontology.

For example, many terms of the FOAF ontology (Brickley and Miller, 2005) are often used to describe contact details of persons. FOAF was originally designed to

describe social networks, but its vocabulary also allows to formalize address books of all kinds.

The following methods in this thesis can be used to measure this criteria: Method 6: URI declarations and punning (Page 75), Method 10: Blank nodes (Page 82), Method 13: Querying for anti-patterns (Page 114), Method 15: Stability (Page 140), Method 17: Maximum depth of the taxonomy (Page 145), Method 19: Formalized competency questions (Page 154), Method 21: Affirming derived knowledge (Page 157), Method 22: Expressive consistency checks (Page 160), and Method 23: Consistency checking with rules (Page 161).



### 3.6.3 Clarity

**Clarity** measures how effectively the ontology communicates the intended meaning of the defined terms. Definitions should be objective and independent of the context. Names of elements should be understandable and unambiguous. An ontology should use definitions instead of descriptions for classes. Entities should be documented sufficiently and be fully labeled in all necessary languages. Complex axioms should be documented. Representation choices should not be made for the convenience of the notation or implementation, i.e. the encoding bias should be minimized.

For example, an ontology may choose to use URIs such as `ex:a734` or `ex:735` to identify their elements (and may even omit the labels). In this case, users of the ontology need to regard the whole context of the elements in order to find a suitable mapping to their own conceptualizations. Instead, the URIs could already include hints to what they mean, such as `ex:Jaguar` or `ex:Lion`.

The following methods in this thesis can be used to measure this criteria: Method 1: Linked data (Page 67), Method 2: Linked data (Page 69), Method 3: Hash vs slash (Page 71), Method 4: Opaqueness of URIs (Page 73), Method 6: URI declarations and punning (Page 75), Method 7: Typed literals and datatypes (Page 78), Method 8: Language tags (Page 79), Method 9: Labels and comments (Page 81), Method 14: Analysis and Examples (Page 125), and Method 18: Class / relation ratio (Page 146).

### 3.6.4 Completeness

**Completeness** measures if the domain of interest is appropriately covered. All questions the ontology should be able to answer can be answered. There are different aspects of completeness: completeness with regards to the language (is everything stated that could be stated using the given language?), completeness with regards to the domain (are all individuals present, are all relevant concepts captured?), completeness with regards to the applications requirements (is all data that is needed present?), etc. Completeness also covers the granularity and richness of the ontology.

For example, an ontology to describe the nationalities of all members of a group

should provide the list of all relevant countries. Such closed sets in particular (like countries, states in countries, members of a group) can often be provided as an external ontology by an authority to link to, and thus promise completeness.

The following methods in this thesis can be used to measure this criteria: Method 3: Hash vs slash (Page 71), Method 6: URI declarations and punning (Page 75), Method 7: Typed literals and datatypes (Page 78), Method 9: Labels and comments (Page 81), Method 10: Blank nodes (Page 82), Method 11: XML validation (Page 86), Method 12: Structural metrics in practice (Page 101), Method 15: Stability (Page 140), Method 16: Language completeness (Page 141), Method 17: Maximum depth of the taxonomy (Page 145), and Method 19: Formalized competency questions (Page 154).

### 3.6.5 Computational efficiency

**Computational efficiency** measures the ability of the used tools to work with the ontology, in particular the speed that reasoners need to fulfill the required tasks, be it query answering, classification, or consistency checking. Some types of axioms may cause problems for certain reasoners. The size of the ontology also affects the efficiency of the ontology.

For example, using certain types of axioms will increase the reasoning complexity. But more important than theoretical complexity is the actual efficiency of the implementation used in a certain context. For example, it is known that number restriction may severely hamper the efficiency of the KAON2 reasoner (Motik, 2006), and should thus be avoided when that system is used.

The following methods in this thesis can be used to measure this criteria: Method 6: URI declarations and punning (Page 75), Method 7: Typed literals and datatypes (Page 78), Method 10: Blank nodes (Page 82), Method 12: Structural metrics in practice (Page 101), and Method 16: Language completeness (Page 141).

### 3.6.6 Conciseness

**Conciseness** is the criteria that states if the ontology includes irrelevant elements with regards to the domain to be covered (i.e. an ontology about books including axioms about African lions) or redundant representations of the semantics. An ontology should impose a minimal ontological commitment, i.e. specify the weakest theory possible. Only essential terms should be defined. The ontology's underlying assumptions about the wider domain (especially about reality) should be as weak as possible in order to allow the reuse within and communication between stakeholders that commit to different theories.

For example, an ontology about human resource department organization may take a naïve view on what a *human* actually is. It is not required to state if a human has a soul or not, if humans are the result of evolution or created directly by God, when

human life starts or ends. The ontology would remain silent on all these issues, and thus allows both creationists and evolutionists to use it in order to make statements about which department has hired whom and later exchange that data.

The following methods in this thesis can be used to measure this criteria: Method 5: URI reuse (Page 74), Method 10: Blank nodes (Page 82), Method 15: Stability (Page 140), Method 17: Maximum depth of the taxonomy (Page 145), Method 18: Class / relation ratio (Page 146), and Method 20: Formalized competency questions (Page 155).



### 3.6.7 Consistency

**Consistency** describes that the ontology does not include or allow for any contradictions. Whereas accuracy states the compliance of the ontology with an external source, consistency states that the ontology itself can be interpreted at all. Logical consistency is just one part of it, but also the formal and informal descriptions in the ontology should be consistent, i.e. the documentation and comments should be aligned with the axioms. Further ordering principles can be defined that the ontology has to be consistent with, such as the OntoClean constraints on the taxonomy (Guarino and Welty, 2002).

Note that within this thesis we will deal with logical consistency and coherence only superficially. There is an active research community in the area of ontology debugging, that covers discovering, explaining, and repairing errors that lead to inconsistency and incoherence, see for example (Parsia *et al.*, 2005; Lam, 2007; Haase and Qi, 2007).

An example for a non-logical inconsistency is the description of the element `ex:Jaguar` being *"The Jaguar is a feral cat living in the jungle."*, but having a logical axiom `ClassAssertion(ex:Car_manufacturer ex:Jaguar)`. Such discrepancies are often the result of distributed ontology engineering or a badly implemented change management procedures in ontology maintenance.

The following methods in this thesis can be used to measure this criteria: Method 3: Hash vs slash (Page 71), Method 4: Opaqueness of URIs (Page 73), Method 5: URI reuse (Page 74), Method 9: Labels and comments (Page 81), Method 12: Structural metrics in practice (Page 101), Method 13: Querying for anti-patterns (Page 114), Method 14: Analysis and Examples (Page 125), Method 16: Language completeness (Page 141), Method 21: Affirming derived knowledge (Page 157), Method 22: Expressive consistency checks (Page 160), and Method 23: Consistency checking with rules (Page 161).

### 3.6.8 Organizational fitness

**Organizational fitness** aggregates several criteria that decide how easily an ontology can be deployed within an organization. Tools, libraries, data sources, and other

ontologies that are used constrain the ontology, and the ontology should fulfill these constraints. Ontologies are often specified using an ontology engineering methodology or by using specific data sets. The ontology metadata could describe the applied methodologies, tools, and data sources, and the organization. Such metadata can be used by the organization to decide if an ontology should be applied or not.

For example, an organization may decide that all ontologies used have to align to the DOLCE upper level ontology (Gangemi *et al.*, 2002). This will help the organization to align the ontologies and thus reduce costs when integrating data from different sources.

The following methods in this thesis can be used to measure this criteria: Method 1: Linked data (Page 67), Method 2: Linked data (Page 69), Method 3: Hash vs slash (Page 71), Method 4: Opaqueness of URIs (Page 73), Method 5: URI reuse (Page 74), Method 8: Language tags (Page 79), Method 9: Labels and comments (Page 81), Method 11: XML validation (Page 86), and Method 19: Formalized competency questions (Page 154).

## 3.7 Methods

Evaluation methods either describe procedures or specify exactly the results of such procedures in order to gain information about an ontology, i.e. an ontology description. An evaluation method assesses specific features or qualities of an ontology or makes them explicit. The procedures and result specifications given in Part II are not meant to be implemented literally. Often such a literal or naïve implementation would lead to an unacceptably slow runtime, especially with mid-sized or big ontologies. Many of the methods in this thesis remain for now without an efficient implementation.

The relationship between criteria and methods is complex: criteria provide justifications for the methods, whereas the result of a method will provide an indicator for how well one or more criteria are met. Most methods provide indicators for more than one criteria, therefore criteria are a bad choice to structure evaluation methods.

A number of the methods define measures and metrics and also offers some upper or lower bounds for these metrics. Note that these bounds are not meant to be strict, stating that any ontology not within the bounds is bad. There are often perfectly valid reasons for not meeting those limits. These numbers should also not lead to the implementation of automatic fixes in order to implement changes to an ontology that make the ontology abide to the given limits, but nevertheless decrease the ontology's overall quality. The numbers given in the method descriptions are chosen based on evaluating a few thousand ontologies in order to discover viable margins for these values. In the case a certain measure or metric goes well beyond the proposed value but the ontology author or evaluator has good reasons, they should feel free to ignore that metric or measure or, better, explain in a rationale why it is not applicable in the

given case.

The Semantic Web is still in its infancy. It is to be expected that as we gather more experience with engineering and sharing ontologies, we will find better values for many of these bounds. It is also expected that further methods will be introduced and existing ones may become deprecated. The framework described here allows to accommodate for such changes. The set of methods is flexible and should be chosen based on the needs of the given evaluation.

Part II of this thesis describes evaluation methods. In order to give some structure for the description of the methods, the following section introduces different aspects of an ontology. These aspects provide the structure for Part II.



## 3.8 Aspects

An ontology is a complex, multi-layered information resource. In this section we will identify different aspects that are amenable to the automatic, domain- and task-independent verification of an ontology. Based on the evaluations of the different ontology aspects, evaluators can then integrate the different evaluation results in order to achieve an aggregated, qualitative ontology evaluation. For each aspect we show evaluation methods within the following chapters.

Each aspect of an ontology that can be evaluated must represent a degree of freedom (if there is no degree of freedom, there can be no evaluation since it is the only choice). So each aspect describes some choices that have been made during the design of the ontology. Some tools do not offer a degree of choice on certain aspects. In such cases evaluation methods for this aspect do not lead to useful insights. In turn, these tools should be evaluated in order to result in the best possible choice for those fixed aspects.

- **Vocabulary**. The vocabulary of an ontology is the set of all names in that ontology. Names can be URI references or literals, i.e. a value with a datatype or a language identifier. This aspect deals with the different choices with regards to the used URIs or literals (Chapter 4).
- **Syntax**. Web ontologies can be described in a number of different surface syntaxes. Often the syntactic description within a certain syntax can differ widely (Chapter 5).
- **Structure**. A Web ontology can be described by an RDF graph. The structure of an ontology is this graph. The structure can vary highly even describing semantically the same ontology. The explicitly given graph is evaluated by this aspect (Chapter 6).
- **Semantics**. A consistent ontology is interpreted by a non-empty, usually infinite set of possible models. The semantics of an ontology are the common

characteristics of all these models. This aspect is about the formal meaning of the ontology (Chapter 7).

- **Representation**. This aspect captures the relation between the structure and the semantics. Representational aspects are usually evaluated by comparing metrics calculated on the RDF graph with features of the possible models as specified by the ontology (Chapter 8).

- **Context**. This aspect is about the features of the ontology when compared with other artifacts in its environment, which may be, e.g. a data source that the ontology describes, a different representation of the data within the ontology, or formalized requirements for the ontology in form of competency questions or additional semantic constraints (Chapter 9).

Note that in this thesis we assume that logical consistency or coherence of the ontology is given, i.e. that any inconsistencies or incoherences have been previously resolved using other methods. There is a wide field of work discussing these logical properties, and also well-developed and active research in debugging inconsistency and incoherence, e.g. (Parsia *et al.*, 2005; Lam, 2007; Haase and Qi, 2007).

Ontologies are **inconsistent** if they do not allow any model to fulfill the axioms of the ontology. **Incoherent** ontologies have classes with a necessarily empty intension (Haase and Qi, 2007). Regarding the evaluation aspects, note that the vocabulary, syntax, and structure of the ontology can be evaluated even when dealing with an inconsistent ontology. This also holds true for some parts of the context. But semantic aspects – and thus also representational and some contextual aspects – can not be evaluated if the ontology does not have any formal models.

# Part II

# Aspects


4 Vocabulary 65

5 Syntax 83

6 Structure 99

7 Semantics 127

8 Representation 143

9 Context 151

## Chapter 4

# Vocabulary

O, be some other name!
What's in a name?
that which we call a rose
By any other name
would smell as sweet

*(William Shakespeare, 1564–1616, Romeo and Juliet (Shakespeare, 1597))*

Evaluating the vocabulary aspect of an ontology means to evaluate the names used in the ontology. In this chapter we discuss methods for evaluating the vocabulary of an ontology, and provide a comparison for some of the values based on a large corpus of Web ontologies.

The **vocabulary** of an ontology is the set of all *names* used in it. Names can be either URIs or literals. The set of all URIs of an ontology is called the *signature* of the ontology (and is thus the subset of the vocabulary without the literals). URIs are discussed in Section 4.1. *Literals* are names that are mapped to a concrete data value, i.e. instead of using a URI to identify an external entity, literals can be directly interpreted. Literals are presented in Section 4.2. Finally, we will also discuss *blank nodes*, i.e. unnamed entities within ontologies (Section 4.3).

## 4.1 URI references

Most names in ontologies are URI references (Uniform Resource Identifier, (Berners-Lee *et al.*, 2005)). URIs are more generic forms of URLs (Uniform Resource Locator,

(Berners-Lee *et al.*, 1994)). Unlike URLs, URI references are not limited to identifying entities that have network locations, or use other access mechanisms available to the computer. They can be used to identify anything, from a person over an abstract idea to a simple information resource on the Web (Jacobs and Walsh, 2004).

An URI reference should identify one specific resource, i.e. the same URI reference should not be used to identify several distinct resources. A URI reference may be used to identify a collection of resources, and this is not a contradiction to the previous sentence: in this case the identified resource is the *collection* of resources, and thus a resource of its own. Classes and properties in OWL ontologies are also resources, and thus are identified by a URI reference.

A particular type of resources are *information resources*. Information resources are resources that consist of information, i.e. the digital representation of the resource captures the resource completely. This means that an information resource can be copied without loss, and it can be downloaded via the Internet. Therefore information resources can be located and retrieved with the use of a URL. An example of an information resource is the text of Shakespeare's *"Romeo & Juliet"* (from which this chapter's introductory quote is taken) which may be referenced, resolved, and downloaded via its URL `http://www.gutenberg.org/dirs/etext97/1ws1610.txt`

*Non-information resources* can not be downloaded via the Internet. There may be metadata about non-information resources available, describing the resource. An example is the book *"Romeo & Juliet"*: the book itself can not be downloaded via the Internet (in contrast to its content). There may be metadata stating e.g. the weight or the prize of the book. In order to state such metadata we need to be able to identify the book, e.g. using its ISBN number (ISO 2108, 2005). In this case we can not use an URL: since the resource is not an information resource, it can not be located on the Web, and thus can not be accessed with an URL. Nevertheless it may (and should) have an URI in order to identify the resource.

Non-information resources and information resources are disjoint classes (i.e. no information resource can at the same time be a non-information resource and vice versa). A further formalization of information resources can be found e.g. in the *"Functional Requirements for Bibliographic Records"* ontology FRBR (Tillett, 2005), widely used in the bibliographic domain, or in the DOLCE-based *"Ontology of Information Objects"* (Guarino, 2006).

### 4.1.1 Linked data

URI references are strings that start with a protocol. If the protocol is known and implemented by the ontology based application, then the application may *resolve* the URI, i.e. use the URI according to the protocol in order to find more information about the identified resource. In case the URI is an URL, the identified information resource can be accessed and downloaded by using the protocol.

URI references consist of an URI with an optional fragment identifier. Most URI references in Web ontologies are indeed using a protocol that can be resolved by the machine in order to fetch further information about the given URI reference. Most commonly this is achieved by using the HyperText Transport Protocol HTTP (Fielding *et al.*, 1999). We have examined the Watson corpus (see Section 11.3) to figure out the usage of protocols on the Web. The Watson corpus contains 108,085,656 URIs. Only 491,710 (0.45%) of them are URIs not using the HTTP protocol.

Other prominent protocols besides HTTP are *file* (37,023 times; for local files), *mailto* (22,971 times; for email adresses), *mid* (13,448 time; for emails), *irc* (3,260 times; for internet relay chat channels and user ids), *ftp* (1,716 times, for the file transfer protocol), *tel* (703 times, for telephone numbers), or *https* (186 times; for secure HTTP connections). Sometimes these protocols are just mistyped (such as *hhpt*).

Sometimes, QNames (see Section 5.2) can mistakenly be interpreted as an URI, especially in XML serialization (Jacobs and Walsh, 2004). The namespace prefix will then be interpreted as the protocol scheme. We discovered that this makes up a good deal of the Non-HTTP protocols: the two most prominent non-HTTP schemes were *UnitOfAssessment* (46,691 times, or 9.5% of all Non-HTTP URIs) and *Institute* (42,331 times, or 8.6%). Both of them are meant to represent namespace prefixes in their ontologies, not URI schemes. These are errors in the ontology that can be easily discovered using Method 1. Further examples for mis-interpreted namespace prefixes include *xs* (19,927 times; used for XML schema datatypes), *rdf* (272 times), *rdfs* (7 times), and *owl* (84 times).

**Method 1 (Check used protocols)**
All URIs in the ontology are checked to be well-formed URIs. The evaluator has to choose a set of allowed protocols for the evaluation task. The usage of any protocol other than HTTP should be explained. All URIs in the ontologies have to use one of the allowed protocols.

Resolving an HTTP URI reference will return an HTTP response code and usually some content related to the URI reference. Certain HTTP responses imply facts about the used URI reference. These facts are given in Table 4.1, e.g. a 303 response on an URI without a fragment identifier implies the equality of the requested URI and the URI returned in the location field of the response. If the response code is a 200, it has even stronger implications: then the URI is actually the name of the served resource, i.e. the URI is a information resource that can (and is) accessible over the

Table 4.1: An overview of what different response codes imply for the resolved HTTP URI reference $U$. $I$ is the information resource that is returned, if any. $L$ is the URI given in the location field of the response. The table covers the most important responses only, the others do not imply any further facts.

| Response code | $U$ has a fragment identifier | $U$ has no fragment identifier |
|---|---|---|
| 200 OK | $I$ should describe $U$ | $U$ is the name of $I$.<br>$I$ is an information resource. |
| 301 Moved Permanently | $L$ should describe $U$ | $L$ and $U$ are names of $I$.<br>$I$ is an information resource. |
| 303 See Other | $L$ should describe $U$ | $L$ should describe $U$ |
| *Any other* | Nothing implied for $I$ | Nothing implied for $I$ |

Web (Sauermann and Cyganiak, 2008). Section 4.1.2 gives more details on fragment identifiers and how they can be used for evaluation.

Both OWL classes and OWL properties are not information resources. With Table 4.1 this allows us to infer that in OWL DL both class and property names (without fragment identifiers) should not return a 200 or a 301 when resolved via the HTTP protocol. In OWL 2 this is not true anymore, since punning allows URIs to be individuals, properties, and classes at the same time (Motik, 2007). But as we will discuss later, punning should be avoided (see Section 4.1.5).

The table also lists the cases when the served resource should describe the name. We can easily check if this is the case, if the description is sufficiently useful, and if it is consistent with the knowledge we already have about the resource.

We have tested all HTTP URIs from the Watson EA corpus (see Section 11.3). For the slash namespaces URIs, we got 14,856 responses. Figure 4.1 shows the distribution of the response codes. It can be easily seen that for the vast majority (about 75%) of URIs we get `200 OK`, which means that the request returns a document. We conclude that the current Semantic Web is indeed a Web of metadata describing the connections between resources, i.e. documents, on the Web. 85% of the tested URIs return a `200` or `3XX` response.[1] We got 509 responses on the URIs with hash namespaces, of which significantly more (about 85%) responded with a `200 OK`. This shows that in general hash URIs are better suited for terminological entities, and there should be good reasons for using a slash namespace.

*Significance of the difference between hash and slash URIs.* For significance testing we apply the two-proportion z-test. Let the null hypothesis be $P_s = P_h$, i.e. the probability for a slash URI to return a `200 OK` being the same as the probability for a hash

[1] A `3XX` response means an HTTP response in the range `300-307`, a group of responses meaning redirection (Fielding *et al.*, 1999)

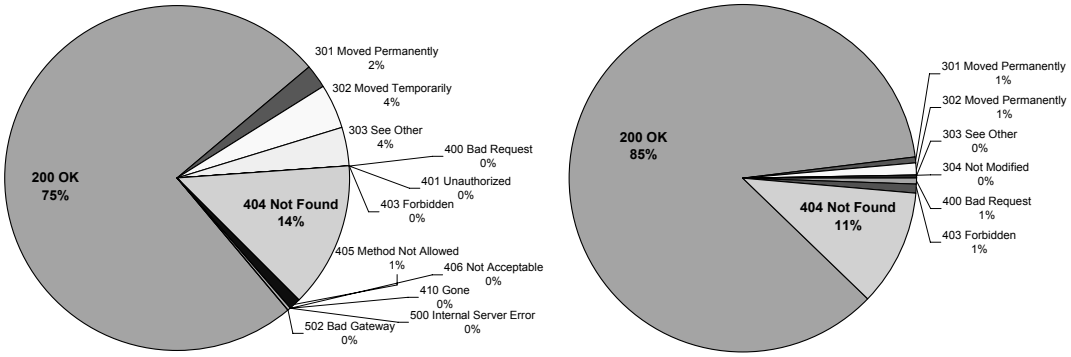


Figure 4.1: Distribution of the HTTP response codes on the HTTP URIs from the Watson EA corpus. The left hand side shows the slash URIs, the right hand side hash URIs.

URI. According to our tests on the Watson EA corpus, we set $p_s = n_s/t_s \approx 0.7513$ and $p_h = n_h/t_h \approx 0.8585$ (with sample sizes $n_s = 14,856$ resp. $n_h = 509$ and $t_s = 11,161$ resp. $t_h = 437$ positive samples, i.e. URIs returning `200 OK` codes). We calculate the pooled sample proportion $\hat{p} = \frac{p_s n_s + p_h n_h}{n_s + n_h} \approx 0.7548$. The standard error results in $e = \sqrt{\hat{p}(1-\hat{p})(1/n_s + 1/n_h)} \approx 0.0194$. The test statistic is a z-score defined as $z = \frac{p_s - p_h}{e} \approx -5.5316$. From that it follows that the probability that the null hypothesis is true is $p < 0.0001$, which means the result is statistically highly significant. □

Names from the same slash namespace should always return the same response code. Differing response codes indicate some problems with the used terms. For example, the social website *LiveJournal*[2] exports FOAF profiles about their members, including their social contacts, interests, etc. Some of the terms return a `303 See Other` (e.g. `foaf:nick`, `foaf:knows`, `foaf:Person`), whereas others return a `404 Not Found` (e.g. `foaf:tagLine`, `foaf:member_name`, `foaf:image`). Investigating this difference uncovers that the first set are all terms that are indeed defined in the FOAF ontology, whereas the second set of terms does not belong to the FOAF ontology.

**Method 2 (Check response codes)**
For all HTTP URIs, make a `HEAD` call (or `GET` call) on them. The response code should be `200 OK` or `303 See Other`. Names with the same slash namespace should return the same response codes, otherwise this indicates an error.

[2] http://www.livejournal.com

Note that this method can only be applied after the ontology has been published and is thus available on the Web.

In summary, the usage of resolvable URI references allows us to use the Web to look up more information on a given URI. This can help to discover available mappings, or to explore and add new information to a knowledge based system. This is the major advantage of using the Semantic *Web* instead of simply an ontology based application.

### 4.1.2 Hash vs slash

There was a long running debate in the Semantic Web community on the usage of fragment identifiers in URI references. The basic question is on the difference between using `http://example.org/ontology#joe` and `http://example.org/ontology/joe` in order to refer to a non-information resource. The former type of URI is called a hash URI (since the local part is separated by the hash character # from the namespace), the latter type a slash URI (since the local part is separated by the slash character / from the namespace). The discussion was resolved by the W3C Technical Advisory Group (Lewis, 2007; Jacobs and Walsh, 2004; Berrueta and Phipps, 2008).

When resolving a hash URI, only the namespace is resolved. All hash URIs with the same namespace thus resolve to the same resource. This has the advantage that the ontology can be downloaded in one pass, but it also has the disadvantage that the file can become very big. Therefore, terminological ontologies and ontologies with a closed, rarely changing, and rather small set of individuals (e.g. a list of all countries) would use hash URIs, whereas open domains with often changing individuals often use slash URIs (see for example in Semantic MediaWiki, Section 10.2.3).

We analyzed the Watson corpus to see if there is a prevalence towards one or the other on the Web. We found 107,533,230 HTTP URIs that parse. 50,366,325 were hash URIs, 57,166,905 were slash URIs. Discounting repetitions, there were 5,815,504 different URIs in all, 2,247,706 of them hash URIs, 3,567,789 slash URIs.

Regarding their distribution over namespaces, there are much bigger differences: we find that there are only 46,304 hash namespaces compared to 2,320,855 slash namespaces. The hash namespaces are, in average, much bigger than the slash namespaces. 2,197,267 slash namespaces (94.67%) contain only a single name, whereas only 16,990 hash namespaces (36.69%) contain only one name. On the other extreme, only 253 slash namespaces (0.01%) contain more than a 100 names, in contrast to 2,361 hash namespaces (5.10%) that have more than 100 names. Still, as we can see in Table 4.2, the namespaces with the biggest names are slash namespaces, being several times as big as the biggest hash namespace. What does that mean?

Table 4.2: The five hash and slash namespaces with the biggest number of names.

| **Hash namespace** | **# names** |
|---|---|
| http://purl.org/obo/owl/FMA# | 75,140 |
| http://www.loa-cnr.it/ontologies/OWN/OWN.owl# | 65,975 |
| http://www.hero.ac.uk/rae/# | 64,799 |
| http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl# | 58,077 |
| http://www.geneontology.org/owl/# | 29,370 |
| **Slash namespace** | **# names** |
| http://www.livejournal.com/ | 454,376 |
| http://www.ecademy.com/ | 104,987 |
| http://www.deadjournal.com/ | 79,093 |
| http://www.aktors.org/scripts/ | 41,312 |
| http://www.hi5.com/profile/ | 32,652 |

Slash namespaces are used in two very different ways in ontologies. First, they are used to identify resources on the Web and to provide metadata about them. This explains the huge number of slash namespaces with only a few names: only 3,142 slash namespaces (0.14%) have more than 10 names (compared to 4,575 hash namespaces, or 9.88%). In this case, when providing metadata, the names are often distributed all over the Web, and thus introduce many different namespaces. The other usage of slash namespaces is for collectively built ontologies: since hash namespaces reside in one single file, they can not deal well with very dynamic vocabularies, that add and remove names all the time and change their definitions. Looking at the five biggest slash namespaces in Table 4.2, we see that four of five namespaces belong to social networks (*aktors.org* is the Website of a UK-based Semantic Web research project). Looking at the hash namespaces, we also see huge ontologies, but they represent much more stable domains, providing vocabularies for the life sciences (FMA, Gene ontology, NCI thesaurus), a vocabulary for education assessment (RAE), and an OWL translation of WordNet (OWN). None of them are interactively built on the Web by an open community, but rather curated centrally by editors.

**Method 3 (Look up names)**
For every name that has a hash namespace make a `GET` call against the namespace. For every name that has a slash namespace make a `GET` call against the name. The content type should be set correctly. Resolve redirects, if any. If the returned resource is an ontology, check if the ontology describes the name. If so, $N$ is a *linked data conformant name*. If not, the name may be wrong.

### 4.1.3 Opaqueness of URIs

URIs should be named in an intuitive way. Even though the URI standard (Berners-Lee *et al.*, 2005) states that URIs should be treated opaque and no formal meaning should be interpreted into them besides their usage with their appropriate protocols, it is obvious that a URI such as `http://www.aifb.kit.edu/id/Rudi_Studer` will invoke a certain denotation in the human reader: the user will assume that this is the URI for Rudi Studer at the AIFB, and it would be quite surprising if it were the URI for the movie Casablanca.

On the other hand, an URI such as `http://www.aifb.kit.edu/id/p67` does not have an intuitive denotation, and so they become hard to debug, write manually, and remember. Their advantage is that they do not depend on a specific natural language. An unfortunate number of tools still displays the URI when providing a user interface to the ontology. Therefore intuitive URIs, that actually identify the entities to which their names allude to (for the human reader) and that have readable URIs, should be strongly preferred. Also, since URIs unlike labels should not change often (Sauermann and Cyganiak, 2008), it is important to catch typos when writing URIs.

URIs should follow a naming convention. When using natural language based names, the local name of classes may use plural (`Cities`) or singular forms (`City`), the local name of properties may use verbs (`marries`), verbal phrases (`married_to`), nouns (`spouse`), or nominal phrases (`spouse_of` or `has_spouse`). All of these naming conventions have certain advantages and disadvantages: phrases make the direction of the property explicit and thus reduce possible sources of confusion (given the triple `Aristotle Teacher Plato`, is it immediately clear who the teacher is, and who the student?). But using simple nouns can help with form based interfaces such as Tabulator (Berners-Lee *et al.*, 2006a), a Semantic Web browser. Tabulator also constructs a name for the inverse property by appending " *of*" to the word, e.g. the inverse of *Teacher* would be *Teacher of*.

Capitalization and the writing of multi-word names should also be consistent. The ontology authors should decide which names to capitalize (often, classes and individuals are capitalized, whereas properties are not). Multi-word names (i.e. names that consist of several words, like *Wide square*) need to escape the whitespace between the words (since whitespaces are not allowed in URIs). Often this is done by camel casing (i.e. the space is removed and every word after a removed space starts with a capital letter, like `WideSquare`), by replacing the space with a special character (often an underscore like in `Wide_square`, but also dashes or fullstops), or by simple concatenation (`Widesquare`).

Many of these choices are just conventions. The used naming conventions should be noted down explicitly. Metadata about the ontology should state the naming convention for a given vocabulary. Many of the above conventions can then be tested automatically. It is more important to consistently apply the chosen convention than

to choose the best convention (especially, since the latter is often unclear).

In addition, URIs on the Semantic Web should follow also the same rules that URIs on the hypertext Web should follow. These are (Berners-Lee, 1998):

- don't change (i.e. don't change what the resource refers to, nor change the URI of a resource without redirecting the old URI)
- be human guessable (i.e. prefer `http://example.org/movie/The_Matrix` over `http://example.org/movie/tt0133093`)
- be reasonably short (which also means not to use deep hierarchical nesting, but a rather flat structure)
- don't show query parameters (i.e. don't use URIs such as `http://example.org/interest?q=Pop+Music`, use `http://example.org/interest/pop_music` instead)
- don't expose technology (e.g. don't use file extensions like in `http://example.org/foaf.rdf`, use `http://example.org/foaf` instead)
- don't include metadata (e.g. don't add the author or access restrictions into the URI, e.g. as in `http://example.org/style/timbl/private/uri`, just use `http://example.org/style/uri`. Authorship and access level may change)

**Method 4 (Check naming conventions)**
A proper naming can be checked by comparing the local part of the URI with the label given to the entity or by using lexical resources like Wordnet (Fellbaum, 1998). Formalize naming conventions (like multi-word names and capitalization) and test if the convention is applied throughout all names of a namespace. Check if the URI fulfills the general guidelines for good URIs, i.e. check length, inclusion of query parameters, file extensions, depth of directory hierarchy, etc.)

Note that only local names from the same *namespace*, not all local names in the *ontology*, need to consistently use the same naming convention, i.e. names reused from other ontologies may use different naming conventions.

### 4.1.4 URI reuse

In order to ease sharing, exchange, and aggregation of information on the Semantic Web, the reuse of commonly used URIs proves to be helpful (instead of introducing new names). At the time of writing many domains still do not provide an exhaustive

lexicon of URIs yet, but projects such as *Freebase*[3] or *DBpedia* (Auer and Lehmann, 2007) already offer a huge amount of identifiers for huge domains. Some domains, like life sciences, music, computer science literature, or geography already have very exhaustive knowledge bases of their domains. These knowledge bases can easily be reused.

Analyzing the Watson EA corpus, we find that 75% of the ontologies use 10 or more namespaces, in 95.2% of the ontologies the average number of names per namespaces is lower than 10, in 76.5% it is lower than 3, in 46.4% lower than 2. This means that most ontologies use many namespaces, but only few names from each namespace. Considering knowledge bases this makes perfect sense: in their FOAF files persons may add information about their location by referencing the Geonames ontology, and about their favourite musician referencing the MusicBrainz ontology. Terminological ontologies often reference the parts of external ontologies relevant to them in order to align and map to their names.

**Method 5 (Metrics of ontology reuse)**
We define the following measures and metrics:

- Number of namespaces used in the ontology $N_{NS}$
- Number of unique URIs used in the ontology $N_{UN}$
- Number of URI name references used in the ontology $N_N$ (i.e. every mention of a URI counts)
- Ratio of name references to unique names $R_{NU} = \frac{N_{UN}}{N_N}$
- Ratio of unique URIs to namespaces $R_{UNS} = \frac{N_{UN}}{N_{NS}}$

Check the following constraints. The percentages show the proportion of ontologies that fulfill this constraint within the Watson EA corpus, thus showing the probability that ontologies not fulfilling the constraint are outliers.

- $R_{NU} < 0.5$ (79.6%)
- $R_{UNS} < 5$ (90.3%)
- $N_{NS} \geq 10$ (75.0%)

[3] http://www.freebase.com

### 4.1.5 URI declarations and punning

Web ontologies do not require names to be declared. This leads to the problem that it is impossible for a reasoner to discern if e.g. `ex:Adress` is a new entity or merely a typo of `ex:Address`. This can be circumvented by requiring to *declare* names, so that tools can check if all used names are properly declared. This further brings the additional benefit of a more efficient parsing of ontologies (Motik and Horrocks, 2006).

The declarations are axioms, stating not only that a name exists but also its type, i.e. if it is declared as a class, an individual, a datatype, object or annotation property. This feature was introduced in OWL 2, and thus does not yet appear in ontologies outside of test cases.

**Method 6 (Check name declarations)**
Check for every URI if there exists a declaration of the URI. If so, check if the declared type is consistent with the usage. This way it is possible to detect erroneously introduced punning.

In OWL 1 DL the set of class names, individual names, and property names were disjoint. This restriction was removed in OWL 2, and now it is allowed to use the names for either the individual, the property or the class. Based on the position in the axiom it is always clear which type of entity the name refers to. There are good reasons to allow punning: for example, the entity *lion*, depending on the context, may represent the individual *lion* that is of the type *species*, or it may be the type of the lion *Simba* (Berrueta and Phipps, 2005; Motik, 2007). There is no necessity to introduce different names for the two, or to render the merger of two ontologies inconsistent where *lion* is used as a class in the one ontology, and as an individual in the other. Nevertheless, punning may cause confusion with the user, especially since most tools are not yet well equipped to deal with punning. Punning should only be carefully applied.

## 4.2 Literals

Ontologies often contain **literals** that represent **data values**. These data values can be very varied, e.g. numbers (such as *4.75*), points in time (such as *2:14 pm CEST on 6th March 2009*), or strings (e.g. *Jaguar*). The importance of literals on the Semantic Web can be shown by their sheer number of occurrences: the Watson corpus consists of 59,749,786 triples, and 26,750,027 of them (42.8%) include a literal.

Anything that can be represented by a literal could also be represented by an URI. For example, we could introduce the URI *ex:Number4dot75* to be the URI to represent

the number *4.75*. Using OWL Full, we could state that the literal and the URI are the same individual. Often it is more convenient to use a literal instead of a URI, especially since their meaning is already agreed on. Literals have the disadvantage that in triples they are only allowed to be objects. This means that we cannot make statements about literals directly, e.g. say that *4* is an instance of the class `ex:EvenNumber`. This can be circumvented by using URIs as proxies for data values. It is currently discussed to drop this limitation and to allow literals to also be subjects in triples.

Literals can be typed (see Section 4.2.1) or plain. A plain literal may be tagged with a language tag (Section 4.2.2). The standard does not allow literals to be both typed and language tagged. Language tagged literal would often make little sense: the integer *4* is the same number regardless of the language. Since it makes sense for text, the specification for the new data type `rdf:text` allows for language tagged typed text literals (Bao *et al.*, 2009b).

The RDF standard states that plain literals denote themselves (Hayes, 2004), i.e. the plain literal *Jaguar* denotes the ordered list of characters *Jaguar*. Most of the literals on the Web are plain literals – only 1,123,704 (4.2%) of them are typed.

### 4.2.1 Typed literals and datatypes

A typed literal is a pair of a lexical representation and a data type. The data type is given by an URI that defines the interpretation of the lexical representation. Most ontologies use data types defined by the XML Schema Definition (Fallside and Walmsley, 2004). The OWL standard requires all tools to support `xsd:string` and `xsd:integer` (Bechhofer *et al.*, 2004) and names further recommended data types. OWL 2 extends the number of required data types considerably, adding further data types for numbers, text, boolean values, binary data, URIs, time instants, and XML literals. OWL 2 also adds the possibility to constrain data type literals by facets (Motik *et al.*, 2009b).

Figure 4.2 shows the most often used data types. The most often used data types all belong to the set of recommended data types by the specification. The only two non-recommended data types that are used more often than a hundred times are from a deprecated XML schema version[4] and from the W3C's calendar working group.[5]

A fairly common error in data types is to use a namespace prefix (`xs:string` appears 19,927 times, `xsd:string` 518 times). Other common errors include misspelling of the data type URIs (e.g. forgetting the hash, or miscapitalizing the local name) or using deprecated versions of the XML schema.

The Semantic Web standards allow to define new custom data types, but this option is very rarely used. It makes it nevertheless hard to automatically discover if a data type URI is just an unknown data type, or if it is indeed a typo. Data type URIs

[4] `http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger`, used 157 times
[5] `http://www.w3.org/2002/12/cal/icaltzd#dateTime`, used 128 times

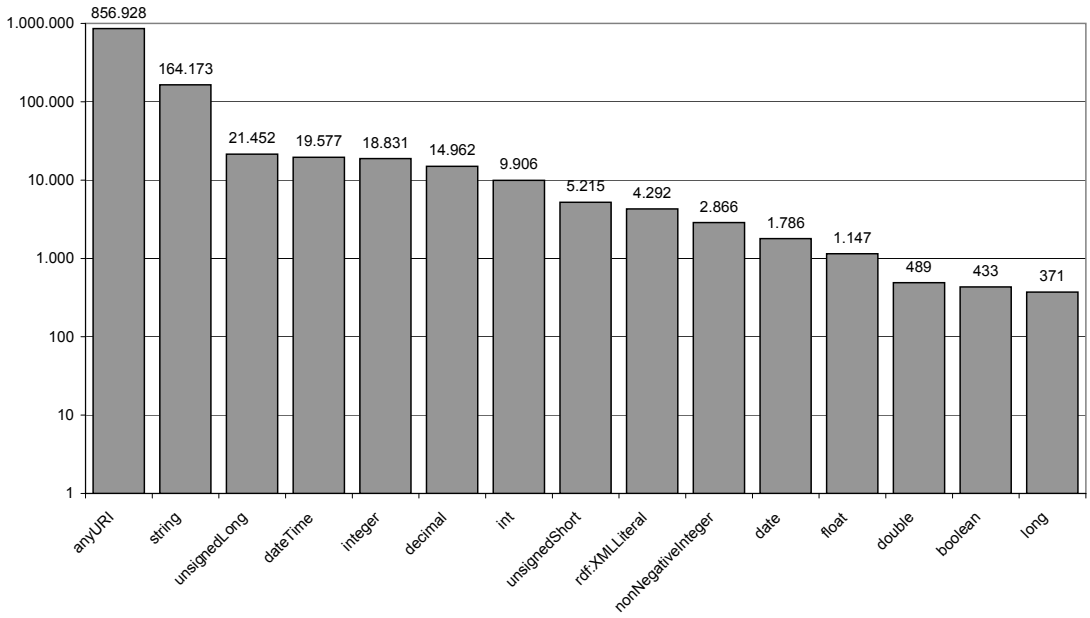


Figure 4.2: The fifteen most often used data types in the Watson corpus. Note the logarithmic scale.

should be resolvable just as all other URIs and thus allow a tool to make an automatic check. When evaluating ontologies, the evaluator should decide on a set of allowed data types. This set depends on the use case and the tools being used. All tools should be able to deal with the data types in this set. This closed set will help to discover syntactic errors in the data type declarations of the literals in the ontology.

It should be checked if all used data types in the ontology are understood by the tools dealing with these data types. That does not mean that all tools need to understand all the used data types. A tool may be perfectly capable of dealing with an unknown data type as far as the task of the tool itself is concerned. For example, a tool used for converting RDF/XML-serialization into OWL Abstract Syntax does not need to understand the data types. A reasoner who needs to check equality of different values on the other hand needs to understand the used data types.

The second check that is relevant for typed literals is to check if the literals are syntactically correct for the given data type. A typed literal of the form `"Four"` and data type `http://www.w3.org/2001/XMLSchema#integer` is an error and needs to be corrected. For this it is important for the evaluation tool to be able to check the syntactic correctness of all used data types. This should be considered when choosing the set of allowed data types. Otherwise it will be hard to discover simple syntactic errors within literals.

**Method 7 (Check literals and data types)**
A set of allowed data types should be created. All data types beyond those recommended by the OWL specifications should be avoided. Creating a custom data type should have a very strong reason. `xsd:integer` and `xsd:string` should be the preferred data types (since they have to be implemented by all OWL conformant tools).

Check if the ontology uses only data types from the set of allowed data types. All typed literals must be syntactically valid with regards to their data type. The evaluation tool needs to be able to check the syntactical correctness of all allowed data types.

### 4.2.2 Language tags

Language tags can be used on plain literals to state the natural language used by the literal. This enables tools to choose to display the most appropriate literals based on their user's language preferences. An example for a literal with a language tag is `"university"@en` or `"Universität"@de`. Language tags look rather simple, but are based on a surprisingly large set of specifications and standards.

Language tags are specified in the IETF RFC 4646 (Phillips and Davis, 2006b). IETF RFC 4647 specifies the matching of language tags (Phillips and Davis, 2006a). Language tags are based on the ISO codes for languages, if possible taking the Alpha-2 code (i.e. two ASCII letters representing a language) as defined by (ISO 639-1, 2002), otherwise the Alpha-3 code (three ASCII letters representing a language) defined by (ISO 639-2, 1998). Not all languages have an Alpha-2 code. The specification allows to use Alpha-3 codes only if they do not have an Alpha-2 code. For example, to state that the literal *Gift* is indeed the English word, we would tag it with `en`, the Alpha-2 code for the English language. If we wanted to state that it is a German word, we would have tagged it with `de`, the Alpha-2 code for German. If it were the Middle English word, it would have to be tagged with `enm` (since no two letter code exists).

Language tags can be further refined by a script, regional differences, and variants. All these refinements are optional. The script is specified using ISO codes for scripts (ISO 15924, 2004). To state that we use Russian with a latin script would be `ru-latn`. Every language has a default script, that should not be specified when used, e.g. `en` always assumes to be `en-latn`, i.e. English is written in latin by default. The IANA registry maintains a complete list of all applicable refinement subtags, and also specifies the default scripts for the used languages.[6] Following the optional script parameter,

[6] `http://www.iana.org/assignments/language-subtag-registry`

the language can be specified to accommodate regional differences. The codes for the regions are based on either the countries and regions ISO codes (ISO 3166, 1999) or, alternatively, on the UN M.49 numeric three-digits codes (UN M.49, 1998). English as spoken in Hong Kong would either be defined as `en-hk` or `en-344`. The regional modifiers should only be used when needed, and preferably omitted. For example, instead of using `ja-jp` for *Japanese as spoken in Japan* the tag `ja` would be preferred. Finally, a relatively small number of further variants can be specified defined for special cases such as historic language deviations, sign languages, and similar.

It is also possible to define private language tags or tag refiners, denoted by an `x`. So one could use `en-x-semweb` as the language tag for the kind of English that is spoken by the Semantic Web research community, where certain terms have meanings deviating from standard English. Private tags should be avoided, and indeed, our analysis of the Watson corpus shows that none are used.

Language tags are case insensitive. So it does not matter if one uses `en-us`, `EN-US`, `en-US`, `En-uS`, or any other combination of upper and lower case characters. On the Semantic Web it seems to be usual to use lower case characters only, with less than 200 occurrences of upper case codes.

We examined the usage of language tags on the Semantic Web to find if the standards are applied correctly. For such a complex mesh of standards we found surprisingly few errors. All in all, 17,313,981 literals have a language tag (67.6% of all plain literals). English (`en`) was by far the most often used tag, applied 16,767,502 times (96.8%), followed by Japanese (`ja`) with 519,191 tags (3.0%). The other languages are less widely used by far, Suomi (`fi`) is used 9,759 times, German (`de`) 4,893 times, French (`fr`) 1,810 times, and Spanish (`es`) 866 times.

The most often applied refined tags are `en-us` with 2,284 tags, `en-gb` 1,767 times, and `fr-fr` 288 times. The latter could be regarded as redundant, since `fr` would be sufficient. Further inappropriate usages or errors that can be found in the language tags are `en-uk` (it should be `en-gb`, used 90 times), `frp` (undefined tag, used 30 times), and `jp` (should be `ja`). In summary, with respect to the complexity of the used standards, the number of errors regarding language tags is extremely low. This probably stems from the fact that most of the advanced features are never used: no single tag specifies a script (besides one example literal) or a variant, and only a handful of tags specify a region, never using the UN M-49 standard (besides one example literal)[7] but always ISO 3166 codes.

[7] Both cited example literals are based on the W3C's write up on its internationalization effort, highlighting the possibilities of languages tags. See here: http://www.w3.org/International/articles/language-tags/Overview.en.php

**Method 8 (Check language tags)**
Check that all language tags are valid with regards to their specification. Check if the shortest possible language tag is used (i.e. remove redundant information like restating default scripts or default regions). Check if the stated language and script is actually the one used in the literal.

Check if the literals are tagged consistently within the ontology. This can be checked by counting $n_l$, the number of occurrences of language tag $l$ that occurs in the ontology. Roughly, $n_l$ for all $l$ should be the same. Outliers should be inspected.

### 4.2.3 Labels and comments

Labels are used in order to provide a human readable name for an ontological entity. Every ontological entity should have labels in all relevant languages. Almost none of the ontologies in the Watson corpus have a full set of labels in more than one language, i.e. most ontologies are not multi-lingual. Thus they miss a potential benefit of the Semantic Web, i.e. the language independence of ontologies. Comments add further human-readable explanations to a specific entity, and should also be language tagged.

Labels and comments should follow a style guide and be used consistently. A style guide should define if classes are labeled with plural or singular noun, if properties are labeled with nouns or verbs, and under what circumstances comments should be used. Labels and comments should never use camel case or similar escape mechanisms for multi word terms, but instead simply use space characters (or whatever is most suitable for the given language). I.e. an URI `http://example.org/LargeCity` should have a label `"large city"@en`. External dictionaries such as WordNet (Fellbaum, 1998) can be used to check consistency with regards to a style guide.

In an environment where ontologies are assembled on the fly from other ontologies (Alani, 2006), the assembled parts may follow different style guides. The assembled ontology will then not adhere to a single style guide and thus offer an inconsistent user interface. It is not expected that a single style guide will become ubiquitous on the whole Web. Instead, an ontology may specify explicitly what style guide it follows, and even provide labels following different style guides. For example, the SKOS ontology (Miles and Bechhofer, 2009) offers more specific subproperties for labels such as `skos:prefLabel`. This would allow to introduce a subproperty of label that is style guide specific, which would in return allow for the consistent display of assembled ontologies.

Even when subproperties of `rdfs:label` are defined, there should always be one label (per supported language) given explicitly by using `rdfs:label` itself. Even

though this is semantically redundant, many tools (especially visualization tools) do not apply reasoning for fetching the labels of an entity but simply look for the explicit triple stating the entity's label.

**Method 9 (Check labels and comments)**
Define the set of relevant languages for an ontology. Check if all label and comment literals are language tagged. Check if all entities have a label in all languages defined as being relevant. Check if all entities that need a comment have one in all relevant languages. Check if the labels and comments follow the style guide defined for the ontology.



## 4.3 Blank nodes

**Blank nodes** are an RDF feature that allows to use a node in the RDF graph without giving it a URI. This way the node can only be indirectly referenced (if at all), for example by using an inverse functional property. Blank nodes relieve the author of an RDF graph to come up with good URIs for every node, which would impose additional costs on creating the graph.

There are two different scenarios for using blank nodes. First, blank nodes are used in the structural representation of certain OWL axioms within RDF graphs. Second, blank nodes are used for anonymous ontology entities. An example for the first scenario is the representation of a disjoint union axiom in RDF. The axiom
`DisjointUnion(C D E)`
will be represented by the following RDF triples:

```
C owl:disjointUnionOf _:x .
_:x rdf:first D .
_:x rdf:rest _:y .
_:y rdf:first E .
_:y rdf:rest rdf:nil .
```

Blank nodes in RDF are represented by using the namespace prefix _ (underscore). In the given example, there are two blank nodes, `_:x` and `_:y`. They do not represent any entities in the domain, but are introduced only out of structural necessity since we cannot state the relationship between three entities (`C`, `D`, and `E`) from the original axiom directly with triples. We defined these kind of blank nodes to be *structural blank nodes*. Even though they could be given URIs, these URIs would not represent any concept in our conceptualization and thus should be avoided.

The second scenario uses blank nodes to represent anonymous ontology entities. For example, in the first few years it was regarded as good practice *not* to define a URI for persons in FOAF documents but to use blank nodes instead. The argument in favor of using blank nodes for persons was that it was regarded inappropriate to name people via URIs. This echoes the sentiment of *"I am not a number"*, or rather, *"I am not a URI"*. FOAF preferred to identify persons by using inverse functional properties such as eMail-adresses or their hash sums.

Web architecture later suggested that all entities of interest should have a URI (Jacobs and Walsh, 2004). The FOAF project also deprecated the use of blank nodes for persons (since they are definitively entities of interest). Using a URI for an entity allows for all the advantages described earlier about linked data (see Section 4.1.1), most importantly the possibility to look up further data about the given entity by resolving the URI.

In summary, blank nodes should be avoided unless structurally necessary.

**Method 10 (Check for superfluous blank nodes)**
Tables 2.1 and 2.2 list all cases of structurally necessary blank nodes in RDF graphs. Check for every blank node if it belongs to one of these cases. Besides those, no further blank nodes should appear in the RDF graph. All blank nodes not being structurally necessary should be listed as potential errors.

## Chapter 5

# Syntax



> Words differently arranged have a different meaning, and meanings differently arranged have a different effect.
>
> *(Blaise Pascal, 1623–1662, Pensées (Pascal, 1670))*

Web ontologies are serialized in a big (and growing) number of different surface syntaxes. Surface syntaxes can be classified into two different groups: the ones that describe a graph (which in turn describes the ontology), and the ones that describe the ontology directly (a graph can still be calculated from the ontology based on the transformation described in (Patel-Schneider and Motik, 2009)). Examples of the former group are RDF/XML (Bechhofer *et al.*, 2004) or NTriples (Grant and Beckett, 2004), examples of the latter are the Manchester Syntax (Horridge *et al.*, 2006), OWL Abstract Syntax (Patel-Schneider *et al.*, 2004), or the OWL XML Presentation Syntax (Hori *et al.*, 2003). All these syntaxes are transformable automatically from and into each other.

Each surface syntax warrants their own evaluation methods. Common features that can be evaluated over most of the syntaxes in a fairly uniform way are the proper and consistent indentation in the file and the order of the triples (for graph-based syntaxes) or axioms (for ontology-based syntaxes). Triples forming complex axioms should be grouped together. Groups of axioms forming an ontology pattern should also be grouped together. Terminological axioms should precede facts, and terminological axioms and facts about the same entity should be grouped together. Entities should be declared before usage (see Section 4.1.5). The file's encoding should be appropriate.

There should be a good reason for using anything but UTF-8 as the file encoding (Unicode Consortium, 2006).

Even though ontologies will rarely be edited in simple text editors, these guidelines will help tremendously once it does happen. Understanding a disjoint union axiom when the triples making up the axiom are scattered throughout the graph serialization creates an unnecessary challenge. We assume that the capability to debug the source file of an ontology increases dramatically when the guidelines given above are implemented. These guidelines are derived from the rules applied for software engineering big programs, where similar rules exist for the layout of the code in the source files (Kernighan and Plauger, 1978).

In this thesis we refrain from instantiating the given guidelines for any specific surface syntax. The rest of this chapter will first discuss two further specifics of the syntax which apply for most surface syntaxes (syntactic comments in Section 5.1 and qualified names in Section 5.2) and close with introducing a possibility of using XML validation for the completeness test of OWL knowledge bases (Section 5.3).

## 5.1 Syntactic comments

Many OWL surface syntaxes allow for comments. For example, an XML-based surface syntax may contain an XML-style comment such as `<!-- Created with Protege -->`. We call these syntactic comments and discern them from RDF-style comments, i.e. comments that are part of the RDF graph (see Section 4.2.3). Syntactic comments are often lost when two ontologies are merged, or when one ontology is transformed from one syntax to another. RDF-style comments on the other hand are stable with regards to transformation and merger.

For many syntaxes it is best practice to include some comments with metadata about the ontology document. XML documents, for example, often start with an introductory comment stating the version, author, and copyright of the given document.

In OWL document files this is not necessary, since all these informations can be expressed as part of the ontology itself. For example, the above quoted XML-style comment `<!-- Created with Protege -->` is injected by the Protégé editor (Noy *et al.*, 2000) in order to show that the ontology was created with that editor. Instead a triple could state that the ontology has been created using the Protégé editor.

```
ex:Ontology_A ex:authoringtool sw:Protege .
```

Most of the metadata in syntactic comments can be expressed using statements about the ontology. This allows ontology repositories to automatically and uniformly interpret the metadata about the ontologies in the repository, and to provide access and searches over the ontology using the very same sophisticated tools that are used to access the data within the ontology themselves.

## 5.2 Qualified names

Most serialization formats include mechanisms to abbreviate URI references. They are often based on known XML-based approaches, such as XML entities (Bray *et al.*, 2008) or XML namespaces (Bray *et al.*, 2006). Thus, instead of having to write the full URL `http://xmlns.com/foaf/0.1/knows` we can abbreviate names either using XML entities as `&foaf;knows` or using XML namespaces as `foaf:knows`.

When defining abbreviations and prefixes in an ontology document, care should be taken to bind well known abbreviations with the appropriate URIs. For example, the `foaf` prefix (Brickley and Miller, 2005) is so prevalent on the Web that it should always be defined as `http://xmlns.com/foaf/0.1/`. Some tools even assume that certain prefixes such as `rdf`, `rdfs`, or `owl` are defined as specified in the OWL standards (Smith *et al.*, 2004), and fail to load ontologies properly where this is not the case. Table 1.1 lists the prefixes that are used in this thesis.

*Swoogle* (Ding *et al.*, 2004) is a Semantic Web crawler and ontology repository that offers a number of statistics on the crawled ontologies. Swoogle can be used to offer default definitions for namespaces, so that we can check automatically if the prefix definitions follow common usage patterns on the Web.[1] Another site with namespace usage statistics is offered by *Ping the Semantic Web*, a service that helps with crawling and maintaining an ontology repository.[2] *prefix.cc*[3] is a socially constructed website, where users can add, edit, and vote on the prefixes and their resolution. Web service interfaces allow the automatic querying of *prefix.cc* for usage in a serialization or validation tool.

Note that using namespaces inconsistently with common usage will not lead to errors. When merging and combining ontologies, the underlying OWL libraries will handle the namespaces correctly, and do not care if we define `http://xmlns.com/foaf/0.1/` to be `foaf` or `dc`. It merely helps to avoid confusion for the human user not to break expectations regarding the definitions of namespaces, and to follow common practice on the Web.

## 5.3 XML validation

The Semantic Web was created as a set of standards building on top of each other, and incorporating already existing and well established standards such as URIs for identifiers (Berners-Lee *et al.*, 2005) or XML for serialization (Bray *et al.*, 2008). The inherent promise was that due to the reuse of these standards, widely deployed tools and expensively built expertise will not be lost but remain relevant and in continued

[1] `http://ebiquity.umbc.edu/blogger/2007/09/23/top-rdf-namespaces/`
[2] `http://pingthesemanticweb.com/stats/namespaces.php`
[3] `http://prefix.cc`

use. This promise has not been fully realized.

A characteristic of the standard RDF/XML syntax (Beckett, 2004) is that it can be used to let ontologies, particularly simple knowledge bases, mimic traditional XML documents and even be accompanied by an XML schema. But we will show in Section 5.3.2 that most XML oriented tools can only deal superficially with RDF/XML files. Due to numerous possibilities the RDF graph can be expressed in XML, creating applications using the XML set of tools and expertise is often inefficiently expensive and unreasonably hard. This is particularly true for typical XML evaluation approaches such as document validation. Basically, they are currently not applicable.

In this section an approach resolving this problem is presented. It uses well-established standards to sufficiently constrain the serialization of RDF/XML in order to be usable by classic XML processing approaches. We suggest to use the same approach humans would use in order to serialize their conceptualizations in XML, namely following an XML schema. Three popular XML schema languages are currently widely used, the XML-native Document Type Definitions (DTD) (Bray *et al.*, 2008), the W3C XML Schema Definition language (XSD) (Fallside and Walmsley, 2004), and the Regular Language for XML Next Generation (RELAX NG) (Clark and Murata, 2001). Due to its legibility and high availability we choose DTD for our prototype implementation, but the work can be extended to use the other schema languages as well, as will be discussed in Section 5.3.7. Note that by restricting RDF/XML with DTDs, the resulting files are still fully compliant RDF files, and therefore can be used by other Semantic Web tools out of the box.

**Method 11 (Validating against an XML schema)**
An ontology can be validated using a standard XML validator under specific circumstances. In order to apply this, the ontology needs to be serialized using a pre-defined XML schema. The semantic difference between the serialized ontology and the original ontology will help in discovering incompleteness of the data (by finding individuals that were in the original ontology but not in the serialized one). The peculiar advantage of this approach is that it can be used with well-known tools and expertise.

Depending on the schema, this method may severely impact the extensibility of the ontology document. The advantage of using an XML validating ontology document is that we can check if the document is data complete with regards to a previously made specification. This way tools can check before hand if the data is not only consistent with a given ontology, but also if there is sufficient data in order to perform a specific task.

The major theoretical challenges lie in the details of the interpretation of the XML schema file as a specification for generating the requested serialization. XML schemas can be roughly regarded as formal grammars meant to check a given input XML document for validity. Instead, we use the very same XML schema file in order to query a given RDF graph and then define how to serialize the results. This interpretation forces us to define a number of constraints on the covered XML schemas (i.e., arbitrary DTDs can not be used as described in Section 5.3.3). We describe a dialog-based tool to quickly create RDF/XML-compliant DTDs that contain the data that is requested by the developer, thus allowing the developer to easily adhere to these constraints without deep understanding of Semantic Web standards.

We provide a prototype implementation of the approach, and present an example instantiation of the method in order to demonstrate its usefulness and applicability. For our example we take FOAF files from the Web, pipe them through our implementation, and then use standard XML processing techniques (XSLT) in order to generate HTML pages. We provide the example workflow as a Web service.

Section 5.3.3 describes the approach to create the serializations. Section 5.3.4 proposes how the dialog-driven DTD creation tool works. This is followed by a description of the current prototype implementation and demonstration in Section 5.3.5. Section 5.3.6 describes related approaches towards the goal we are outlining, before we close with a discussion of further research questions. As a running example we describe a Web service that takes arbitrary FOAF-files (Brickley and Miller, 2005) and translates them to HTML and vCard presentation.

### 5.3.1 Example

An example of an XML-based serialization of RDF is given by RSS 0.9 (Beged-Dov *et al.*, 2000). RSS (either used as a shortcut for *RDF Site Summary* or *Really Simple Syndication*) is a format to enable the syndication of Web feeds of e.g. blogs, podcasts, video diaries, updates, etc.

RSS files are both valid RDF files and valid DTD-described XML files, where the DTD describes the document structure. Listing 5.1[4] is an abbreviated version of the DTD document for RSS 0.9 by Dan Libby (entities defining special characters and comments have been removed for brevity).

The DTD has the root element `rdf:RDF` (defined in the actual RSS document, see Listing 5.2). The DTD gives the grammar for the XML elements: the root `rdf:RDF` contains the elements `channel`, `image`, `item`, `textinput` in arbitrary order and number, but nothing else (note that the `*` around the brackets in line 1 makes the `?` and `+`

[4]Original source at `http://my.netscape.com/rdf/simple/0.9/` unavailable, copy according to `http://www.rssboard.org/rss-0.9.dtd`

Listing 5.1: Document type definition for RSS 0.9 documents.

```
<!ELEMENT rdf:RDF (channel | image? | item+ | textinput?)*>
<!ATTLIST rdf:RDF xmlns:rdf CDATA
     #FIXED "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns CDATA #REQUIRED>
<!ELEMENT channel (title | description | link)*>
<!ELEMENT image (title | url | link)*>
<!ELEMENT item (title | link)*>
<!ELEMENT textinput (title | description | name | link)*>
<!ELEMENT title (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT link (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT name (#PCDATA)>
```

inside the brackets meaningless). All these may contain a further number of elements, which are all flat `#PCDATA` fields.

An RSS document that is valid with regards to the given DTD could look like the example in Listing 5.2. A standard RDF tool will hardly serialize its RDF graph like this. The DTD defines the order of the elements, and it defines the names of the XML elements to be used. For an RDF library the order does not matter, and there are a number of different ways to serialize triples in XML. In order to serialize an RDF graph so that we can use it for validation against a schema, the serializer must be aware of the schema.

Listing 5.2: Example RSS document.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF PUBLIC
  "-//Netscape␣Communications//DTD␣RSS␣0.9//EN"
  "http://www.rssboard.org/rss-0.9.dtd">
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://my.netscape.com/rdf/simple/0.9/">
  <channel>
    <title>semanticweb.org</title>
    <link>http://semanticweb.org</link>
    <description>Semantic Web Community</description>
  </channel>
  <image>
```

```
    <title>SemanticWeb.org</title>
    <url>
 http://semanticweb.org/images/Semantic-Web-Logo-by-W3C.png
    </url>
    <link>http://semanticweb.org</link>
  </image>
  <item>
    <title>List of SemWeb tools</title>
    <link>http://semanticweb.org/wiki/Tools</link>
  </item>
  <item>
    <title>W3C puts OWL2 in last call</title>
    <link>http://semanticweb.org/wiki/News</link>
  </item>
</rdf:RDF>
```

Now we can check the RSS document against the DTD given above, and the XML validator will provide us with a list of all validation errors. If the file validates, we do not only know that it is a valid XML document and a valid RDF file, but that the entities described in the graph have certain properties filled (i.e. all `items` will have a `title` property).

Unlike the cardinality axiom `SubClassOf(Item ExactCardinality(1 title))` that allows us to infer that every `item` has to have a `title` property (be it given or not), validating against the above DTD will guarantee us that we *know* the actual value of the `title` property for every `item`. This allows us to state *data completeness*. This means that for certain properties we do not only know that they exist, but we also know that we know their values (this is similar to the usage of autoepistemic logic with OWL as described in Section 9.2.3).

### 5.3.2 Motivation

Today, one of the major hindrances towards achieving the wide deployment of Semantic Web technologies is the lack of sufficient expertise. If at all, Semantic Web technologies have only recently been introduced to students' curricula, and most general audience books on Semantic Web technology have only been published in the last two years (Pollock, 2009; Segaran *et al.*, 2009; Allemang and Hendler, 2008; Hitzler *et al.*, 2009). On the other hand, XML expertise is widely available. There are numerous books and courses on XML technologies, and many websites provides access to communities of practice, often even within the specific domain of the user.

Also, even though the number of Semantic Web tools is growing rapidly, many of them require a deeper understanding of the technology than is readily available.

Strengthening the ties between RDF and XML allows not only to reuse existing expertise, but also to re-enable the already existing tools.

To illustrate the problems with using RDF/XML, consider the following ontology, serialized in N3 (Berners-Lee, 2006):

```
aifb:Rudi_Studer rdf:type foaf:Person .
```

The following four documents are examples that all serialize this single triple in RDF/XML:

Listing 5.3: Expanded RDF/XML serialization

```
<rdf:RDF>
  <rdf:Description rdf:about="&aifb;Rudi_Studer">
    <rdf:type>
      <rdf:Description rdf:about="&foaf;Person"/>
    </rdf:type>
  </rdf:Description>
</rdf:RDF>
```

An object may be moved to the property element as an attribute value:

Listing 5.4: Object as attribute value

```
<rdf:RDF>
  <rdf:Description rdf:about="&aifb;Rudi_Studer">
    <rdf:type rdf:Resource="&foaf;Person"/>
  </rdf:Description>
</rdf:RDF>
```

Typing information can be moved to the element:

Listing 5.5: Typing by element name

```
<rdf:RDF>
  <foaf:Person rdf:about="&aifb;Rudi_Studer"/>
</rdf:RDF>
```

Finally, the encompassing `rdf:RDF` root element is optional if it only has one child node:

Listing 5.6: Removing optional root element

```
<foaf:Person rdf:about="&aifb;Rudi_Studer"/>
```

All documents have very different XML infosets and very different XML serializations, but equal RDF semantics. Thus, creating a simple list of all persons according to the RDF semantics is not trivial without using an RDF parser. An XQuery or an XSLT transformation will be cumbersome to write. In order to simplify the authoring of XML-based solutions, we provide a workflow that will normalize the RDF/XML serialization following a given XML schema.

The example in Listing 5.7 shows an XML DTD that normalizes the way to express FOAF data in XML files. It states that the root element has an arbitrary number of `foaf:Persons` (line 1) and each `foaf:Person` must have a `foaf:name` and may have some mail address (given by `foaf:mbox`, line 6). Lines 2-5 define the namespaces (Bray *et al.*, 2006) of the resulting document. Note that the namespaces are fixed. This is to circumvent the fact that DTDs per se are not namespace-aware (unlike other XML schema languages like XSD or RelaxNG). Line 7 ensures that every `foaf:Person` instance has a URI and may not be a blank node.

Listing 5.7: A DTD for a fragment of FOAF

```
 1 <!ELEMENT rdf:RDF (foaf:Person*)>
 2 <!ATTLIST rdf:RDF xmlns:rdf  CDATA #FIXED
 3       "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
 4 <!ATTLIST rdf:RDF xmlns:foaf CDATA #FIXED
 5       "http://xmlns.com/foaf/0.1/">
 6 <!ELEMENT foaf:Person (foaf:name foaf:mbox*)>
 7 <!ATTLIST foaf:Person rdf:about CDATA #REQUIRED>
 8 <!ELEMENT foaf:name (#PCDATA)>
 9 <!ELEMENT foaf:mbox EMPTY>
10 <!ATTLIST foaf:mbox rdf:resource CDATA #REQUIRED>
```

A FOAF-file that validates against the given DTD in Listing 5.7 is given in Listing 5.8. The result is a reasonably legible XML file and also a fully valid RDF file, which can be used with all Semantic Web tools.

Listing 5.8: FOAF-file after normalization

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <foaf:Person
      rdf:about="http://www.aifb.kit.edu/id/Rudi_Studer">
    <foaf:name>Rudi Studer</foaf:name>
    <foaf:mbox rdf:resource="mailto:studer@kit.edu" />
  </foaf:Person>
</rdf:RDF>
```

Even though the approach does not allow to create arbitrary XML – specifically, it does not allow to create XML documents that are not valid RDF – and thus cannot be used as a one-step solution towards reusing already existing DTDs, we will show that one can create a custom DTD to serialize the RDF file first, and then translate it to any representation the user requires (even in non-XML formats) by using readily available XSLT tools and expertise. Therefore it is possible to ground any RDF file into an already existing file format in two steps, given that all required information is available.

### 5.3.3 Normalizing the serialization

In order to create the normalized serialization we use the provided DTD to generate a series of SPARQL queries. The results of the queries are then used to write the actual serialization. In this section we describe how this is accomplished.

The given DTD has to fulfill a number of constraints that will be listed explicitly in the following section. First we need to read and define all given namespaces from the root element and declare them appropriately in the SPARQL query. Furthermore we add the RDF and RDFS namespaces, if not already given (for the namespaces used here, see Table 1.1).

Next we go through every possible child element type of the root element `rdf:RDF`. In our example we can see in line 1 that there is only one possible child element type, `foaf:Person`. This is the type of the sought for instances, i.e. we translate it into the following query fragment:

```
SELECT ?individual
WHERE {
  ?individual rdf:type foaf:Person .
}
```

Next, for every required child element of the found elements, we add a respective line to the `WHERE`-clause. In our example, `foaf:Person` requires only `foaf:name` (line 6, `foaf:mbox` is optional). So we add the following line (introducing a new variable every time):

```
  ?individual foaf:name ?v1 .
```

If the `foaf:name` itself would have pointed to another element, this element would be the type of `?v1`, and in return we would add all required properties for `?v1` by adding the subelements of this type, and so on.

The result of this query is a list of all individuals that are described with all the necessary properties as defined by the DTD. In the example they are, thus, not only

instances of `foaf:Person` but also of $\mathbf{K}$`foaf` : `name`.$\top$, i.e. we know that they have at least one `foaf:name` given (see Section 9.2.3).

Next we iterate over the result list, asking for every required and optional property individually. In the example, we would issue the following two queries:

```
SELECT ?result
WHERE {
  aifb:Rudi_Studer foaf:name ?result .
} LIMIT 1

SELECT ?result
WHERE {
  aifb:Rudi_Studer foaf:mbox ?result .
}
```

Since line 6 of the DTD states that there is only one `foaf:name`, we limit the first query to 1. Again, if the subelement of a property would have been stated in the DTD, it would have been required to add all required properties for `?result`, just as it was done for `?v1` above. Furthermore, each required and optional property of `?result` also has to be gathered in the same way as we do for all the instances of the root element's children.

With the results of the queries, we can start generating the serialization. Listing 5.8 shows such a serialization that was created with the normalization tool.

A further note on the approach: since the resulting queries are all describing conjunctive queries, we can use the queries on a reasoning-aware SPARQL endpoint. This allows us to employ the power of RDFS and OWL to provide for a fully transparent ontology mapping. For example, even if the individuals in the input file are actually having the class `swrc:Student`, as long as this is a subclass of `foaf:Person` the results will come back as expected and the normalized serialization will contain all instances of direct and indirect subclasses. This provides a powerful and easy way to quickly add further knowledge to existing XML-based tool chains.

### 5.3.4 Creation of compliant schemas

The provided DTDs have to fulfill a number of constraints so that they can be used by the normalization tool. This section lists those constraints explicitly. Since it would require quite some expertise with regards to RDF/XML-serializations (which we explicitly do not expect) to create DTDs fulfilling these constraints, we also propose a dialog-based tool to easily create such DTDs.

An RDF-compliant DTD, that can be used by the normalizer described in the previous section, has to fulfill the following constraints:

- the resulting XML-file must be a valid RDF/XML-file
- all used namespaces have to be fixed in the DTD and defined in the root element of the resulting XML-file
- the root element of the XML-file has to be `rdf:RDF`
- since DTDs are not context-sensitive, the DTD can use each element only a single time

Especially the last constraint is a severe restriction necessary due to the shortcomings of DTDs. In Section 5.3.7 we will take a look at possible remedies for this problem using other, more modern XML schema languages.

The first constraint is basically impossible to fulfill without deep knowledge of the RDF/XML serializations. Because of that we suggest a tool that analyses a given dataset or ontology and then guides the developer through understandable dialog options to create a conformant DTD. The tool follows the following approach:

1. the tool loads a number of RDF files. It does not matter if these files contain terminological ontologies, knowledge bases, or populated ontologies.
2. the tool offers the developer to select a class from the given RDF files. `rdfs:Class` and `owl:Class` are both considered. The developer has to decide if the result should contain exactly one instance, or an arbitrary number of instances.
3. for the selected class, all sensible properties are offered. Sensible properties are those that either are defined with a domain being the given class, or where the instances of the given class have assertions using the property. For each selected property the developer has to decide if this property is required, can be repeated arbitrary times, or both.
4. for each selected property the developer has to decide on the type of the filler, especially if it is a datatype value or an individual, and if the latter, if it is of a specific class (which again will be selected from a provided list, based both on the range of the property and the classes of the actual fillers in the knowledge base).
5. if a class was selected, enter recursively to Step 3.
6. as soon as a property is selected and fully described, the developer can select another property by repeating from Step 3 on.
7. as soon as a class is fully described, the developer can continue with another class by repeating from Step 2.

The tool has to be careful not to allow the selection of any element twice. This constraint is stronger than required, and we expect future work and more thorough analysis to relax it and thus extend the expressivity of the tool. This is especially true when moving to more powerful schema languages that are aware of the context an element is used in. We are also aware that the list of sensible properties may be incomplete. We discuss this in Section 5.3.7.

### 5.3.5 Implementation

In order to demonstrate the feasibility of our approach, we have developed a prototype Web service implementation. The input is an arbitrary RDF file. It should describe a person using the FOAF vocabulary. The Web service uses an extension of the DTD given in Listing 5.7. The full DTD can be accessed online from our demonstration site.[5]

The first step of the output is an RDF file describing one person, using a specified syntax (an incomplete example is given in Listing 5.8, for complete examples please refer to the demo site).

Now developers can post-process the output of the serializer with the same ease they would have with any XML files, and they do not need to have any further knowledge of Semantic Web technologies to do so. There are numerous technologies for the further processing of XML files. In our example implementation we use the XML transformation language XSLT (Clark, 1999). XSLT is capable of translating XML files either to other XML files, or to any other format as well. In our example, we provide an XSLT transformation to turn the resulting RDF file into a vCard (Dawson and Howes, 1998). vCard is a pre-XML IETF standard for exchanging contact data, which is in wide use. Listing 5.9 gives the full XSLT for this translation. The demo website also offers a translation into HTML. An XSLT file offering the same results over arbitrarily serialized RDF would have been much longer, and harder to write and maintain.

Listing 5.9: XSLT for transforming FOAF to vCard

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/">
<xsl:output method="text" indent="yes"
            media-type="text/x-vcard" />
<xsl:template match="foaf:Person">
BEGIN:VCARD
VERSION:3.0
```

[5] http://km.aifb.uni-karlsruhe.de/services/RDFSerializer/

```
UID:<xsl:value-of select="@rdf:about" />
N;CHARSET=UTF-8:<xsl:value-of select="foaf:family_name" />;
<xsl:value-of select="foaf:firstName" />;;;
FN;CHARSET=UTF-8:<xsl:value-of select="foaf:name" />
<xsl:for-each select="foaf:mbox">
EMAIL;TYPE=internet:<xsl:value-of select="@rdf:resource" />
</xsl:for-each>
URL:<xsl:value-of select="foaf:homepage/@rdf:resource" />
CLASS:PUBLIC
ORG;CHARSET=UTF-8:;
END:VCARD
</xsl:template>
</xsl:stylesheet>
```

The demonstration site uses Xalan[6] as the the XSLT processor. Applying XSLT transformations to the sample RDF file yields the the result given in Listing 5.10.

Listing 5.10: A vCard file created by transforming RDF/XML

```
BEGIN:VCARD
VERSION:3.0
UID:http://www.aifb.kit.edu/id/Rudi_Studer
N;CHARSET=UTF-8:Studer;Rudi;;;
FN;CHARSET=UTF-8:Rudi Studer
EMAIL;TYPE=internet:mailto:rudi.studer@kit.edu
URL:http://www.aifb.kit.edu/web/Rudi_Studer
CLASS:PUBLIC
ORG;CHARSET=UTF-8:;
END:VCARD
```

### 5.3.6 Related approaches

In this section we discuss alternative approaches towards bridging the gap between the Semantic and the Syntactic Web.

1. The main related approach is to combine or extend XSLT with capabilities to seamlessly deal with RDF, and still continue to provide the same output formatting power. There are a number of implementations towards this goal, such as

[6] http://xml.apache.org/xalan-j/

RDF Twig,[7] TreeHugger,[8] or RDFXSLT.[9] XSPARQL[10] provides similar capabilities, but by extending SPARQL. These approaches have all proven feasible, but do not resolve the original issue: they all require the developer to understand RDF.

2. A very early approach is DTDMmaker (Erdmann and Studer, 2001) which automatically creates DTDs based on the vocabulary defined in (F-logic) ontologies. It also reports about the same issues identified here (about the shortcomings of DTDs) but is not flexibly customizable and also does not create RDF-conformant DTDs.

3. Another approach is to write custom hard-coded translators, that first read the RDF graph and then create the serialization. This may be the easiest way, but it requires hard-coding a proprietary solution for each use case, and it requires the programmer to learn the API of at least one RDF parser. Our approach does not require any newly written, proprietary code.

4. One approach is to use SPARQL first, and then use the SPARQL XML result format as the basis for the XSLT translations. This approach is viable, but it requires the developer both to understand SPARQL and the SPARQL result format, and then create XSLT scripts to deal with the representation of their data in the SPARQL result format instead of an XML vocabulary that is close to the domain.

5. The serializers of RDF generating tools could be rewritten so that they always return the required serialization. But this would mean that only output of these tools can be used, and we lack interoperability with other tools using the same ontology, since the XML serialization would usually not be specified. If the serialization indeed is specified, then this would have been done using an XML schema, and therefore it just hard codes our proposed approach into the RDF source. This is the approach that RSS has chosen, as shown in Section 5.3.1.

6. Another approach is to change the tool using the data so that it becomes RDF aware, i.e. instead of translating the RDF graph to the required format we can enable the tool to use RDF. Even though we personally would prefer this approach, in our example use case this would require to extend all existing tools that can consume vCard to also accept RDF descriptions of persons. This renders this solution unlikely.

[7] http://rdftwig.sourceforge.net/
[8] http://rdfweb.org/people/damian/treehugger/index.html
[9] http://www.wsmo.org/TR/d24/d24.2/v0.1/20070412/rdfxslt.html
[10] http://xsparql.deri.org/spec/

We conclude that the major difference of our approach is in the zero-knowledge assumption in using it: no knowledge of Semantic Web technologies is required. Expertise in wide-spread XML technologies is sufficient to start using Semantic Web knowledge bases as data sources.

### 5.3.7 Open questions for XML schema-based RDF validation

We described and implemented an approach towards bridging the gap between classic XML-technologies and the novel Semantic Web technology stack. The current implementation is already usable, but it exhibits a number of limitations. We list these limitations here in order to explicitly name open challenges.

- The given approach can be redone using other, more powerful and modern XML schema languages. These languages add further features, e.g. cardinalities.
- DTDs do not allow for context sensitive grammars. Therefore elements can only appear once in every DTD, which severely constraints their expressive power. For example, it is not possible to ensure that every person in a FOAF-file requires a name and mailbox and may have friends (which are persons), and at the same time define that friends should not have a mailbox. Using a context-sensitive XML schema language can remedy this.
- Even without moving to more powerful schema languages, the given constraints in Section 5.3.4 can be relaxed. Further analysis is required to understand this bigger language fragment.
- For now the prototype implementation ignores that a property that appears several times should have several different values, i.e. is basically a cardinality declaration. This could be expanded.
- A number of features have not been explored for this first implementation, that future work will take into account, e.g. datatypes, language tags, the `collection` and `xmlliteral` parsetypes, and blank nodes.
- As we have seen, DTDs have to use fixed namespaces and prefixes, whereas XML namespace-aware schema languages could deal with them more elegantly.

For now we provide the current implementation and a Web-accessible demonstration workflow to show the advantages of the described approach. We expect that the approach will be applied in a number of early use cases in order to gain more insight in the actual usage and to direct the future development of the given project.

## Chapter 6

# Structure

Caught up in circles
confusion is nothing new

*(Cindy Lauper, b.1953,*
*Time after Time*
*(Lauper and Hyman, 1983))*



The most widely explored measures used on ontologies are structural measures. Graph measures can be applied on the complete or partial RDF graph describing the ontology. An example of an extensively investigated subgraph would be the one consisting only of edges with the name `rdfs:subClassOf` and the nodes connected by these edges (i.e. the explicit class hierarchy). This subgraph can be checked to see if the explicit class hierarchy is a tree, a set of trees, or if it has circularities, etc. If it is indeed a tree, the depth and breadth of the tree can be measured. Current literature defines more than forty different metrics (Gangemi *et al.*, 2005; Gangemi *et al.*, 2006b; Lozano-Tello and Gómez-Pérez, 2004; Tartir *et al.*, 2005) that measure the structure of the ontology. Structural measures have a number of advantages:

- they can be calculated effectively from the ontology graph. Graph metrics libraries are available and can be used for this task.
- they simply yield numbers. This makes tracking the evolution of the ontology easy, because even in case the meaning of the number itself is not well understood, its change often is.
- their results can be checked automatically against constraints. For example, constraining the maximal number of outgoing edges of the type *rdfs:subClassOf* from a single node to five can be checked on each commit of the ontology to a version control system. Upon violation, an appropriate message can be created.

- they can be simply visualized and reported.

Due to these advantages and their simple implementation, most ontology toolkits provide ready access to a number of these metrics. Also ontology repositories often provide annotations and filtering options of the ontologies in the repositories using these metrics.

But in practice, structural metrics are often not well-defined. That is, based on their definitions in literature, it is hard to implement them unambiguously. Also there is often confusion with regards to their meaning. Often they define a new graph structure (and do not use the existing RDF translation), but then fail to define a proper translation of the ontology features to the graph (e.g. how to translate that a property is transitive, how to translate domain and ranges, etc.). Section 6.1 examines four metrics taken from literature and provides exemplary analysis of the shortcomings of their definition, and also offers remedies for these definitional problems.

Besides measures counting structural features of the ontology, the structure can also be investigated with regards to certain patterns. The best known example is to regard cycles within the taxonomic structure of the ontology as an error (Gómez-Pérez, 2004). But also more subtle patterns (or anti-patterns) and heuristics can be used to discover structural errors: Disjointness axioms between classes that are distant in the taxonomic structure (Lam, 2007), as well as certain usages of the universal quantifier (Vrandečić, 2005). Section 6.2 applies the RDF query language SPARQL over the OWL graph in order to detect structural patterns and anti-patterns.

Section 6.3 shows how the meta-ontology defined in Section 3.2 can be used to detect constraint validations within the ontology structure. As an example, we will formalize the OntoClean constraints in OWL and show how OWL reasoners can be used to discover OntoClean constraint validations (Guarino and Welty, 2002).

## 6.1 Structural metrics in practice

In this thesis we focus on the foundational aspects that form the base for automatically acquirable measures. Therefore we will not define a long list of metrics and measures, but rather take a step back and discuss conditions that measures have to adhere to in order to be regarded as semantically aware ontology metrics. This also helps to understand clearly what it means for a metric to remain on a a structural level.

Thus the scope of this work compares best to other metric frameworks, such as the QOOD (quality oriented ontology description) framework (Gangemi *et al.*, 2006b) and the $O^2$ and *oQual* models (Gangemi *et al.*, 2006a). The authors created semiotic models for ontology evaluation and validation, and thus describe how measures should be built in order to actually assess quality. They also describe the relation between the ontology description, the ontology graph, and the conceptualization that is expressed

within the graph, and they define measures for the structural, functional, and usability dimension. A graphical overview of $O^2$ is given in Figure 11.1 on page 189.

Simple structural measures can uncover a number of interesting features such as the reasoning complexity of the given ontology. Complexity is actually a measure of an ontology language, and defines the complexity for reasoning over instances of that language. The complexity of the OWL languages are known, but that does not give us much information on particular ontologies. The expressivity of the used language merely defines an upper bound on the complexity that applies to the reasoning tasks. With a simple list of the constructs used within the ontology one can further refine the used language fragment, and thus arrive at a possibly lower complexity bound and thus a better estimation of the reasoning task. For example, OWL DL is known to correspond to the description logic $\mathcal{SHOIN}(\mathbf{D})$ (Horrocks and Patel-Schneider, 2004), and thus reasoning tasks such as satisfiability checks are known to be NExpTime-Complete (Schaerf, 1994).

Most OWL ontologies do not use the more expressive constructs (Wang *et al.*, 2006; d'Aquin *et al.*, 2007a). Ontology editors such as SWOOP (Kalyanpur *et al.*, 2006) show the language fragment that is actually being used in a given ontology. But if the ontology only uses a certain set of constructs, we know that we are, e.g., in a tractable fragment (as an example, the Semantic MediaWiki query language described in Chapter 10 corresponds to a tractable fragment of OWL). Furthermore, even if more expressive constructs are used, queries to the ontology can often be answered much more efficiently than the theoretical upper bound suggests. Experiments indicate that a priori estimates of *resoning speed* can be pursued based purely on structural features (Wang and Parsia, 2007).

**Method 12 (Ontology complexity)**
We define measures counting the appearance of each ontology language feature. We do this by first defining a filter function $O_{\mathcal{T}} : \mathcal{O} \rightarrow \mathcal{O}$ with $\mathcal{T}$ being an axiom or an expression type (as given in Chapter 2). $O_{\mathcal{T}}$ returns all the axioms of axiom type $\mathcal{T}$ or all axioms having an expression of type $\mathcal{T}$.

We can further define a counting metric $N_{\mathcal{T}} : \mathcal{O} \rightarrow \mathbb{N}$ as $N_{\mathcal{T}}(O) = |O_{\mathcal{T}}(O)|$. We also define $N(O) = |O|$.

Then we can further define a few shortcuts, derived from the respective letters defining DL languages (Baader *et al.*, 2003), for example:

- **Number of subsumptions** $N_{\sqsubseteq}(O) = N_{\texttt{SubClassOf}}(O) = |O_{\texttt{SubClassOf}}(O)|$: the number of subsumption axioms in the ontology
- **Number of transitivities** $N_{+}(O) = N_{\texttt{TransitiveProperty}}(O)$: the number of properties being described as transitive

- **Number of nominals** $N_{\mathcal{O}}(O) = N_{\texttt{OneOf}}(O)$: the number of axioms using a nominal expression
- **Number of unions** $N_{\sqcup}(O) = N_{\texttt{UnionOf}}(O)$: the number of axioms using a union class expression
- etc.

With these numbers we can use a look-up tool such as the description logics complexity navigator (Zolin, 2010). If $N_O > 0$, then the nominals feature hast to be selected, if $N_+ > 0$ we need to select role transitivity, etc. The navigator will then give us the complexity of the used language fragment (as far as known).

We further define $H(O) : \mathcal{O} \rightarrow \mathcal{O}$ as the function that returns only *simple subsumptions* in $O$, i.e. only those `SubClassOf` axioms that connect two simple class names.

The rest of this section discusses exemplary measures from the literature, shows how they are ambiguous, and offers actions to remedy these ambiguities (in favor of a purely structural application). In Chapter 8 these measures will be revisited in order to redefine them with the help of normalization, introduced in Chapter 7. Note that the following is criticizing only the actual description of the single measures, not the framework they are described in. The remedies are often simple, and after applying them the now remedied measures can be used with the original intention.

### 6.1.1 Maximum depth of the taxonomy

OntoMetric (Lozano-Tello and Gómez-Pérez, 2004) is a methodology for selecting an ontology, based on five dimensions which in turn are organized into selection factors. The *content* dimension is the one most related to the work presented in this thesis, organized in the four factors *concepts*, *relations*, *taxonomy*, and *axioms*. We will take a closer look at one of the metrics related to the factor *taxonomy*, the *maximum depth*.

The *maximum depth* of the concept hierarchy is *"defined as the largest existing path following the inheritance relationships leading through the taxonomy"*.[1]

This definition leads to a number of problems. First, cycles in the class hierarchy will lead to the result that the *maximum depth* is $\infty$, which may be a not too useful result. Furthermore, consider the following ontology:

[1] Translated from the original Spanish: *"La profundidad máxima en la jerarquía de conceptos: definida como el mayor camino existente siguiendo las relaciones de herencia que puede alcanzar la taxonomía."* (Lozano-Tello, 2002, p. 72)

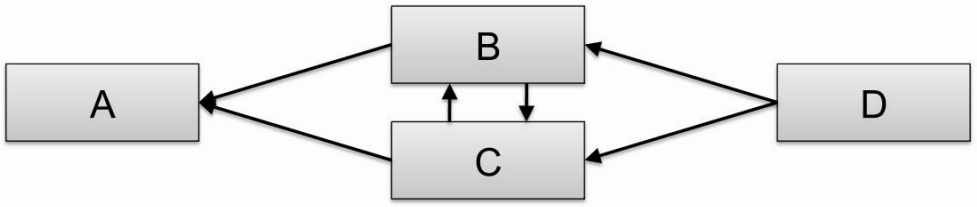


Figure 6.1: Example for a circular hierarchy path.

```
EquivalentClasses(A MinCardinality 1 R)
EquivalentClasses(B MinCardinality 2 R)
```

Since the definition only considers explicit inheritance relationships, it will miss that `B` is a subclass of `A`, and thus it will report a *maximum depth* of only 0 (no explicit inheritance relationships) instead of 1.

In Section 7.1 we will introduce the technique of normalization, which will allow us to use the maximum depth metric as it is intuitively meant to be used (see Section 8.2).

In order to define a purely structural *maximum depth*, it would be required to first rename the metric so that the meaning of the resulting number is clear. The appropriate name obviously depends on the actual definition of the metric, and one possible definition could be: the biggest number of subsumption axioms connecting consecutively simple class names without the same class name ever being repeated. An appropriate name for such a metric could be *maximum subsumption path length*.

Compared to the original definition, this definition can deal with cycles in the subsumption graph. Some issues still remain. Consider the following ontology, given in Figure 6.1:

```
SubClassOf(B A)
SubClassOf(B C)
SubClassOf(C B)
SubClassOf(D C)
SubClassOf(D B)
```

The longest path is four (either `ABCD` or `ACBD`), even though we would expect the maximal depth of the class hierarchy to be 3 (since `B` and `C` are on the same hierarchy level). But this is an inherent flaw of structural metrics. Only by renaming the metric from a name that implies that it measures a semantic feature (*maximum depth of the taxonomy*) to a name that states explicitly that it measures a structural feature (*maximum subsumption path length*) we can break the original intuition and thus achieve a metric that does not deceive due to its name.

### 6.1.2 Class / relation ratio

In (Gangemi *et al.*, 2005) the authors introduce 32 *"measures of the structural dimension"*. Note that these measures do not fully correspond to what we define to be structural measures within this thesis. E.g., (M24) (*Consistency ratio*) is defined as $\frac{n_{Cons}}{n_G}$ with $n_{Cons}$ being the number of consistent classes and $n_G$ the number of classes. In order to count the number of consistent classes we need a reasoner and thus this measures not a structural, but rather what we define a representational aspect of an ontology.

Measure (M29) in (Gangemi *et al.*, 2005) is called the *"Class / relation ratio"*, suggesting that it returns the ratio between classes and relations (or properties). The exact definition of the measure is: *"*$\frac{n_{G \in S}}{n_{R \in S}}$ *where* $n_{G \in S}$ *is the cardinality of the set of classes represent[ed] by nodes in g, and* $n_{R \in S}$ *is the cardinality of the set of relations represented by arcs in g"* (Gangemi *et al.*, 2005).

But applying the definition yields the ratio between the number of nodes representing classes and the number of nodes representing relations within the ontology graph, which will be a different number since a number of nodes, and thus names, can all denote the same class or relation. Therefore the metric is improperly named, since it does not yield the ratio between classes and relations, but rather between class names and relation names.

In Section 8.3 we will return to this metric and redefine it properly to capture the intuition behind the name. For now, in order to achieve a structural metric, it is again required to rename the metric, e.g. to *"class name / property name ratio"*. It is unclear if this metric is useful, but since it is far easier obtained than the actual *"class / relation ratio"* metric and will often correlate to it (compare Chapter 8), we expect it to remain displayed by ontology tools.

### 6.1.3 Relationship richness

OntoQA is an analysis method that includes a number of metrics (Tartir *et al.*, 2005), and thus it allows for the automatic measurement of ontologies. They define metrics such as richness, population, or cohesion. Whereas all these metrics are interesting, they fail to define if they are structurally or semantically defined – which is a common lapse.

As an example we choose the metric (RR) *"relationship richness"*. (RR) is defined as $RR = \frac{|P|}{|SC|+|P|}$, *"as the ratio of the number of relationships (P) defined in the schema, divided by the sum of the number of subclasses (SC) (which is the same as the number of inheritance relationships) plus the number of relationships"* (Tartir *et al.*, 2005). In our terminology $|P|$ is the number of property names, and $|SC|$ the number of subsumptions with the subclass being a class name.

According to the authors, this metric *"reflects the diversity of relations and place-*

*ment of relations in the ontology"* and is based on the assumption that an ontology that *"contains many relations other than class-subclass relations is richer than a taxonomy with only class-subclass relationships"*. We will look at a number of examples in order to evaluate this claim.

Consider the following top level ontology (the class hierarchy is from Proton (Terziev *et al.*, 2005), but the axioms are written differently to illustrate the argument. The Proton ontology uses indeed simple subsumptions):

```
DisjointUnion(proton:Entity
              proton:Abstract
              proton:Happening
              proton:Object)
EquivalentClasses(proton:Object
                  UnionOf(proton:Agent
                          proton:Location
                          proton:Product
                          proton:Service
                          proton:Statement))
```

Since we have not used any explicit subsumption axioms, $|SC|$ is 0. Since no properties have been introduced, $|P|$ is 0 as well, leading to $RR = \frac{0}{0+0}$, which is undefined. Now we can easily imagine adding classes to the ontology, all defined by class equivalences or disjoint unions – both $|SC|$ and $|R|$ will always be 0, and $RR$ will remain undefined. But now we add the following axiom from Proton:

```
PropertyRange(proton:isOwnedBy proton:Agent)
```

$RR$ will suddenly become $\frac{1}{0+1} = 1$, i.e. as rich as possible. Even adding further properties will not increase the relationship richness at all, it will constantly remain 1. Now consider the following axiom:

```
SubClassOf(proton:Agent proton:Object)
```

This axiom explicates one of the subsumptions relation stated indirectly in the class equivalence axiom of the ontology, i.e. it is redundant. But if we add this axiom to our original ontology instead of the range axiom, our $RR$ value would have dropped to 0, since $|SC|$ woulde be 1 and $|R|$ remain 0. And now we can add a mix of property axioms and such explicated subsumption axioms and basically get any $RR$ value that we want, by simply explicating some of the subsumption axioms.

Therefore we state that $RR$ is not a useful measure to capture relationship richness. We return in Section 8.4 to this measure and discuss how the measure can be repaired.

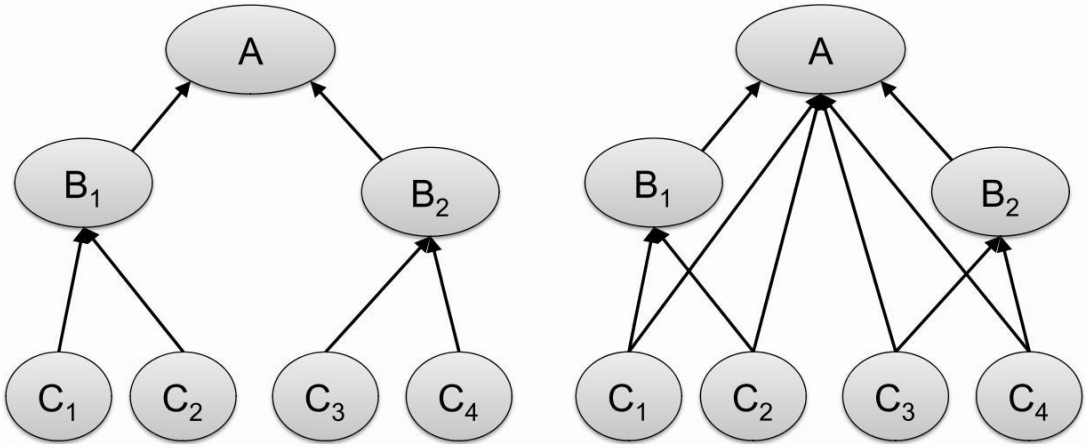


Figure 6.2: Two class hierarchies with identical semantics. Semantic similarity measure *ssm* of $C_1$ and $C_3$ is lower in the left ontology than in the right one.

### 6.1.4 Semantic similarity measure

Another example of a structural metric is given by (Alani and Brewster, 2006), where the authors describe metrics for ranking ontologies, such as the class match or the density measure. Interestingly even the so called semantic similarity measure *ssm* is not a semantic measure in the sense described here, since they apply all these measures on the graph that describes the ontology, not on the ontological model. *ssm* is defined between two classes and is the reciprocal of the length of the shortest path in the ontology graph from one class to the other (Alani and Brewster, 2006).

In Figure 6.2 we see two class hierarchies that represent the same semantics. The structure in the right hand side ontology contains a number of redundant, explicit subsumptions, i.e. the ontology graph contains a few redundant `rdfs:subClassOf`-arcs. Since the semantics have not changed, one would expect that the semantic similarity measure would remain constant in both ontologies, which is not the case.

In fact, one could always connect any class C with an explicit subsumption to the top class without changing the semantics, and thus have any two classes be connected in two links via the top class. Note that when introducing *ssm*, (Alani and Brewster, 2006) notes already that further studies are required in order to find whether the assumption that *ssm* describes the semantic similarity depends on certain properties of the ontology. As shown here, it does: in Section 8.5 we will describe these properties.

## 6.2 SPARQL for finding patterns

Patterns are crucial elements in all mature fields of engineering (Alexander *et al.*, 1977; Gamma *et al.*, 1995). In ontology engineering, research has focused on ontology design patterns to build ontologies using a set of predefined patterns (Gangemi, 2005; Svatek, 2004; Staab *et al.*, 2001). In this case, patterns are used in order to formalize common configurations of entities. But patterns can also emerge, i.e. they can appear in an ontology without the prior intention to use a specific pattern, especially in the case of networked ontologies.

One of the major tasks when working with patterns is to recognize them. In order to offer the best support, ontology engineering tools need to recognize patterns and offer appropriate user support for working with them. Otherwise patterns can easily become compromised by tools that are unaware of them. We introduce an approach for detecting patterns in ontologies using readily available SPARQL query engines (Prud'hommeaux and Seaborne, 2008).

We investigate several issues and problems that arise when using SPARQL to detect patterns in OWL ontologies. We discuss three approaches towards resolving these problems and report empirical results when applying one of them. In order to keep the examples simple we will concentrate on one of the best known ontology engineering patterns, the partition pattern. We expect many of the problems and solutions investigated here to also apply to more complex patterns.

The following section describes how DL patterns can be translated to RDF graphs in numerous different way. In Section 6.2.2 we discuss several approaches to remedy these problems, and empirically test the most naïve approach with surprisingly good results. We then apply the results on the notion of anti-patterns in Section 6.2.5.

### 6.2.1 Translating patterns from OWL to RDF

The partition pattern, as defined by (Gómez-Pérez *et al.*, 2003), is maybe the best known ontology engineering pattern. Partitions are also often used as a building block for more complex patterns, as for example in the *values as subclasses partitioning a quality* pattern (Rector, 2005). The pattern partitions a class $A$ into a number of disjoint subclasses $B_1 \dots B_n$. For simplicity, we assume $n = 2$ in the following examples.

```
SubClassOf(B1 A)
SubClassOf(B2 A)
SubClassOf(IntersectionOf(B1 B2) owl:Nothing)
```

The partition pattern is displayed in Figure 6.3, using the same graphical syntax as in the Semantic Web best practice notes (Rector, 2005). An example for the application of this pattern is the partition of *animal* into *biped* and *quadruped* (this also

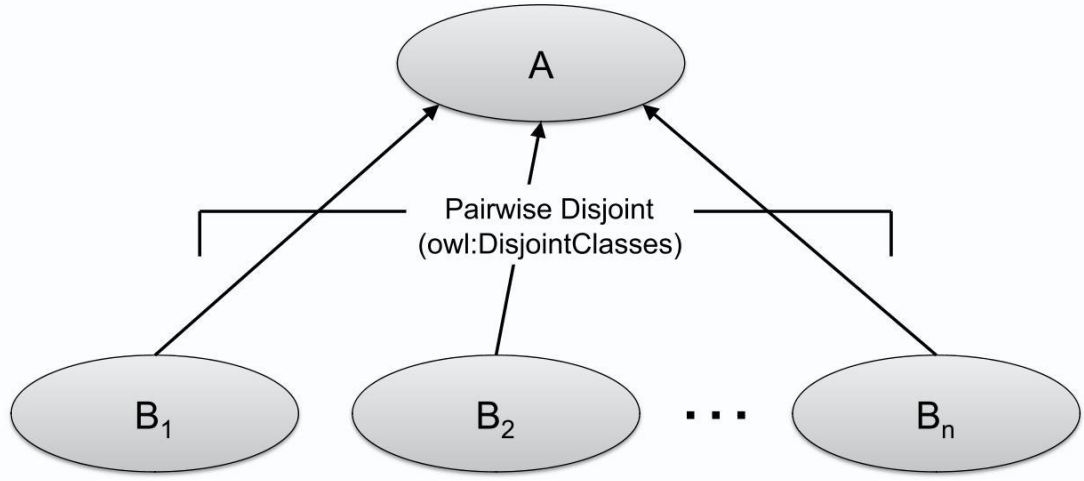


Figure 6.3: The partition pattern: class $A$ is partitioned into the subclasses $B_1 \dots B_n$

exemplifies the difference to a *complete partition*, as defined by (Gómez-Pérez *et al.*, 2003), that would make the far stronger claim that every individual of the class $A$ needs to be an individual of exactly one of the subclasses $B_1 \dots B_n$ by adding the axiom `SubClassOf(A UnionOf(B1 B2 ... Bn))`. The `DisjointUnion` axiom type was introduced in OWL2 and can state a complete partition directly (see Section 2.2.2). This is not true for the simple partition, that is taken as an example in this section.

Since SPARQL is a query language for RDF we need to consider how this pattern is represented in RDF. This means that the pattern needs to be translated from OWL to RDF (as shown in Table 2.1 and 2.2).

Unfortunately, this does not necessarily lead to a unique representation in RDF. In order to make the relation to SPARQL more obvious, we use the Notation3 syntax for RDF (Berners-Lee, 2006) instead of the more verbose standard RDF/XML-serialization (Klyne and Carroll, 2004).

```
B1 rdfs:subClassOf A .
B2 rdfs:subClassOf A .
_:a rdf:type owl:Class .
_:a owl:intersectionOf _:b .
_:b rdf:first B1 .
_:b rdf:rest _:c .
_:c rdf:first B2 .
_:c rdf:rest rdf:nil .
_:a rdfs:subClassOf owl:Nothing .
```

Another, semantically equivalent possibility to describe the pattern in OWL is by using the constructor for disjoint classes directly:

```
SubClassOf(B1 A)
SubClassOf(B2 A)
DisjointClasses(B1 B2)
```

This representation has the advantage of being easily extensible in case $n > 2$, since we do not need to add numerous `SubClassOf`-Axioms stating incoherent unions like in the first variant. As an RDF graph the second variant could look like this:

```
B1 rdfs:subClassOf A .
B2 rdfs:subClassOf A .
B1 owl:disjointWith B2 .
```

Note that the `DisjointClasses`-Axiom leads to at least $\frac{n^2-n}{2}$ triples (since it needs to be stated for all possible pairs of $B$s, and thus will be less terse for $n > 2$).

Another possible instantiation of partition would be given by the following semantically equivalent version of the partition pattern:

```
SubClassOf(B1 A)
SubClassOf(B2 intersectionOf(A complementOf(B1)))
```

This in turn can be translated to the following RDF graph:

```
B1 rdfs:subClassOf A .
B2 rdfs:subClassOf _:a .
_:a rdf:type owl:Class .
_:a owl:intersectionOf _:b .
_:b rdf:first A .
_:b rdf:rest _:c .
_:c rdf:first _:d .
_:c rdf:rest rdf:nil .
_:d rdf:type owl:Class .
_:d owl:complementOf B1 .
```

We assume that the partition pattern will be usually represented by one of these three patterns, or patterns similar to these. Even though many more are conceivable, they would be more complicated and thus less likely.

### 6.2.2 Querying with SPARQL

The RDF graphs can be translated straight-forward to the following SPARQL query.

```
select distinct ?A ?B1 ?B2 where {
  ?B1 rdfs:subClassOf ?A .
  ?B2 rdfs:subClassOf ?A .
  ?B1 owl:disjointWith ?B2 .
}
```

With an OWL DL aware SPARQL implementation this query should return the same result on all the variants above. But none of the implementations we are aware of gives this result (this result is unsurprising, since SPARQL is not a query language that fully addresses querying OWL DL ontologies, and the relation between the two is underspecified in the SPARQL standard (Prud'hommeaux and Seaborne, 2008). Most often SPARQL engines allow to query for ABox results, i.e. so-called conjunctive queries, but forbid TBox queries such as the one given above).

Implementations that are not aware of OWL DL can not be used to discover all instances of the pattern with the given SPARQL queries. In this case all possible patterns would be needed to be queried to gain a complete result, which is not practical.

In order to be able to search for knowledge engineering patterns using SPARQL, we thus can follow three possible approaches:

1. Implementing and using an OWL DL aware SPARQL engine
2. Materializing or normalizing the relevant parts of the ontology and then use an OWL DL unaware SPARQL engine
3. Committing to an incomplete search by searching only specific variants of the pattern's implementations

We consider the first option to be out of scope for this thesis, and thus we will discuss the other two options in the following two sections.

### 6.2.3 Ontology normalization

In Section 7.1 we define ontology normalization as an approach to make features of the semantics of an ontology explicit within the ontology's structure. By using an OWL DL unaware SPARQL engine, SPARQL is useful only for querying the structure of the ontology, i.e. its explicit RDF graph.

In order to use the query successfully with a SPARQL engine that is not aware of the OWL DL semantics, we first have to use an OWL DL reasoner to materialize all relevant parts of the ontology. In the case of the partition, we would need to materialize

the `subClassOf`- and `disjointWith`-constructs. But only direct `subClassOf`-relations should be materialized. Stating disjointness between classes that are not siblings is not considered a partition. If we materialize the complete `subClassOf`-graph, we could not recognize direct siblings anymore. Therefore we need to use the first three normalization steps introduced in Section 7.1 in order to avoid this problem.

Furthermore, the `disjointWith`-graph should not be fully materialized either, since the same problem would emerge. Subclasses of a class $A$ that was declared disjoint to a class $B$ would all be disjoint to $B$ as well. Such inherited disjointness should not be materialized.

Note that the normalization steps that would be required for recognizing the partition pattern are valid *only* for the partition pattern. If we wanted to recognize other patterns, we would have to consider anew which normalization steps would be required, and which could be left out.



### 6.2.4 Incomplete searches

In this section we explore empirically what happens if we drop the requirement of guaranteeing the discovery of a knowledge pattern. We took the SPARQL partition query from Section 6.2.2 and ran it against our corpus (see Section 11.3). We furthermore performed several other tests in order to understand how complete the query results are.

Of the 1331 ontologies of the Watson OWL corpus (see Section 11.3), only 55 where using disjointness axioms at all (4.1%). This is roughly consistent with previous surveys on the usage of expressive axioms in OWL ontologies on the Web: (Wang *et al.*, 2006) reported that 7.6% of the surveyed ontologies were using the `disjointWith`-construct, and (d'Aquin *et al.*, 2007a) reported 4.5% of OWL DL ontologies were using the $\mathcal{ALC}$ language fragment or stronger. The differences can be easily explained due to the different approaches towards collecting and filtering the ontology corpora.

Using the SPARQL query to detect partition patterns, we discovered the partition pattern in 47 of these 55 ontologies (85.5%). This means that most of the time when the disjointness axiom is used, the partition pattern is also applied. This fact was recognized by the DAML+OIL ontology language (Horrocks *et al.*, 2001) that included some more powerful construct to present partitions and complete partitions, such as `disjointUnion` (which has been introduced in OWL2 (Grau *et al.*, 2008) again).

We have taken a look at the 8 ontologies that did not show the partition pattern but still used disjointness (B90, B98, C37, E71, G13, H71, L22, and L66). E71, G13, and L22 turned out to be versions of the time ontology that implement an anti-pattern as discussed in Section 6.2.5. We will argue there that these ontologies should instantiate the partition pattern, but fail to do so. B90, B98, and L66 are obviously test ontologies and not meant to be used. C37 uses disjointness to partition the whole domain. Finally, H71 contains a partition that is unrecognized by the SPARQL

pattern, because the partitioned class is not subclassed directly but rather through an `IntersectionOf`-construct.

In order to see if any of the other two variants (or even similar implementations) given in Section 6.2.1 were used, we issued further queries. In order to detect variant 1 and similar patterns, we used the essential part of the variant 1 pattern:

```
select ?a where {
  ?a rdfs:subClassOf owl:Nothing .
}
```

This query did not return any results. Whereas in DL subclassing $\bot$, i.e. the empty class, is often used to state constraints, in OWL this possibility goes practically unused. Since OWL includes numerous powerful constructs to state such constraints directly, the rather counterintuitive usage of the empty class is usually avoided. Based on this result we can conclude that variant 1 and any similar patterns were not used to create partitions.

Based on these empirical results we may claim that any appearance of `owl:Nothing` in an OWL ontology is an anti-pattern (see Section 6.2.5) and thus indicates problems with the formalization. Furthermore, also any appearance of `owl:disjointWith` outside of a partition is a strong indicator for either a problem in the ontology, or an incomplete formalization. Further anti-patterns should be gathered from the Web.

In order to detect instantiations of variant 3, we search for any subclasses that are used to define a complement:

```
select ?a ?b where {
  ?c owl:complementOf ?b .
  ?b rdfs:subClassOf ?a .
}
```

This query returned 12 ontologies (A23, B14, C35, C53, F96, I27, J57, K13, L03, L77, M31, and N11). We examined them manually and found that in all but two cases the complement was used to state a restriction on the values of a property (e.g. $Vegetarian \equiv \forall eats.\neg Meat$). In the remaining two cases, the complement appeared once in a complex class description[2] and once to partition the whole domain.[3]

Based on these empirical results we see that the query indeed has detected all but three (C37, H71 and N11) instances of the partition pattern (thus yielding a recall of 94%). So even though we have discussed numerous problems and drawbacks of

[2] F96: $ViewableFile \equiv File \sqcap \neg(MediaFile \sqcap \neg ImageFile)$

[3] N11: $IntangibleEntity \equiv \neg TangibleEntity$

using SPARQL queries for detecting ontology patterns, experimental results are very promising.

Future work needs to investigate further patterns in order to see if these positive results will be confirmed for other, more complicated patterns. We assume though that current ontology engineering practice is lagging well behind the state of the art in ontology pattern research. Therefore possibilities for large-scale experiments will be rather limited.

### 6.2.5 Querying for anti-patterns

To detect so called anti-patterns is at least as important as detecting patterns in ontologies. Anti-patterns are strong indicators for problems in an ontology. The notion of anti-patterns was introduced to software engineering by (Koenig, 1995). He defines an anti-pattern to be similar to a pattern, *"except that instead of a solution it gives something thats looks superficially like a solution, but isn't one."*

Staying close to our example of the partition pattern we introduce the similar anti-pattern of a *skewed partition*. In a skewed partiton it is not the sibling classes that are declared to be disjoint, but rather classes in an *uncle*-relation, i.e. a class is disjoint to the sibling of its superclass (Lam, 2007). The following SPARQL query detects such an anti-pattern.

```
select distinct ?A ?B1 ?B2 ?C1 where {
  ?B1 rdfs:subClassOf ?A .
  ?B2 rdfs:subClassOf ?A .
  ?C1 rdfs:subClassOf ?B1 .
  ?C1 owl:disjointWith ?B2 .
}
```

The pattern was detected in five ontologies (E71, F54, G13, L22, and N25). Upon manual inspection they all turned out to be different versions of the time ontology (Hobbs and Pan, 2004). Figure 6.4 illustrates the relevant part of the ontology.

In the time ontology, a `TemporalEntity` has a start and an end time. There are two kind of `TemporalEntity`: `Instant` and `Interval`. An `Instant` is defined as a `TemporalEntity` that starts and ends at the same time, whereas a `ProperInterval` is defined as an `Interval` that does have a start time that is different from the end time.[4] Now what is the meaning of the class `Interval`?

`Interval` was introduced to capture also degenerate intervals that start and end at the same time. But such degenerate intervals are equivalent to `Instant`, and thus

[4]The discussed semantics of the time ontology are not fully captured in the OWL version of the time ontology, but are formalized as described in their first order logic description (Hobbs and Pan, 2004)

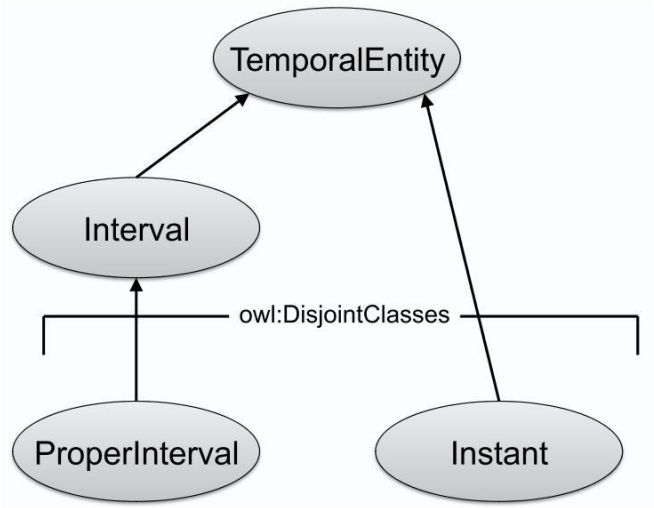


Figure 6.4: The upper levels of the time ontology. Note that `ProperInterval` and `Instant` are declared disjoint even though they are not sibling classes.

`Interval` is equivalent to `TemporalEntity`. The upper level of the time ontology could thus be simplified by removing the `Interval` class and creating a partition of `TemporalEntity` into `ProperInterval` and `Instant` (actually, a complete partition).

Cleaning up the ontology in such a way increases the understandability, and avoids confusion (users would rightly assume a difference between interval and temporal entitiy, why else would there be two distinct classes?). Changing to a complete partition as discussed above, and removing a class makes the ontology smaller, and thus easier to understand, and brings the invoked conceptual model when studying the ontology closer to its formal semantics.

**Method 13 (Searching for Anti-Patterns)**
SPARQL queries over the ontology graph can be used to discover potentially problematic patterns. For example results to the following queries have been found to be almost always problems.

Detecting the anti-pattern of subsuming nothing:

```
select ?a where {
  ?a rdfs:subClassOf owl:Nothing .
}
```

Detecting the anti-pattern of skewed partitions:

```
select distinct ?A ?B1 ?B2 ?C1 where {
  ?B1 rdfs:subClassOf ?A .
  ?B2 rdfs:subClassOf ?A .
  ?C1 rdfs:subClassOf ?B1 .
  ?C1 owl:disjointWith ?B2 .
}
```

A bigger library of such anti-patterns would help to flag areas in ontologies that warrant further investigations. Since such a library can be checked automatically, we can include it easily in an ontology build system.



## 6.3 The AEON approach

OntoClean (Guarino and Welty, 2000; Guarino and Welty, 2002) is a well-known methodology for ontology evaluation. More precisely, OntoClean enables the formal analysis of classes and their subsumption hierarchy based on the philosophical notions *rigidity*, *unity*, *dependency* and *identity* (known as the OntoClean meta-properties). By tagging the classes with the OntoClean meta-properties, the ontology engineer can capture more of what the ontology means in a concise and formal way. For example, by stating that a certain class is rigid or not helps to understand if this class is meant to be used as a role that applies to an individual, or rather as an essential type. By raising these questions and the ontology engineer answering them, the ontology will be more specific and easier to be used consistently.

From a practical perspective OntoClean provides means to derive measurable mismatches of a taxonomy with respect to an ideal structure which takes into account the semantics of subsumption. Such mismatches have a structural nature, e.g. one is able to derive that a certain concept should not be the subconcept of another concept. OntoClean provides an explanation of why mismatches occur which subsequently might help to improve the taxonomic structure. The philosophical notions of OntoClean may be the subjects of long discussions, however, strictly speaking, this is not part of the evaluation but of the ontology engineering because deciding the proper nature of a class forces the ontology to commit itself to a more specified meaning, which in turn allows for a more objective evaluation technique.

The application of OntoClean consists of two main steps. First, all classes are tagged with regards to the OntoClean meta-properties. Second, the tagged classes are checked against predefined constraints, with constraint violations indicating potential misconceptualizations in the subsumption hierarchy. Although OntoClean is well documented in numerous publications (Guarino and Welty, 2002; Guarino and Welty, 2004;

Guarino and Welty, 2000), and its importance is widely acknowledged, it is still used rather infrequently due to the high costs for application. Several tools supporting the OntoClean methodology have been developed and integrated into ontology editors such as ODEClean for WebODE (Fernández-López and Gómez-Pérez, 2002), OntoEdit (Sure *et al.*, 2003) or Protégé (Grosso *et al.*, 1999).

In order to leverage the adoption of OntoClean, we have developed AEON, an approach to automatize both steps of OntoClean. By means of AEON, we can automatically tag any given ontology with respect to the OntoClean meta-properties and perform the constraint checking. For creating the taggings, our implementation of AEON[5] makes extensive use of the World Wide Web as the currently biggest existing source of common sense knowledge. In line with several approaches such as (Cimiano *et al.*, 2005) and (Etzioni *et al.*, 2004) we defined a set of domain independent patterns which can be considered as indicators *for* or *against* Rigidity, Unity, Dependence and Identity of given concepts in an ontology.

To evaluate our automatic tagging approach we created a gold standard, i.e. we created a manually tagged middle-sized real-world ontology, and compared AEON results against it. A number of OntoClean experts as well as ontology engineering experts were involved in the creation of the more than 2,000 taggings in the gold standard. Each expert had to tag the PROTON ontology (Terziev *et al.*, 2005) with OntoClean meta-properties. Even though from a philosophical perspective one may argue that there can be only one OntoClean tagging for a given ontology our experiments had the interesting and important finding, that the experts agreed only to a certain extend on how to tag each individual concept. This shows again the difficulty of applying OntoClean in real-world settings. We see it as an advantage of our approach that it is based on the text corpus of the whole Web, instead of being defined by a small group or a single person. As key result of our evaluation our approach compares favorably with respect to the quality of the automatic taggings while reducing significantly the time needed to do the tagging.

In order to check the OntoClean constraints automatically, we decided to reuse an existing OWL DL formalization of the OntoClean meta-properties and constraints (OntoClean ontology). We used the meta-ontology given in Section 3.2 to represent the tagged ontology and were then able to automatically check the tagged ontology according to the OntoClean ontology. We expected two types of errors when analyzing the inconsistencies. First, the tagging of a concept is incorrect, and second, the corresponding taxonomic relationship is incorrect. We found both kinds of errors in our experimental data and looked at some of the errors in more detail to understand the rationale behind.

In the next section, we briefly introduce the idea behind OntoClean. Then we describe the four meta-properties and the most important OntoClean constraints. The

[5] http://ontoware.org/projects/aeon/

automatic tagging has been mostly developed by Johanna Völker and is described in (Völker *et al.*, 2008) in full detail, providing also the experimental validation of the automatic tagging approach. In Section 6.3.3, we describe AEON's support for the second step of OntoClean, i.e. the checking of constraints based on the meta-property taggings. For this purpose we reused an existing OWL DL version of the constraints and reified the tagged ontology. We also present some examples for the kind of inconsistencies we found in the tagged ontology.

### 6.3.1 OntoClean in theory

We provide a brief introduction to OntoClean, for a more thorough description refer for example to (Guarino and Welty, 2000). In the OntoClean vocabulary, *properties* are what we call *classes* in this thesis. *Meta-properties* are therefore properties of properties. Within this section we will use the term *meta-property* in the usual OntoClean sense, whereas we will refrain from using the term *property* but rather consistently use the term *class* as introduced in Chapter 2.

Applying the OntoClean methodology consists of two main steps.

- First, every single class of the ontology to be evaluated is tagged with occurrences of the core meta-properties, which are described in Section 6.3.2. Thus, every class has a certain tagging such as `+R+U-D+I`, where for example `+R` denotes that a concept carries *Rigidity* and `+U` denotes that the concept carries *Unity*. We call an ontology with tagged classes a tagged ontology (with regards to OntoClean, to be precise).

- Second, after the tagging, all subsumptions of the ontology are checked according to the OntoClean constraints (described in Section 6.3.3). Any violation of a constraint indicates a potential misconceptualization in the subsumption hierarchy.

The key idea of OntoClean is to constrain the possible taxonomic relations by disallowing subsumptions between specific combinations of tagged classes. This way, OntoClean provides a unique approach by formally analyzing the classes intensional content and their subsumption relationships. In other words, applying OntoClean means comparing the taxonomical part of a tagged ontology versus a predefined ideal taxonomic structure which is defined by the combination of meta-properties and constraints.

After performing the two steps the result is a tagged ontology and a (potentially empty) list of misconceptualizations. According to this list an ontology engineer may repair (in an OntoClean sense) the ontology, in order to remove all discovered misconceptualizations.

### 6.3.2 OntoClean meta-properties

As already indicated, the main ingredients of OntoClean are four meta-properties and a number of rules. The four meta-properties are: *rigidity (R)*, *unity (U)*, *dependence (D)* and *identity (I)*. They base on philosophical notions as developed by (Strawson, 1976) and others, even dating back to Aristotle (Aristotle, 330 BC). Here we will offer a short description of these meta-properties.

- **Rigidity.** Rigidity is based on the notion of *essence*. A class is essential for an instance *iff* it is necessarily an instance of this class, in all worlds and at all times. *Iff* a class is essential to all of its instances, the class is called rigid and is tagged with `+R`. *Iff* it is not essential to some instances, it is called non-rigid, tagged with `-R`. An anti-rigid class is one that is not essential to any of its instances. It is tagged ∼`R`. An example of an anti-rigid class would be `teacher`, as no teacher has always been, nor is necessarily, a teacher, whereas `human` is a rigid class because all humans are necessarily humans and neither became nor can stop being a human at some time.

- **Unity.** Unity tells us what is part of the object, what is not, and under what conditions the object is *whole* (Guarino and Welty, 2004). This answer is given by an *unity criterion* (UC), which describes the conditions that must hold among the parts of a certain entity to consider that entity as a whole. For example, there is an unity criterion for the parts of a human body, as we can say for every human body which parts belong to it. Classes carrying an UC have Unity and are tagged `+U` else `-U`.

- **Dependence.** A class $C_1$ is dependent on a class $C_2$ (and thus tagged `+D`), *iff* for every instance of $C_1$ an instance of $C_2$ must exist. An example for a dependent class would be `food`, as instances of `food` can only exist if there is something for which these instances are food. This does not mean that an entity being food ceases to exist the moment all animals die out that regarded it as food, it just stops being food. Another way to regard dependency is to distinguish between *intrinsic* and *extrinsic* class. Intrinsic classes are independent, whereas extrinsic classes need to be given to an instance by circumstances or definitions.

- **Identity.** A class with identity is one, where all instances can be identified as themselves, by virtue of this class or a superclass. This means that the class carries an *identity criterion* (IC). It is tagged with `+I`, and with `-I` otherwise. It is not important to answer the question of what this IC is (this may be hard to answer), it is sufficient to know that the class carries an IC. For example, the class `human` carries an IC, as we are able to identify someone as being the same or not, even though we may not be able to say what IC we actually used for

that. On the other hand, a class such as `red` would be tagged `-I`, as we cannot tell instances of red apart because of their color.

OntoClean differentiates between the two tags `I` and `O`, whereby the first means, that the concept simply carries an Identity Criterion (also through inheritance) and the second that it carries its *own* Identity Criterion. The difference is not relevant for this thesis, as the tagging `+O` may just be treated like the tagging `+I`, as `+O` implies `+I` anyway and there are no subsumption constraints about the tag `O`.

### 6.3.3 OntoClean constraints

A number of OntoClean rules is applied on the tagged ontology. We use the existing OntoClean rules to check a tagged ontology for consistency. Here, we will give some illustrative example for these rules. For a full list refer to (Guarino and Welty, 2004). As shown in (Sure *et al.*, 2003) such rules can be formalized as logical axioms and validated automatically by an inference engine. We will do so in the following section.

- **$\sim R$ can't subsume $+R$.** Having a class $C_1$ subsuming the class $C_2$, with $C_1$ tagged $\sim R$ and $C_2$ tagged $+R$, would lead to the following inconsistency: $C_2$ must always hold true for all of its instances (this is the meaning of the tagging: $C_2$ is a rigid concept). $C_2$, as a subsumed concept, would always imply $C_1$ for all of its instances. Therefore there are at least some instances of $C_1$ that are necessarily $C_1$ as they are $C_2$. Thus $C_1$ can not be anti-rigid, as the tagging says, because this would mean that it is not necessarily true for any of its instances – which would be a contradiction. *Example:* `food`, an anti-rigid class, subsuming `apple`, a rigid class. As it is explained in (Guarino and Welty, 2004), nothing is essentially food – it may or may not be eaten. On the other hand, an apple is always an apple. But if apples were subsumed by `food`, there would be some food that would essentially be food, namely apples, since every apple would always be an apple and thus food – which would be a contradiction to the statement that no food is essentially food.

- **$+I$ can't subsume $-I$.** If this rule was broken, it would mean that instances of the subsumed class can not be identified – although they are also instances of the subsuming class, which explicitly allows for the identification of the instances. This would be a contradiction, revealing an error in our taxonomy (or tagging).

- **$+D$ can't subsume $-D$.** `food` is an example for a dependent class. Modeling the class `candy`, we decide that everything with more than 20% sugar is candy, thus the class would be independent. We let `food` subsume `candy`, and the formal analysis shows this rule is broken. This points us to either an error in the taxonomy or in the tagging. In this example we see that the quality of the taxonomical analysis is only as good as the quality of the applied tagging.

### 6.3.4 Constraint checking

Equipped with the OntoClean taggings we are able to check the hierarchical part of the ontology with regards to the meta-property constraints defined by OntoClean. In order to check these constraints automatically, we use the meta-ontology described in Section 3.2 and extend it in order to provide a formalization of the constraints in OWL in order to check the reified ontology.

Listing 6.1: OntoClean constraints meta-ontology in OWL.

```
(1)  TransitiveProperty(meta:subClassOf)
(2)  DisjointUnion(meta:Class
                   oc:RigidClass oc:NonRigidClass)
(3)  DisjointUnion(meta:Class
                   oc:UnityClass oc:NonUnityClass)
(4)  DisjointUnion(meta:Class
                   oc:DependentClass oc:NonDependentClass)
(5)  DisjointUnion(meta:Class
                   oc:SortalClass oc:NonSortalClass)
(6)  SubClassOf(oc:AntiRigidClass oc:NonRigidClass)
(7)  SubClassOf(oc:AntiUnityClass oc:NonUnityClass)
(8)  EquivalentClasses(oc:RigidClass
           AllValuesFrom(meta:subClassOf
                         ComplementOf(oc:AntiRigidClass)))
(9)  SubClassof(oc:UnityClass
           AllValuesFrom(meta:subClassOf
                         ComplementOf(oc:AntiUnityClass)))
(10) SubClassOf(oc:DependentClass
           AllValuesFrom(inverseOf(meta:subClassOf)
                         DependentClass))
(11) SubClassOf(oc:SortalClass
           AllValuesFrom(inverseOf(meta:subClassOf)
                         SortalClass))
```

The formalization given in Listing 6.1 is based on the OntoClean formalization in OWL DL as described in (Welty, 2006). The ontology builds on top of the meta-ontology introduced in Section 3.2, and it is updated to use features from OWL 2.[6] Axiom (1) adds the transitivity of the `meta:subClassOf` property, so that indirect subsumptions are checked as well. Axioms (2)-(7) describe the tagging hierarchy and the

[6]The original ontology can be found at http://www.ontoclean.org/ontoclean-dl-v1.owl

complete partitions (i.e. each class is either `+R` or `-R`, `+U` or `-U`, etc.) of `meta:Class` as described in Section 6.3.2. The class `oc:Sortal` describes the identity meta-property. Finally, the axioms (8)-(11) describe the actual constraints, as partially given in Section 6.3.3. Just to take an example, Axiom (10) describes the constraint with regards to dependency, i.e. `+D` needs to be subsumed by `+D`. Note that the ontology, in particular axioms (1) and (8), infer that all rigid classes have to be subsumed by rigid classes.

We took the tagging created in the previous sections and formalized them in OWL DL as well, taking each tagging and interpreting it as a class instantiation of the respective class described in the OWL DL constraint ontology. Then we added the reification of the class hierarchy. For each tag, declare the individual that relates to the class from the original ontology to belong to the class corresponding to that tag within the constraint ontology. For example, if a class `C` is tagged `+R`, take the reifiied class $i_C$ and add the fact `ClassAssertion(oc:RigidClass` $i_C$`)`. If a class `C` is tagged `-U`, take the individual $i_C$ and add the fact `ClassAssertion(oc:NonUnityClass` $i_C$`)`, etc.

The thus created ontology can be simply checked for satisfiability by an OWL DL reasoner (actually, even much weaker reasoners would be sufficient due to the limited language fragment used in the constraint ontology). Standard reasoning services and ontology debugging tools can be applied in order to discover and repair inconsistencies.

For our experiments, we used RaDON (Ji *et al.*, 2009) a tool for inconsistency diagnosis based on KAON2 (Motik, 2006). RaDON features a number of algorithms for checking consistency and coherence of both TBox and ABox – among them an algorithm for identifying a set of minimal inconsistent subontologies for any given ontology. The algorithm starts with the inconsistent ontology, and (i) tries to find *any* minimal inconsistent subontology (Haase *et al.*, 2005). Then, (ii) it removes *any* axiom from this minimal inconsistent subontology – which fixes (at least) one inconsistency in this part of the ontology. Finally, (iii) the algorithm starts all over again beginning with step (i) until the whole ontology is consistent. Obviously, this algorithm is non-deterministic, but it gives us a good approximation of the total number of inconsistencies in the ontology.

The disadvantage of this algorithm is that it does not compute the absolute number of minimal inconsistent subontologies, because in step (ii) it potentially fixes more than one inconsistency. And finally, it is not deterministic. Therefore, different experiments can result in different sets of minimal inconsistent subontologies.

Take the classes `apple` and `food`. `apple` is tagged `+R`, whereas `food` is tagged `~R` (as described in Section 6.3.3). Now for the sake of the example let's assume that `apple` is defined as a subclass of `food`. We reify this axiom as described above, which results in the following formalization:

```
(a) ClassAssertion(oc:AntiRigidClass food)
(b) ClassAssertion(oc:RigidClass apple)
```

```
(c) PropertyAssertion(meta:subClassOf apple food)
```

Together with axiom (8) from the constraint ontology in Listing 6.1:

```
(8) EquivalentClasses(oc:RigidClass
          AllValuesFrom(meta:subClassOf
                        ComplementOf(oc:AntiRigidClass)))
```

This leads to an unsatisfiability: `apple` is a `RigidClass` (b), which has a local range axiom for the `subClassOf` relation (8) so that from the instantiated relation (c) we must infer that `food` belongs to the `ComplementOf(AntiRigidClass)` class description – which is a clear contradiction to the given fact (a) `ClassAssertion(AntiRigidClass food)`.

Reifying the subsumption hierarchy in OWL DL (Vrandečić *et al.*, 2006c) and the formalization of the OntoClean constraints also in OWL DL, allowed us to simply merge the two and reuse standard tools in order to detect inconsistencies with regards to the formal properties of the ontologies.

### 6.3.5 Analysis and Examples

For evaluating our approach, we have created a set of manual taggings for an ontology provided by three independent annotators $A_1$, $A_2$ and $A_3$ (Völker *et al.*, 2008). Table 6.1 shows the number of inconsistencies (each of them corresponding to a constraint violation) which were detected by RaDON for these tagging sets. On average, 17 constraint violations were found per annotator – most of them related to rigidity and identity. This also holds for the agreements, i.e. the data sets consisting of those taggings where two or three annotators agreed upon the same meta-property tagging for each concept. A further analysis and discussion of the annotator agreement can be found in (Völker *et al.*, 2008). As shown by the lower part of the table the overall number of inconsistencies drops for the intersection of any two human taggings to an average of 3.0 constraint violations per data set.[7] This can be explained by the fact that the number of agreed taggings is much lower than the overall number of tagged concepts.

How does this compare to the automatically generated taggings? After training a classifier on each of the data sets we obtained seven fully automatic taggings. Since AEON so far has not been trained to distinguish between an anti-rigid and non-rigid (respectively, anti-unity and non-unity) tagging we converted all taggings to their

[7] The agreement statistics represent lower bounds as they are computed in a cautious manner with respect to the number of possible inconsistencies. If at least one of the individual annotators tagged the regarding concept as `-R` whereas the others agreed upon ∼`R`, we assumed the agreement to be `-R` (or `-U`, respectively), i.e. the weaker tagging.

| | | Constraint Violations | | | |
|---|---|---|---|---|---|
| | Inconsistencies | R | U | D | I |
| $A_1$ | 24 | 20 | 2 | 1 | 1 |
| $A_2$ | 14 | 7 | 0 | 1 | 6 |
| $A_3$ | 13 | 3 | 0 | 0 | 10 |
| **avg** | **17.0** | **10.0** | **0.7** | **0.7** | **5.7** |
| $A_1$ / $A_2$ | 5 | 1 | 1 | 1 | 2 |
| $A_1$ / $A_3$ | 2 | 1 | 0 | 0 | 1 |
| $A_2$ / $A_3$ | 2 | 1 | 0 | 0 | 1 |
| **avg** | **3.0** | **0.3** | **0.3** | **1.0** | **1.3** |
| $A_1$ / $A_2$ / $A_3$ | 2 | 0 | 0 | 0 | 2 |

Table 6.1: Constraint Violations for Manual Taggings. The lower part of the table shows constraint violations based on taggings from the inter-annotater agreed sets, e.g. $A_2$ / $A_3$ shows the number of violations based on only the taggings where annotator $A_2$ and $A_3$ agreed on.



stricter counterpart wherever possible, in order to obtain an upper bound for the number of constraint violations. The results of the inconsistency diagnosis computed for these taggings are presented in Table 6.2. As expected, the average number of constraint violations per data set increased significantly from 17 to 40.7. The automatic unity taggings seems to cause far more inconsistencies than it is the case for any of the manual taggings – probably, due to the fact that anti-unity tags are very rare in the manually created tagging sets.

In the following we present some illustrative examples for inconsistencies which were detected in the ontology, and discuss possible reasons for the corresponding constraint violations.

We expected two different kinds of errors when analyzing the unsatisfiabilities: (i) the tagging was incorrect, e.g., because the annotators interpreted a concept in a different way than its author (presumably) did, or (ii) the subsumption relation was wrong, i.e. contradictory to a plausible meta-property assignment. We assumed that the OntoClean constraints as given in Listing 6.1 are correct.

An example of an error of the first kind is given by the following facts:

```
(a) ClassAssertion(oc:AntiRigidClass company)
(b) ClassAssertion(oc:RigidClass mediaCompany)
(c) PropertyAssertion(meta:subClassOf mediaCompany company)
```

The facts contradict axiom (8) from the constraint ontology:

| | | Constraint Violations | | | |
|---|---|---|---|---|---|
| | Inconsistencies | R | U | D | I |
| $A_1$ | 74 | 23 | 43 | 1 | 7 |
| $A_2$ | 17 | 3 | 8 | 5 | 1 |
| $A_3$ | 31 | 1 | 0 | 0 | 30 |
| **avg** | **40.7** | **9.0** | **17.0** | **2.0** | **12.7** |
| $A_1$ / $A_2$ | 13 | 1 | 6 | 1 | 5 |
| $A_1$ / $A_3$ | 5 | 2 | 0 | 0 | 3 |
| $A_2$ / $A_3$ | 7 | 3 | 0 | 0 | 4 |
| **avg** | **8.3** | **2.0** | **2.0** | **0.3** | **4.0** |
| $A_1$ / $A_2$ / $A_3$ | 3 | 0 | 0 | 0 | 3 |

Table 6.2: Constraint Violations for Automatic Taggings

```
(8) EquivalentClasses(oc:RigidClass
        AllValuesFrom(meta:subClassOf
                      ComplementOf(oc:AntiRigidClass)))
```

This error uncovers the improper tagging given by the taggers: `MediaCompany` and `Company` should be tagged in a compatible way. As of now, `Company` is said to be not rigid for all of its instances, whereas `MediaCompany` is rigid for its instances. Granted that the subsumption of `MediaCompany` by `Company` is correct, the error must be with the tagging of the two concepts. But here the taggers seem to have two different notions of companies in mind when they tagged `Company` and `MediaCompany`. An anti-rigid company is the role an organization can have: a university, which is an educational organization, can become a company; or a company can turn into a not-for-profit charity organization. In this case, the concept company means an organization that is meant to generate profit. On the other hand, a company that is tagged rigid actually is a type for individuals: now, a company can not cease to be a company any more, but a change as described above would basically require to generate a new individual. It depends heavily on the conceptualisation which of the two company concepts are useful within a given ontology. The example given above shows a confusion with regards to the concepts, and thus a possible source for errors.

An error of the second kind was discovered in the subsumption relation between `Group` and `PoliticalParty`, which is only an indirect relation: `Group` is, in PROTON, actually a superclass of `Organization` which in turn is a superclass of `PoliticalEntity` which is a superclass of `PoliticalParty`. The problem here is indeed in the subsumption relation between `Group` and `Organization`: a group is defined in PROTON as *"a group of agents, which is not organized in any way."* (Terziev *et al.*, 2005). This description is not applicable for an organization (since an organization is, by its very

nature and as shown in its name, organized). In formal terms, `Group` was tagged as ∼`U`, whereas `Organization` (and also PoliticalParty) were tagged as `+U`, which causes an inconsistency (based on axioms 1 and 24 of the constraints meta-ontology). The ontology would need to be changed to reflect that (such a change is discussed in (Guarino and Welty, 2004), where, incidentally, this very same pair, organization and group, is taken as an example).

Another example of a subsumption relation that required closer inspection was the subsumption of `Hotel` by `Building`. Is `Hotel` rather the role or the function of a building, whereas the building describes the object itself? Then the building would have a height, whereas the hotel would have a manager. A good example to illustrate the difference would be the number of rooms: it is counted differently for hotels than for buildings. This illustrates that it is not obvious if `Hotel` should be subsumed by `Building` or not, as the constraint violation suggested.

All the given examples are taken from the automatically tagged ontology. They illustrate that AEON points to possible taxonomic errors in an ontology, and guides the ontology engineer towards problematic parts of the ontology.



**Method 14 (OntoClean meta-property check)**
An ontology can be tagged with the OntoClean meta-properties and then automatically checked for constraint violations. Since the tagging of classes is expensive, we provide an automatic tagging system AEON.

All constraint violations, i.e. inconsistencies in the meta-ontology, come from two possible sources:

- an incorrect meta-property tagging, or
- an incorrect subsumption.

The evaluator has to carefully consider each inconsistency, discover which type of error is discovered, and then either correct the tagging or redesign the subsumption hierarchy.

## Chapter 7

# Semantics

So, so you think you can tell
heaven from hell,
blue skies from pain?

*(Pink Floyd, 1965–1994,*
*Wish You Were Here*
*(Waters and Gilmour, 1975))*



As in many other related fields, one can only control what one can measure (DeMarco, 1982). Measuring ontologies is necessary to evaluate ontologies both during engineering and application and is a necessary precondition to perform quality assurance and control the process of improvement. Metrics allow the fast and simple assessment of an ontology and also to track their subsequent evolution. In the last years, many ontology metrics and measures have been suggested and some initial work has been done to study the nature of metrics and measures for ontologies in general. We are extending this work.

There is a recurring set of problems with existing ontology metrics and measures. We have argued in Chapter 6 that most metrics are based on *structural notions* without taking into account the *semantics* which leads to incomparable measurement results. First, most ontology metrics are defined over the RDF graph that represents an OWL DL ontology and thus are basically graph metrics solely. Second, a very small number of metrics take the semantics of OWL DL into account (subsumption etc.). Third, few metrics consider the open world assumption. We believe that foundational work addressing these issues will substantially facilitate the definition of proper ontology metrics in the future.

In this chapter we will describe these issues in more detail, and suggest methods to avoid them. These issues are not always problematic: we will also explore under which circumstances they are problematic, and when they can be considered irrelevant. We

will outline the foundations for a novel set of metrics and measures, and discuss the advantages and problems of the given solutions. Our approach is based on notions of *ontology normalization* for measuring (Section 7.1), of *stable metrics* (Section 7.2), and of *language completeness* (Section 7.3). Normalization will help to properly define better ontology metrics in subsequent research in this area. Stability and completeness will help us understand metrics better.

## 7.1 Normalization

We define ontological, or semantic, metrics to be those which do not measure the structure of the ontology, but rather the models that are described by that structure. In a naïve way, we could state that we base our metrics not on the explicit statements, but on every statement that is entailed by the ontology.

But measuring the entailments is much harder than measuring the structure, and we definitively need a reasoner to do that. We also need to make a difference between a statement $X$ that is entailed by an ontology $O$ to be true ($O \models X$), a statement that is not entailed by an ontology ($O \not\models X$), and a statement that is entailed not to be true ($O \models \neg X$). To properly regard this difference leads us to so called stable metrics that can deal with the open world assumption of OWL DL.

Note that *measuring the entailments* is rather an intuitive description than an exact definition. In many cases – for example for a measure that simply counts the number of statements in an ontology – measuring all entailed statements instead of measuring all explicit statements often leads to an infinite number of statements. Just to give one example, the ontology

```
SubClassOf(
    SomeValuesFrom(R owl:Thing) C)
```

also entails the statements

```
SubClassOf(
    SomeValuesFrom(R
        SomeValuesFrom(R owl:Thing)) C)
```

and

```
SubClassOf(
    SomeValuesFrom(R
        SomeValuesFrom(R
            SomeValuesFrom(R owl:Thing))) C)
```

and so on, an endless chain of `SomeValuesFrom`-descriptions. But only terminating measures are of practical interest, and thus we need approaches that allow us to capture ontological metrics in a terminating way.

In order to gain the advantage of the simple and cheap measurement of structural features, we can transform the structure of the ontology. These transformation need to preserve the semantics of the ontology, that is, they need to describe the same models. But they also need to make certain semantic features of the ontology explicit in their structure – thus we can take structural measures of the transformed ontology and interpret them as ontological measures of the original ontology. We call this kind of transformations normalization (Vrandečić and Sure, 2007). We discuss this with examples in Section 8.1.

We define five normalization steps:

1. name all relevant classes, so no anonymous complex class descriptions are left (Section 7.1.1)
2. name anonymous individuals (Section 7.1.2)
3. materialize the subsumption hierarchy and normalize names (Section 7.1.3)
4. instantiate the most specific class or property for each individual and property instance (Section 7.1.4)
5. normalize property instances (Section 7.1.5)

Normalization offers the advantage that metrics are much easier defined on the normalized ontology since some properties of the graph are guaranteed: the ontology graph will have no cycles, the number of normal class names and actual classes will be equal, and problems of mapping and redundancy are dealt with. We give an example in Section 7.1.6 to illustrate some of the steps.

Often normalizations do not result in canonical, unique results (like, for example conjunctive normal forms). The normalization as described here can be extended in order to result in canonic normalized forms, but the benefits of such an extension are not clear. Considering that common serializations, like the RDF/XML serialization of OWL ontologies (Smith *et al.*, 2004), lack a canonic translation anyway, they cannot be compared on a character by character base as some version control systems like CVS or SVN would require.

Also, normalization is not an applicable solution for every metric. For example, if we want to know the number of atomic classes in an ontology, first normalizing it and then calculating the number actually will return the wrong result in the general case. The goal of normalization is to actually provide the metric designer some tools in order to simplify the description of their metric. In Section 8.1 we describe an example of how to apply the normalization for the description of a metric.

Further note that the algorithms provided in this section are merely effective but not efficient. They are given for the purpose of understanding normalization, and not as a blueprint for implementing them. Implementing the given algorithms will be unnecessarily slow, and more clever strategies for efficiently normalizing ontologies remain an open issue.

### 7.1.1 First normalization

In the **first normalization** our aim is to get rid of anonymous complex class descriptions.

After the first normalization, there will be only two types of class axioms left: definitions (i.e. class equivalence axioms between a simple class name and a class description) and simple subsumptions (i.e. subsumptions between two simple class names). Other class axioms (i.e. disjoints, disjoint unions, class equivalences involivng more than one complex class description, and subsumptions involving any complex class descriptions) will be reformulated. Class and property assertions will both use only simple class or property names, and no complex class descriptions.

The first normalization can be done as follows:

1. replace all axioms of the form
   `DisjointUnion(C D E ...)`
   by the following axioms:
   `EquivalentClasses(C UnionOf(D E ...))`
   `EquivalentClasses(owl:Nothing IntersectionOf(D E ...))`

2. replace all axioms of the form
   `Disjoint(C D)`
   by the following axiom:
   `EquivalentClasses(owl:Nothing IntersectionOf(C D))`

3. for every axiom of the form
   `SubClassOf(C D)`
   where `C` (or `D`) is a complex class description, add a new axiom
   `EquivalentClasses(A C)` (or `EquivalentClasses(B D)`)
   with `A` (or `B`) being a new class name. Replace the original axiom with
   `SubClassOf(A D)` (or `SubClassOf(C B)` or even `SubClassOf(A B)`)
   (so that only simple class names remain in the subsumption axiom)

4. in all axioms of the form
   `EquivalentClasses(C D)`
   where both `C` and `D` are complex class descriptions, replace that axiom with the following two axioms:
   `EquivalentClasses(A C)`

`EquivalentClasses(A D)`
with `A` being a new simple class name.

5. replace all axioms of the form
   `EquivalentClasses(C A)`
   where `C` is a complex class description and `A` a simple class name with the axiom
   `EquivalentClasses(A C)`

6. in all axioms having one of the following forms:
   `ClassAssertion(C a)`
   `PropertyDomain(R C)`
   `PropertyRange(R C)`
   `HasKey(C R S)`
   where `C` is a complex class description, replace `C` with `A` (being a new simple class name) and add the following axiom:
   `EquivalentClasses(A C)`

None of these structural changes change the possible models, that means, that they are semantically equivalent. They do introduce new class names to the ontology, which may not be desirable in all cases (for example for presentation purposes, for counting the classes, and so on), but it has to be noted that normalization is done only for measuring processes, and not for the purpose of engineering and processing the ontology (i.e., a normalized ontology is not meant to be published). Note that this way named classes could be introduced that are unsatisfiable. This does not mean that the ontology becomes unsatisfiable, but solely these newly introduced classes. In the third step (Section 7.1.3) we can remove these additional names again.

### 7.1.2 Second normalization

The **second normalization** gets rid of anonymous individuals. This means that every blank node that is an (asserted or inferred) individual needs to be replaced with an URI reference. Especially in FOAF files (Brickley and Miller, 2005) this occurs regularly since, for some time, it was regarded as good practice *not* to define URIs for persons. Integration of data was not done via the URI, but with inverse functional properties. This practice is problematic, since the semantics of blank nodes in RDF are rather often not fully understood, and should thus be avoided. The second normalization as defined here captures the semantics most users seem to want to express anyway, as exemplified by the discussions around FOAF on their mailing list. We already argued in Section 4.3 that such anonymous individuals should be avoided.

It is possible that these newly introduced individual names add a further name to already existing (or other newly introduced) individuals. But since OWL DL does not adhere to a unique name assumptions, this is no problem. Furthermore, the next step of normalization will take care to resolve such synonyms.

### 7.1.3 Third normalization

The **third normalization** will materialize the subsumption hierarchy and normalize the names. The first step requires a reasoner.

1. for all pairs of simple class names (`A`, `B`) in the ontology, add the axiom
   `SubClassOf(A B)`
   if the ontology entails that axiom (that is, materialize all subsumptions between simple named classes)

2. detect all cycles in the subsumption structure. For each set
   $M_i = \{\texttt{A}_1 \ldots \texttt{A}_n\}$
   of classes that participate in a cycle, remove all subsumption axioms from the ontology where both classes of the axiom are members of this set. For each such set $M_i$ introduce a new class name $\texttt{B}_i$. In subsumption axioms where only one class is a member of this set, replace the class with $\texttt{B}_i$ in the axioms. Add the axioms
   $\texttt{EquivalentClasses(B}_i\ \texttt{A}_1\texttt{)}, \ldots, \texttt{EquivalentClasses(B}_i\ \texttt{A}_n\texttt{)}$
   to the ontology. If $\texttt{B}_i$ is unsatisfiable, take `owl:Nothing` instead of $\texttt{B}_i$. If $\texttt{B}_i$ is equal to `owl:Thing`, take `owl:Thing`

3. regarding solely the subontology that consists of all subsumption axioms of the ontology, remove all redundant subsumption axioms (that is, remove all subsumption axioms that are redundant due to the transitivity of the subsumption relation alone). This also removes all subsumption axioms involving `owl:Thing` and `owl:Nothing`

The subsumption structure now forms a directed acyclic graph that represents the complete subsumption hierarchy of the original ontology. We define a set of normal class names of an ontology as follows: every class name that participates in a subsumption axiom after the third normalization of an ontology is a normal class name of that ontology.

Now in all axioms of the types `ClassAssertion`, `PropertyDomain`, `PropertyRange`, and `HasKey` we can replace all not normal class names with its equivalent normal class names.

Note that instead of creating a new class name for each detected cycle, often it will make more sense to choose a name from the set of classes involved in that cycle, based on some criterion (e.g. the class name belonging to a certain namespace, the popularity of the class name on the Web, etc.). For many ontology metrics, this does not make any difference, so we disregard it for now, but we expect the normalizations to have beneficial effects in other scenarios as well, in which case some steps of the normalization need to be revisited in more detail.

Compared to classes, properties are often neglected. Besides inverse properties no other complex property descriptions can be stated in OWL. Therefore property normalization can be regarded as normalizing inverses and property names analogous to class name normalization. All normal property names have to be stated explicitly to be equivalent to all other property names they are equal to (that is, we materialize the equality relations between the normal property names and the non-normal ones). All occurrences of non-normal property names (besides within the axiom stating equality with the normal property name, and besides within annotation property instances) are replaced with the normal property name. We can also normalize the property hierarchy just as we normalized the class hierarchy.

The same holds true for individuals. In case an individual has more than one name, we decide on or introduce a normal one and state explicitly equality to the normal name, and then replace all occurrences of the non-normal individual names with the normal one (besides within the axiom stating equality with the normal individual name, and besides within annotation property instances).

We disregard annotation property instances since they may be used to state annotations about the URI, and not about the actual class, property, or individual. There could be annotations that describe when a certain URI was introduced, who created it, its deprecation state, or that point to a discussion related to the introduction of the URI. Some annotations on the other hand may be useful for the normal name as well – especially labels, or sometimes comments. Since annotations do not have impact on the DL semantics of the ontology anyway, they may be dropped for the purpose of measuring semantic metrics. Nevertheless, if the normalization is done for some other purpose, and it is planned to further use the normalized version of the ontology in some scenario, then the possible replacement of names within annotation property instances depends both on the scenario and the instantiated annotation property (for example, it may be useful to normalize the label when the ontology will be displayed on the user interface, but it may be bad to normalize versioning information that is captured within annotations).

### 7.1.4 Fourth normalization

The **fourth normalization** aims towards instantiating the most specific classes and properties, as this conveys the most information explicitly (and deriving instantiations of higher levels is very cheap because of the asserted explicitness of the hierarchy due to third normalization). This does not mean that every instance will belong to only one class, multiple instantiations will still be necessary in general.

Here is a possible algorithm to perform the fourth normalization of an ontology $O$.

1. for each normal class name `C` and each normal individual name `i` in $O$, add `ClassAssertion(C i)`

to $O$ if it is entailed by the ontology

2. for each normal object property instance
   `PropertyAssertion(R i j)`
   and each normal object property name `S` so that `SubPropertyOf(S R)`
   is an explicit axiom in $O$, add `PropertyAssertion(S i j)`
   if it is entailed by the ontology. Check this also for the property instances added this way (this step will terminate since the subsumption hierarchy is finite)

3. for each normal data property instance
   `PropertyAssertion(T i d)`
   and each normal data property name `U` proceed as in the previous step.

4. create a subontology $I_O$ out of $O$ including only the facts and the explicitly stated subsumption hierarchy of the classes and properties (after third normalization)

5. remove all facts from $O$ that are redundant in $I_O$

We do not want to remove all redundant facts from the ontology at this step, since there may be some facts that are redundant due to an interplay of different other terminological axioms. For example, in the following ontology

```
ClassAssertion(Person Adam)
PropertyAssertion(likes Adam Eve)
PropertyDomain(likes Person)
```

the first statement is actually redundant, but would not be removed by the above algorithm. This is because we only remove axioms that are redundant within the subontology $I_O$, and the axiom stating the domain of `likes` would not be part of it.

### 7.1.5 Fifth normalization

The **fifth normalization** finally normalizes the properties: we materialize property instances of symmetric, reflexive and inverse properties, and we clean the transitivity relationship. This can be done similar to the creation of the subsumption hierarchy in the third normalization: after materializing all property instances, we remove all that are redundant in the subontology $T_O$, which contains only the property instances of all transitive properties, and the axioms stating the transitivity of these properties.

### 7.1.6 Examples of normalization

The metric we will regard in this example is the *maximum depth of the taxonomy* as defined by (Lozano-Tello and Gómez-Pérez, 2004) and described in Section 6.1.1.

What we want to measure is intuitively described as the length of the subsumption hierarchy, or else the number of levels the class hierarchy has. We name the measure *md*.

Let us regard the following ontology:

```
EquivalentClasses(C MinCardinality(1 R))
EquivalentClasses(D MinCardinality(2 R))
EquivalentClasses(E MinCardinality(3 R))
```

By the definition of *md*, the depth of the ontology is 1 (since there are no explicitly stated subsumption axioms, every path has one node). But after normalization the ontology gets transformed to this:

```
EquivalentClasses(C MinCardinality(1 R))
EquivalentClasses(D MinCardinality(2 R))
EquivalentClasses(E MinCardinality(3 R))
SubClassOf(D C)
SubClassOf(E D)
```

Now the very same metric, applied to the normalized ontology, actually captures the intuition of the depth of the ontology and returns 3.

As discussed earlier, this example also shows us that some metrics will not work with normalization. In (Gangemi *et al.*, 2005), metric (M30) is the *axiom/class ratio*. On the original ontology it is 1, but raises to 5/3 in the normalized version. In case the original ontology is being distributed and shared, (M30) – if stated as metadata of the ontology, for example in some kind of ontology repository (Hartmann *et al.*, 2005) – should be 1, and not calculated on the normalized version.

Let us regard another example. In the following ontology

```
SubClassOf(D C)
SubClassOf(E D)
SubClassOf(D E)
SubClassOf(F E)
```

*md* will be $\infty$ due to the subsumption cycle between $D$ and $E$. The cycle can be resolved by rewriting the axioms in the following way:

```
SubClassOf(D C)
EquivalentClasses(D E)
SubClassOf(F E)
```

But due to the definition, *md* would yield 2 here – there are two explicit subsumption paths, (`C`, `D`) and (`E`, `F`), both having two nodes, and thus the longest path is 2. The structural measure again does not bring the expected result. After normalization, though, the ontology will look like this:

```
SubClassOf(A C)
EquivalentClasses(A D)
EquivalentClasses(A E)
SubClassOf(F A)
```

We have introduced a new class name `A` that replaces the members of the cycle (`D`, `E`). Now the depth of the ontology is 3, as we would have expected from the start, since the cycle is treated appropriately.

Existing structural metrics, as discussed in Chapter 6, often fail to capture what they are meant for. Normalization is a tool that is easy to apply and that can easily repair a number of such metrics. Even seemingly simple metrics, as demonstrated here with the ontology depth, are defined in a way that makes too many assumption with regards to the structure of the measured ontologies.

As we can see in this chapter, simple structural measures on the ontology do yield values, and often these values may be highly interesting. If we know that *md* resolves to $\infty$, then this tells us that we have a cycle in the subsumption hierarchy. Also a high number of classes and complex axioms, but a low *md* may indicate an expensive to reason about ontology, since the major part of the taxonomy seems to be implicitly stated (but such claims need to be evaluated appropriately). But both results do not capture what the measure was meant to express, that is, the depth of the class hierarchy.

## 7.2 Stability

Another aspect of semantic metrics is their *stability* with regards to the open world assumption of OWL (Vrandečić and Sure, 2007). The question is, how does the metric fare when further axioms are added to the ontology? For example, a taxonomy may have a certain depth, but new axioms could be added that declare the equivalence of all leaves of the taxonomy with its root, thus leading to a depth of 1. This often will not even raise an inconsistency, but is still an indicator for a weak ontology. Stable metrics are metrics that take the open world assumption properly into account. Stable metrics allow us to make statements about the behavior of an ontology in the context of a dynamic and changing World Wide Web, where ontologies may frequently be merged together in order to answer questions over integrated knowledge.

When an ontology is built, a stable metric will indicate if and how an ontology can be changed. An ontology engineer can prevent certain changes that will render the ontology useless. By adding more heavy-weight axioms to the ontology (for example complete partitions) the minimal depth may raise, indicating a more robust ontology with regards to future changes. Stable metrics are indicators for stable ontologies.

Stability with regards to the knowledge base can also be used by closing certain classes. In some cases we know that a knowledge base offers complete coverage: for example, we may publish a complete list of all members of a certain work group, or a complete list of all countries. In this case we can use nominals to close off the class and declare its completeness. But note that such a closure often has undesirable computational side effects in many reasoners.

Often metrics intend to capture features of the ontology that are independent of the actual representation of the ontology. But as we have seen, structural transformations of the ontology description often lead to differences in the metrics even though the semantics remained untouched. Normalization offers a way to overcome these problems in many cases.

In order to illustrate metrics stability, consider the following ontology:

```
PropertyAssertion(author paper York)
PropertyAssertion(author paper Denny)
PropertyAssertion(author paper Zdenko)
```

Now let us ask the simple question: how many authors does the `paper` have? It seems that the answer should be 3. But now, if you knew that `Zdenko` is just another name for `Denny`, and thus state

```
SameIndividual(Zdenko Denny)
```

then you suddenly would change your answer to 2, or even, becoming more careful, giving an answer such as *"I am not sure, it is either 1 or 2"*. So finally we can state that

```
DifferentIndividuals(York Denny)
```

and thus arrive at the answer that the paper indeed has 2 authors (and even that is possibly wrong if we consider that we could add statements any time in an open world that add further authors to the paper – all we know *as of now* is that the paper has *at least* two authors).

When creating a metric, we have to ask ourselves the following, similar question: how does the metric behave when additions to the ontology happen? Since ontologies are meant to be smushed and integrated constantly and dynamically, can we predict how certain properties of the ontology will behave, that is, if $M(O_1)$ and $M(O_2)$ for a metric

$M$ and two ontologies $O_1$ and $O_2$ are known, what can we state about $M(O_1 \cup O_2)$? Or even, can we give a function $f_M$ so that we can calculate $f_M(M(O_1), M(O_2)) = M(O_1 \cup O_2)$ without having to calculate $M(O_1 \cup O_2)$ directly (which may be much more expensive)?

In the previous section we have discussed the simple example of ontology depth. Given an ontology $O_1$:

```
SubClassOf(D C)
SubClassOf(E D)
```

and a second ontology $O_2$:

```
SubClassOf(C D)
SubClassOf(E D)
```

In this case, $md(O_1) = 3, md(O_2) = 2$. We may expect $md(O_1 \cup O_2)$ to be 3, since $md$ is defined as the maximal depth, but since the union of both ontologies actually creates a cycle in the subsumption hierarchy, $md$ is $\infty$ – or, after normalization, just 2, and thus even smaller than the maximal depth before the union.

We can avoid such behavior of the metrics by carefully taking the open world assumption into account when defining the metric. But this leads us to three possibilities for defining metrics,

1. to base the value on the ontology as it is,
2. to measure an upper bound, or
3. to measure a lower bound.

We need a more complicated example to fully demonstrate these metrics:

```
DisjointUnion(C D E)
SubClassOf(F E)
EquivalentClasses(G ComplementOf(C))
SubClassOf(H C)
ClassAssertion(F i)
ClassAssertion(D j)
ClassAssertion(G k)
```

The normalized version of this ontology looks like this (shortened slightly for readability):

```
EquivalentClasses(C UnionOf(D E))
EquivalentClasses(owl:Nothing IntersectionOf(D E))
SubClassOf(D C)
SubClassOf(E C)
SubClassOf(F E)
EquivalentClasses(G ComplementOf(C))
SubClassOf(H C)
ClassAssertion(F i)
ClassAssertion(D j)
ClassAssertion(G k)
```

*md* of this ontology is 3 (`C`, `E`, `F`). But besides the actual depth, we can also calculate the minimal depth of this ontology, that is, no matter what axioms are added, what is the smallest number of levels the ontology class hierarchy will have (under the condition that the ontology remains satisfiable)?

In the given example, if we add the axiom `EquivalentClasses(F E)` *md* will decrease to 2. But on the other hand, no matter what axiom we further add, there is no way to let `C` collapse with `D` and `E`, therefore `C` is a proper superset of both (that is, it contains more individuals than each `D` or `E` alone). And because `C` cannot become `owl:Thing` (due to `k` being outside of `C`), the minimum depth of the ontology is 2.

The maximum depth of an ontology is usually $\infty$ (since we can always add axioms about an arbitrarily long class hierarchy). Therefore we need to define a maximum depth in a slightly different way in order to be of practical value. In the following, we will discuss two possible definitions.

Instead of allowing for arbitrary axioms that may be added, we only allow to add axioms of the form `SubClassOf(A B)` with `A` and `B` being normal class names of the normalized ontology. Thus, in the above example, we may add the axiom `SubClassOf(H F)` to the ontology in order to increase *md* from 3 to 4. No longer subsumption path is possible, since all the other named classes would become unsatisfiable when added to an existing path. So this metric will provide with a maximum depth of the ontology, assuming no new class names are added.

Another possibility to constrain the axioms to be added, is to allow only for axioms that do not relate to the existing ontology, that is, the intersection of the signatures of the two ontologies is empty. The signature of an ontology is the set of all names used in the ontology (besides the names from the OWL, RDF, RDFS, and XSD namespaces). In this case, *md* of the merged ontology is the maximal *md* of the single ontologies, since no interaction between the axioms can happen that may increase or reduce *md*. We can thus define

$$f_{md}(md(O_1), md(O_2)) = \mathsf{max}(md(O_1), md(O_2))$$

which is much cheaper to calculate than $md(O_1 \cup O_2)$.

Stable metrics are metrics that take the open world assumption into account. Stable metrics will help us to evaluate ontologies for the Wide Wild Web. Since we expect ontologies to be merged on the Web dynamically, stable metrics allow us to state conditions that the ontology will fulfill in any situation. The depth of an ontology may be a too simple example to demonstrate the advantages of stable metrics, but imagine a dynamic, ontology-based graphical user interface. Having certain guarantees with regards to the future development of the properties of the ontology may help the designer of the user interface tremendously, even if it is such a seemingly trivial statement such as *"the depth of the ontology is never less than 3"*.

There is no simple recipe to follow in order to turn a metric into a stable metric, but the question outlined at the beginning of this section, and then discussed throughout the rest – how does the ontology behave when axioms are added? – can be used as a guideline in achieving a stable metric.

**Method 15 (Ensuring a stable class hierarchy)**
Calculate a normalized class depth measure, i.e. calculate the length of the longest subsumption path on the normalized version of the ontology $md(N(O))$. Now calculate the stable minimal depth of the ontology $md^{min}(O)$. If

$$md(N(O)) \neq md^{min}(O)$$

then the ontology hierarchy is not stable and may collapse.

We expect that the ready availability of metrics that take the open world assumption into account will lead to more robust ontologies. Since ontology engineers will have these numbers available at engineering and maintenance time, they will learn easier how to achieve their actual goals. For example, ontology engineers that want to create a class hierarchy that will not collapse to less levels can always check if the minimum depth as described above corresponds to the asserted depth. This would be useful when regarding a class hierarchy with a certain number of levels, which are known not to collapse (e.g. a biological taxonomy). The ontology engineer now could check if the well known number of levels indeed corresponds to the calculated minimum depth.

Tools could guide the ontology engineer towards achieving such goals. Ontology engineers get more aware of such problems, and at the same time get tools to measure, and thus potentially control them.

## 7.3 Language completeness

*Language completeness* is defined on a certain ontology with regards to a specific ontology language (or subset of the language). Given a specific signature (i.e. set of names), language completeness measures the ratio between the knowledge that *can* be expressed and the knowledge that *is* stated. For example, if we have an ontology with the signature `Adam`, `Eve`, `Apple`, `knows`, `eats`, `Person`, `Food`, we can ask which of the individuals are persons, and which of the individuals know each other.

Thus assuming a simple assertional language such as RDF, language completeness with regards to that language (or short: *assertional completeness*) is achieved by knowing about all possible ground facts that can be described by the ontology (i.e. for each fact $\{\texttt{ClassAssertion(C i)}|\forall \texttt{C} \in O, \forall \texttt{i} \in O\} \cup \{\texttt{PropertyAssertion(R i j)}|\forall \texttt{R} \in O, \forall \texttt{i} \in O, \forall \texttt{j} \in O\}$ we can say if it is true or not, and none of them is unknown).

An expressive ontology language allows numerous more questions to be asked besides ground facts: is the domain of `knows` a `Person`? Is it the range? Is `eats` a subproperty of `knows`? In order to have a language complete ontology with regards to the more expressive language, the ontology must offer defined answers for all questions that can be asked with the given language.

**Method 16 (Measuring language completeness)**
We define a function $\Upsilon_i$ with the index $i$ being a language fragment (if none is given, the assertional fragment is assumed) from an ontology $O$ to the set of all possible axioms over the signature of $O$ given the language fragment $i$.

We introduce $C_i$ as *language completeness* over the language fragment $i$.

$$C_i(O) = \frac{|\{X|X \in \Upsilon(O), O \models X \vee O \models \neg X\}|}{|\Upsilon(O)|}$$

Note that the language fragment the completeness measure is using is not tied to the language fragment the ontology is using. Consider for example the following ontology using the above example signature.

```
Disjoint(Food Person)
PropertyDomain(knows Person)
PropertyRange(eats Food)
ClassAssertion(Person Adam)
PropertyAssertion(knows Eve Adam)
PropertyAssertion(eats Eve Apple)
```

With the help of Table 7.1 we can calculate the assertional completeness $C(O) = \frac{17}{24} \approx 0.71$. We see that we are using a far more expressive language to state the ontology than the simple assertional fragment we use for calculating the completeness. Relational exploration is a method to explore language fragments of higher expressivity, and to calculate the smallest set of questions that have to be answered in order to achieve a language complete ontology (Rudolph *et al.*, 2007). In order to improve completeness we can thus add further axioms, either by adding more facts such as

```
PropertyAssertion(knows Adam Eve)
NegativePropertyAssertion(eats Apple Apple)
```

or by adding terminological axioms that allow to infer that certain facts hold, such as

```
SymmetricProperty(knows)
IrreflexiveProperty(eats)
```

which in this case adds exactly the same amount of information to our given signature using the same number of axioms (i.e. improving the completeness to $\frac{19}{24} \approx 0.79$).

| **Class** | Apple | Adam | Eve |
|---|---|---|---|
| Food | ✓ | ✗ | ✗ |
| Person | ✗ | ✓ | ✓ |

| **knows** | Apple | Adam | Eve |
|---|---|---|---|
| Apple | ✗ | ✗ | ✗ |
| Adam | ? | ? | ? |
| Eve | ? | ✓ | ? |

| **eats** | Apple | Adam | Eve |
|---|---|---|---|
| Apple | ? | ✗ | ✗ |
| Adam | ? | ✗ | ✗ |
| Eve | ✓ | ✗ | ✗ |

Table 7.1: Class and property assertions to calculate the language completeness of the example ontology.

Even though both sets of axioms improve the ontology with the same information content, the second set seems intuitively better as it further describes the terms intensionally instead of just adding facts extensionally. How can we capture that in a completeness metric?

Instead of using assertional completeness, which indeed is not suited for capturing intensional completeness, we have to use a more expressive language fragment. For example, by adding the symmetry axiom to the language fragment used for computing language completeness, we see that the second set indeed displays a higher completeness (0.77) than the first set (0.73). The more expressive the language fragment used for calculating the completeness, the more the measure will reflect the value of intensional axioms.

## Chapter 8

# Representation

*Ceci n'est pas une pipe.*

*(René Magritte, 1898–1967,*
*The Treachery of Images*
*(Magritte, 1929))*



Representational aspects of an ontology deal with the relation between the semantics and the structure, i.e. the way the semantics are structurally represented. This will often uncover mistakes and omissions within the relation between the *formal specification* and the *shared conceptualization* – or at least the models which are supposedly isomorphic to the conceptualizations.

## 8.1 Ontological metrics

Normalization helps with the definition of ontological metrics, since they will help a metric designer in being explicitly aware in their choices when creating the metric. Furthermore they offer the ontology designer ready to use methods to easier capture what they mean to express with the designed metric. Sometimes simple structural metrics are sufficient for the task at hand, and many structural metrics exist today. Sometimes an understanding of the semantic model is required, and we have introduced a way to gain that.

The aspect of representation covers how the structure represents the semantic. In order to evaluate features of the representation, we compare the results of the structural measures to the results of the semantic measures. Using the normalization described in Chapter 7, we can even often use the same (or a very similar) metric as described in Chapter 6, applied before and after normalization, and compare the respective results. Any deviations between the results of the two measurements indicate elements of the

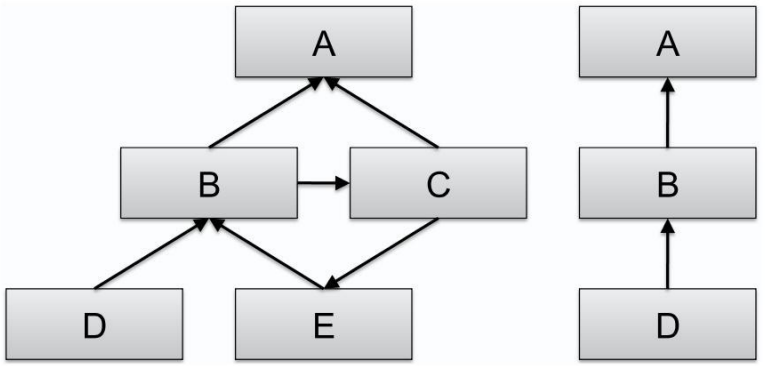


Figure 8.1: A simple taxonomy before (left) and after (right) normalization. The arrows denote subsumption.

ontology that require further investigation. For example, consider the ontology given in Figure 8.1. The number of classes before normalization is 5, and after normalization 3. This difference shows that several classes collapse into one, which may be an error or done by intention. In case this is an error, it needs to be corrected. If this is done intentionally, the rationale for this design decision should be documented in the ontology. This becomes especially evident if you imagine removing any single one of the subsumption relations between `B`, `C` and `E`. The result will be very different, i.e. such an erroneous axiom has a high impact on the resulting conceptualization.

By contrasting the two ontology structures in Figure 8.1 we see that the right one is a more faithful representation of the semantics of the ontology. Both structures have the same semantics, i.e. allow the same sets of models. The right one is more concise, and for most cases more suitable than the left one. Evaluation methods dealing with representational aspect can uncover such differences and indicate problematic parts of an ontology (Vrandečić and Sure, 2007).

In the remainder of this Section, we will discuss the four metrics we have introduced in Section 6.1 as examples of how they can be turned into semantic metrics that actually reflect their descriptions. This is a prerequisite for the discussion of the representational metrics introduced subsequently, and their meaning. For notation, we formalize the five steps of normalization as the functions $N_1$ to $N_5 : \mathcal{O} \to \mathcal{O}$, where $N_{i+1}(O)$ always means $N_{i+1}(N_i(O))$, $N_0$ is the identity function, and $N(O)$ being a shortcut for $N_5(O)$.

## 8.2 Maximum depth of the taxonomy

In Section 6.1.1.we have introduced the *maximum depth of the taxonomy* metric as defined by (Lozano-Tello and Gómez-Pérez, 2004). We have argued that the metric would be more appropriately be named *maximum subsumption path length* instead. Now that we have the tool of normalization at hand, we can actually use that in order to capture the original meaning of the metric: the *maximum depth of the taxonomy* (let that be $TD$) of an ontology $O$ equals the *maximum subsumption path length* (let that be $SL$) of the normalized version of the ontology $O$ (to be exact, after the third normalization), i.e. $TD(O) = SL(N_3(O))$. This resolves all the problems we have mentioned in Section 6.1.1 and the result of the metric does indeed capture its meaning.

Now we can introduce a new metric, $ET(O) = \frac{TD(O)}{SL(O)}$ which describes the *explicitness of the subsumption hierarchy.*

**Method 17 (Explicitness of the subsumption hierarchy)**
Calculate $ET(O)$.

- If $ET(O) = 1$ everything seems fine
- If $ET(O) < 1$ then some of the classes in the ontology have collapsed. Find the collapsed classes and repair the explicit class hierarchy
- If $ET(O) > 1$ part of the class hierarchy has not been explicated. Find that part and repair the class hierarchy

Note that this test does not necessarily discover all errors – one could imagine an ontology where parts of the class hierarchy collapse, and part of the class hierarchy is not explicated so that the result balances out to 1 again. But whenever the metric $ET(O)$ does not result in 1, there is a high probability of an error. In order to find all possible errors in the class hierarchy we would rather calculate

$$D = H(O)/H(N(O)) \cup H(N(O))/H(O)$$

with $H : \mathcal{O} \rightarrow \mathcal{O}$ a function that selects only the *simple subsumptions*. This calculates all single subsumptions that are not part of the hierarchy, and the other way around. Each axiom $x \in D$ is thus a potential problematic axiom that should be checked.

## 8.3 Class / relation ratio

In Section 6.1.2 we discussed the (M29) *class / relation ratio* from (Gangemi *et al.*, 2005) being

$$M29(O) = \frac{|C(O)|}{|P(O)|}$$

(with $C(O)$ yielding the set of used class names in $O$, and $P(O)$ the set of property names in $O$). We have shown that the metric would be better named *class name / property name ratio*. But making the same modification as above and yielding a new metric

$$M29^*(O) = M29(N(O)) = \frac{|C(N(O))|}{|P(N(O))|}$$

also would not yield the ratio between classes and relations, since the normalization does not remove synonymous class and property names (but rather potentially adds further such names).

Instead we need to define a metric that counts the number of normal class and property names (we do that by introducing $C_N(O)$ yielding the set of normal class names in the normalized version of $O$ and $P_N(O)$ yielding the set of normal property names in the normalized version of $O$), and thus leading to a new metric

$$N29(O) = \frac{|C_N(O)|}{|P_N(O)|}$$

Comparing the two ratios $M29(O)/N29(O)$ does not yield a value with an obvious meaning. Instead we should regard the ratio between each of the two components, i.e. the *ratio of classes and class names*

$$R_C(O) = \frac{|C_N(O)|}{|C(O)|}$$

and the *ratio of properties and property names*

$$R_P(O) = \frac{|P_N(O)|}{|P(O)|}$$

**Method 18 (Explicit terminology ratio)**
Calculate $R_C(O)$ and $R_P(O)$.

- If $R_C(O) = R_P(O) = 1$ then this indicates no problems with the coverage of elements with names in the ontology

- If $R_C(O) < 1$ or $R_P(O) < 1$ and the ontology does not include a mapping to an external vocabulary then this indicates possible problems since a number of names have collapsed to describe the same class
- If $R_C(O) < 1$ or $R_P(O) < 1$ and the ontology includes a mapping to an external vocabulary we can remove all axioms providing the mapping and calculate $R_C(O')$ and $R_P(O')$ anew
- If $R_C(O) > 1$ or $R_P(O) > 1$ then this indicates that not all interesting classes or properties have been given a name, i.e. the coverage of classes and properties with names may not be sufficient

In this metric we see that we cannot just mechanically replace the ontology with its normalized version in order to yield a metric that fits to the desired definition. It is also not true that simply comparing the metrics of the normalized and the original ontology yields interesting metrics that provides us with more insight in the quality of the ontology.

## 8.4 Relationship richness

We redefine the original *relationship richness* metric from (Tartir *et al.*, 2005) as described in Section 6.1.3 here as

$$RR^*(O) = \frac{|P(O)|}{|H(O)| + |P(O)|}$$

(with $H$ as defined in Section 8.2 and $P$ as defined in Section 8.3). As discussed in Section 6.1.3 this value is pretty much meaningless as it is.

In order to repair this we cannot just redefine $RR(O) = RR^*(N(O))$ because $P(N(O))$ does not yield the number of properties. Instead we could use the number of normal property names, resulting in

$$RR(O) = \frac{|P_N(O)|}{|H(N(O))| + |P_N(O)|}$$

which is probably the closest we can get to the original intended definition.

Regarding the rationale for this metric, the authors are looking for a metric that *"reflects the diversity of relations and placement of relations in the ontology"* and is based on the assumption that an ontology that *"contains many relations other than class-subclass relations is richer than a taxonomy with only class-subclass relation-*

*ships"*. We question this assumption: it seems more straightforward to simply use the number of classes than the number of class-subclass relations. We do agree with the original definition that the relationship richness should be of the form $\frac{|P_N(O)|}{\mathcal{X}+|P_N(O)|}$, but we disagree that $|H(N(O))|$ is a good value for $\mathcal{X}$ but instead we suggest $|C_N(O)|$. In order to understand the difference we first investigate the relationship between the number of class-subclass relations and the number of classes in an ontology.

We understand the number of class-subclass relations to be $|H(N(O))|$, i.e. the number of *simple subsumptions* in the normalized ontology. The number of classes is $|C_N(O)|$, i.e. the number of normal class names in an ontology. Now we can define the set of all *root classes*, i.e. of all classes that have no given superclass (besides `owl:Thing`) as

$$R(O) = \{\texttt{C} | \texttt{C} \in C_N(O) \wedge \forall \texttt{D} \in C_N(O) : \texttt{SubClassOf(C D)} \notin H(N(O))\}$$

Further we can define the *treelikeness* of the class hierarchy as

$$t(O) = \frac{|C_N(O)/R(O)|}{|H(N(O))|}$$

(or 0 if $|H(N(O))| = 0$). The closer the value to 1 the more treelike the class hierarchy is. So if there is exactly one simple subsumption for each class that is not a root class then the treelikeness of the class hierarchy is 1 (this allows us to easily give a formal definition for the terms *tree* and *set of trees* describing the taxonomy of an ontology: a tree is given if $t(O) = 1 \wedge |R(O)| = 1$, a set of trees if $t(O) = 1 \wedge |R(O)| > 1$).

So if $|C_N(O)|$ is fixed, an increased value of $|H(N(O))|$ leads to a less tree-like class hierarchy, but has no other obvious effect on the ontology. The treelikeness of the hierarchy seems to be independent of the relationship richness. Therefore we suggest to choose another function for $\mathcal{X}$: obvious candidates seem to be the size of the ontology, i.e. the number of axioms $|O|$, the number of terminological axioms, or simply the number of classes. We think that the number of classes is a better choice, since a growth in the number of axioms but having a fixed number of entities indicates an overall growth of richness. Therefore it would be counterintuitive for the relational richness to decrease if existing classes are described in more detail. This effect does not happen if we choose $\mathcal{X}$ to be instead the number of classes, i.e. $|C_N(O)|$.

So we suggest the best metric to capture relational richness that is still close to the original metric as defined by (Tartir *et al.*, 2005), to be

$$RR(O) = \frac{|P_N(O)|}{|C_N(O)| + |P_N(O)|}$$

Again it would not make sense to compare this metric to the original metric. But we see that normalization is a useful tool to define metrics more precisely, and to get

closer to what we want to capture with a given metric.

## 8.5 Semantic similarity measure

The semantic similarity measure is a function $ssm : \mathcal{O} \times \mathcal{C} \times \mathcal{C} \rightarrow \mathbb{R}$ that describes the similarity of two classes in an ontology. It is defined as the reciprocal value of the distance over the subsumption graph of the ontology. Continuing the argumentation from Section 6.1.4, we can see that first normalizing the ontology and then calculating the distance avoids a number of pitfalls.

The problem we run into here, though, is that normalization removes all explicit subsumption axioms connecting `owl:Thing` with the root classes $R(O)$ thus potentially leading to a number of disconnected graphs. In this case, we can either define $ssm$ to be 0, or we can add a further step to the normalization procedure, namely $\forall \texttt{C} \in R(O)$ add the axiom `SubClassOf(C owl:Thing)` to the ontology. This way we connect the whole subsumption graph and will always have a finite number for the distance.

When we introduced normalization, we stated explicitly that the given steps are neither necessary nor sufficient for all tasks. They provide a metric engineer with a new tool to define and test metrics, but as we see in this example we need to further extend the preprocessing of the ontology before we can measure the value of a specific metric. Again, the given step is a semantic-preserving syntactic transformation, but it is required in order to measure the ontology.

It is unclear what the comparision of $ssm$ over the original and the normalized ontologies will yield. Whereas one may think that the normalized ontology provides a better base for calculating a semantic similarity, it may indeed be the case that the opposite is true: usually, even though a redundant subsumption axiom may be represented in the ontology, the inclusion of such an axiom often has a rationale. In learned ontologies, it may be based on the evidence for exactly this relation, in human-engineered ontologies it may be to put emphasis on a specific relation (even though that does not mean anything in the formal semantics). So explicit connections may indicate some semantic closeness that the formal semantics do not properly capture.

This evaluation is out of scope for this thesis. Our task is to provide a framework in which to discuss and define metrics for ontology qualities. We have shown in this section a number of metrics as example and demonstrated the usefulness of the tools provided in this thesis.

## Chapter 9

# Context

**Neo:** I just have never...
**Rama-Kandra:** ...heard a program speak of love?
**Neo:** It's a...human emotion.
**Rama-Kandra:** No, it is a word. What matters is the connection the word implies.

*(Matix Revolutions (Wachowski and Wachowski, 2003))*



There are a number of approaches in the literature describing the creation and definition of artifacts accompanying an ontology. An evaluating tool can load both the additional artifact and the ontology and then perform further evaluations. The additional artifact thus provides a context for the ontology and the evaluation method. Note that we do not consider the whole of the Semantic Web to be a context in this sense. Thus we use the Web for evaluations of other aspects as well, be it for checking the vocabulary (as for linked data in Section 4.1.1) or for automatically deriving properties of classes (as in AEON in Section 6.3). Only specific further artifacts used as input to an evaluation process is considered context within our framework.

One of the earliest approaches toward ontology evaluation was the introduction of *competency questions*, i.e. questions that the ontology should be able to answer (Grüninger and Fox, 1995). In order to enable the automatic evaluation with regards to competency questions, the competency questions need to be formalized in a query language that can be used by the tool the ontology is developed for. A first intuition would claim that the the query language has to be expressive enough to encode the competency questions. If it is not expressive enough, the relevance of the competency

questions needs to be assessed: if the ontology-based tool cannot ask the question, why should the correct answer be important? The additional artifact, in this case, would be the set of formalized competency questions and the required correct answers.

But in some cases a list of formalized competency questions and their correct answers is not feasible or possible to generate. Instead, we often can state certain constraints the answers need to fulfill in order to be possibly correct. Again we have to come to the conclusion that we cannot create an automatic system that allows us to check if the ontology is correct – but merely a system that sends out warnings when something seems to be wrong. The constraints in turn may often require expressivity beyond what OWL offers.

We address problems in ontology engineering and maintenance that arose during the work with ontologies within the case studies in the European FP6 project SEKT.[1] As they often reminded us of problems that occurred in software engineering, a solution that was successfully introduced to software engineering was examined – *unit testing*. Although the notion of unit testing needed to be adapted for ontologies, it inspired a slew of possible approaches. Section 9.1 will show how unit testing for ontologies can be applied.

In Section 9.2 we discuss how further expressivity that goes well beyond the available standardized languages can be used in order to guarantee that the evaluated ontologies fulfill certain, formalized properties. Since these semantics cannot be expressed within OWL there have to be necessarily regarded as contextual in the sense of our definition, i.e. as extra-ontological artifacts that can be used for the evaluation of the ontology.

## 9.1 Unit tests

In the SEKT project one of the case studies aimed at providing an intelligent FAQ system to help newly appointed judges in Spain (Benjamins *et al.*, 2005). The system depends on an ontology for finding the best answers and to find references to existing cases in order to provide the judge with further background information. The applied ontology is built and maintained by legal experts with almost no experience in formal knowledge representation (Casanovas *et al.*, 2005).

As the ontology evolved and got refined (and thus changed), the legal experts noticed that some of their changes had undesired side effects. To give a simplified example, consider the class hierarchy depicted in Figure 9.1. Let's assume that this ontology has been used for a while already, before someone notices that not every academic needs necessarily be a member of an university. So `Academic` becomes a direct subclass of `Person`, instead of `University_member`. But due to this change, also `Professor` is no subclass of `University_member` anymore (a change that maybe was hidden from the

[1] `http://cordis.europa.eu/ist/kct/sekt_synopsis.htm`

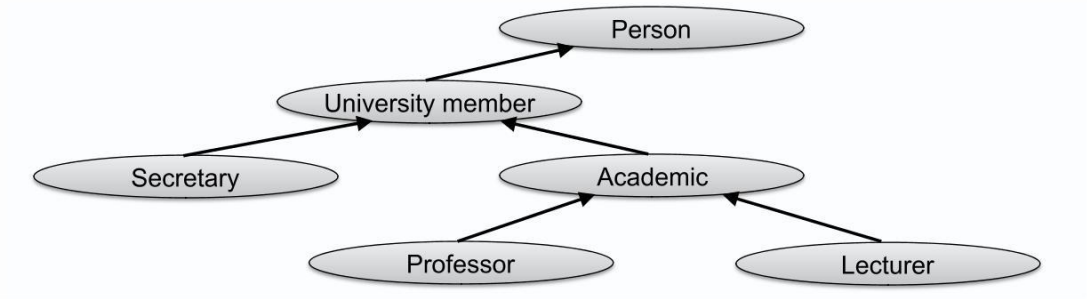


Figure 9.1: Example class hierarchy.

ontology engineer, as the ontology development environment may not have displayed all subclasses of `Academic`).

The resulting ontology remains perfectly satisfiable. But a tool, that, for example, creates a Web page for all members of the university may now skip the professors, since they are not classified as university members any more – an error that would only become apparent in the use of the tool much later and will be potentially hard to track down to that particular ontology change operation. Unit testing for ontologies can discover such problems, and a few other ones as well.

In software engineering, the idea of unit testing (Beck, 1999) was introduced to counter the complexities of modern software engineering efforts. Unit tests are meant to facilitate the development of program modules or units, and to ensure the interplay of such units in the combined system. It results in code that is easier to refactor and simpler to integrate, and that has a formalized documentation (although not necessarily complete). Unit tests can be added incrementally during the maintenance of a piece of software, in order to not accidentally stumble upon an old bug and hunt it down repeatedly. Unit tests in software engineering became popular with the object oriented language Smalltalk, and still to this day remain focused on languages with strong possibilities to create smaller units of code. They are based on several decomposition techniques, most important of all information hiding.

Ontologies behave quite differently than program units. As there is no notion of information hiding in ontology engineering, and thus no black box components, at first the idea of unit testing for ontologies seems not applicable. Therefore we need to adapt the idea for ontologies.

The main purpose of unit tests for ontologies is similar to their purpose in software engineering: whenever an error is encountered with an axiom which is falsely inferred or respectively incorrectly not inferred, the ontology maintainer may add this piece of knowledge to the appropriate test ontology. Whenever the ontology is changed, the changed ontology can be automatically checked against the test ontology, containing

the formalized knowledge of previously encountered errors.

We investigate the benefits of unit testing applied to ontologies, especially their possibilities to facilitate regression tests, to provide a test framework that can grow incrementally during the maintenance and evolution phase of the ontology, and that is reasonably simple to use. In order for unit tests for ontologies to be useful, they need to be reasonably easy to use and maintain. This will depend heavily on the given implementation. The following approach is informed by the idea of *design by contract*, i.e. we enable to formalize what statements should and should not derive from an ontology being developed or maintained, either as formalized competency questions (Section 9.1.1) or as explicit ontology statements (Sections 9.1.2 and 9.1.3).

### 9.1.1 Formalized competency questions

Competency questions, as defined by some methodologies for ontology engineering (such as OTK (Sure and Studer, 2002) or Methontology (Fernández-López *et al.*, 1999)), describe what kind of knowledge the resulting ontology is supposed to answer. These questions are necessarily formalizable in a query language (since otherwise the ontology management module would actually not be able to give the answer to the system later). Formalizing the queries instead of writing them down in natural language, and formalizing the expected answers as well allows for a system to automatically check if the ontology meets the requirements stated with the competency questions.

**Method 19 (Checking competency questions against results)**
Formalize your competency question as a SPARQL query. Write down the expected answer as a SPARQL query result, either in XML (Beckett and Broekstra, 2008) or in JSON (Clark *et al.*, 2007). Compare the actual and the expected results. Note that the order of results is often undefined.

This approach is especially interesting when the expressivity of the query language is outside of the expressivity of the knowledge representation language. This is, for example, the case with SPARQL and OWL. The following SPARQL query for example returns all instances of `Good_mother`, i.e. those mothers that are also friends with their child (we omit namespace declarations for readability). Since there are no property intersections in OWL, one cannot describe a class including all instances of `Good_mother` in OWL.

```
SELECT ?Good_mother
WHERE {
  ?child :mother ?Good_mother .
```

```
  ?child :friend ?Good_mother .
}
```

We consider this method especially useful not for the maintenance of the system, but rather for its initial build, in order to define the extent of the ontology. Note that competency questions usually are just exemplary questions – answering all competency questions does not mean that the ontology is complete. Also note that sometimes, although the question is formalizable, the answer does not necessarily need to be known at the time of writing the question. This is especially true for dynamic ontologies, i.e. ontologies that reflect properties of the world that keep changing often (like the song the user of the system is listening to at query time). In that case we can define some checks if the answer is sensible or even possible (like that the answer indeed needs to be a song).

How can we test such constraints? Instead of using a SPARQL `SELECT` query, we can use a SPARQL `CONSTRUCT` query to create a new ontology with the given results (again, namespace declarations are omitted):

```
CONSTRUCT { ?Good_mother rdf:type :Good_mother }
WHERE {
  ?child :mother ?Good_mother .
  ?child :friend ?Good_mother .
}
```

This will result in an ontology that consists only of class instantiations for the `Good_mother` class. We can now merge the ontology resulting from the SPARQL `CONSTRUCT` query with the background ontology (usually the ontology used for query answering) and a constraint ontology that includes constraints on the results, like the following.

```
DisjointClasses(Good_mother Father)
```

The ontology states that fathers cannot be good mothers. Now should one of the results actually be an instance of father, the resulting merged ontology will be inconsistent.

**Method 20 (Checking competency questions with constraints)**
Formalize your competency question for ontology $O$ as a SPARQL `CONSTRUCT` query that formulates the result in RDF as ontology $R$. Merge $R$ with $O$ and a possibly empty ontology containing further constraints $C$. Check the merged ontology for inconsistencies.

### 9.1.2 Affirming derived knowledge

*Unit tests* for ontologies test if certain axioms can or can not be derived from the ontology (Vrandečić and Gangemi, 2006). This is especially useful in the case of evolving or dynamic ontologies: we can automatically test certain assumptions with regards to a given ontology $O$.

We create two test ontologies $T^+$ (called the *positive test ontology*) and $T^-$ (the *negative test ontology*), and define that an ontology $O$, in order to fulfill the constraints imposed by the test ontologies, needs to fulfill the following conditions: each axiom $A_1^+ ... A_n^+ \in T^+$ must be derivable from $O$, i.e.

$$O \models A_i^+ \quad \forall A_i^+ \in T^+$$

and each axiom $A_1^-, \ldots, A_n^- \in T^-$ must not be derivable from $O$, i.e.

$$O \not\models A_i^- \quad \forall A_i^- \in T^-$$

Note that $T^+$ trivially fulfills the first condition if $O$ is not satisfiable, whereas an empty ontology trivially fulfills the second condition. So it is not hard to come up with ontologies that fulfill the conditions, which shows that unit tests are not meant to be complete formalizations of the requirements of an ontology, but rather helpful indicators towards possible errors or omissions in the tested ontologies.

To come back to our previous example in Section 9.1 a simple test ontology $T^+$ that consists of the single axiom `SubClassOf(Professor University_member)` would have been sufficient to discover the problem described. So after the discovered error, this statement is added to the test ontology, and now this same error will be detected next time automatically by running the unit tests.

The test ontologies are meant to be created and grown during the maintenance of the ontology. Every time an error is encountered in the usage of the ontology, the error is formalized and added to the appropriate ontology (like in the example above). Experienced ontology engineers may add appropriate axioms in order to anticipate and counter possible errors in maintenance.

In software engineering it is often the case, that the initial development of a program is done by a higher skilled, better trained, and more consistent team, whereas the maintenance is then performed by a less expensive group, with less experienced members, that change more frequently. So in software engineering, the more experienced developers often anticipate frequent errors that can happen during maintenance, and create unit tests accordingly in order to put appropriate constraints on the future evolution of the software. We expect a similar development in ontology engineering and maintenance, as soon as ontologies become more common components of information systems. The framework proposed here offers the same possibilities to an ontology engineer.

Why should an ontology engineer not just add the axioms from $T^+$ to $O$, and $\neg A_i^-$ for each $A_i^-$ in $T^-$? There are several reasons:

1. not every axiom $A_i^-$ can be negated. For example, a subsumption statement cannot be negated without inventing new entities.

2. adding such axioms increases redundancy in the ontology, and thus makes it harder to edit and maintain.

3. the axioms may potentially increase reasoning complexity, or else use language constructs that are not meant to be used within the ontology.

4. as discussed in Section 9.1.3, the axioms in $T^-$ may be contradictory.

5. finally, due to the open world assumption, $O \not\models A_i^- \forall A_i^- \in T^-$ is not the same as $O \models \neg A_i^- \forall A_i^- \in T^-$, so that the negative test ontology can actually not be simulated with the means of OWL DL.

A Protégé plug-in implementing an OWL Unit Test framework[2] exists, that allows to perform what we have described with $T^+$ testing for affirming derived knowledge (Horridge, 2005). We generalize and expand the theoretical foundations of the framework.

**Method 21 (Unit testing with test ontologies)**
For each axiom $A_i^+$ in the positive test ontology $T^+$ test if the axiom is being inferred by the tested ontology $O$. For every axiom that is not being inferred, issue an error message.

For each axiom $A_i^-$ in the negative test ontology $T^-$ test if the axiom is being inferred by the tested ontology $O$. For every axiom that is being inferred, issue an error message.

Formalized competency questions and positive unit test ontologies can sometimes be translated from one into the other, but are largely complementary. We already have shown that SPARQL enables queries beyond the expressivity of OWL, and also test ontologies are much more natural to express OWL constructs than SPARQL is (compare with Section 6.2). Finally, the notion of negative test ontologies expand the possibilities of unit testing well beyond formalized competency questions.

[2] http://www.co-ode.org/downloads/owlunittest/

### 9.1.3 Asserting agnosticism

Whereas ontologies in general should be consistent in order to be considered useful, this is not true for the negative test ontology $T^-$. Since $T^-$ is simply a collection of all axioms that should not be inferrable from the tested ontology $O$, it does not need to be satisfiable. Thus they may be two (or more) sets of axioms (subsets of $T^-$) that contradict each other. What does this mean? The existence of such contradicting sets mean that $O$ must not make a decision about the truth of either of these sets, thus formalizing the requirement that $O$ must be *agnostic* towards certain statements.

For example an ontology of complexity classes (where each complexity class is an individual) should have the following two axioms in its negative test ontology:

```
DifferentIndividuals(P NP)
SameIndividual(P NP)
```

It is obvious that the test ontology is inconsistent, but what it states is that any tested ontology will neither allow to infer that `P` and `NP` are equal, nor that they are different. Every ontology that is successfully tested against this negative test ontology will be agnostic regarding this fact.

Besides known unknowns (such as the *P=NP?* problem) we can also assert agnosticism in order to preserve privacy. If we want to ensure that it cannot be inferred if `Alice` knows `Bob` from an ontology (i.e. a set of statements, possibly gathered from the Web), we can state that in a test ontology:

```
PropertyAssertion(foaf:knows Alice Bob)
NegativeProperyAssertion(foaf:knows Alice Bob)
```

Note that if such a test ontology is indeed used for privacy reasons, it should not be made public since knowing it may lead to important clues for the actual statements that were meant to remain secret.

## 9.2 Increasing expressivity for consistency checking

Certain language constructs, or their combination, may increase reasoning time considerably. In order to avoid this, ontologies are often kept simple. But instead of abstaining from the use of more complex constructs, an ontology could also be modularized with regards to its complexity: an ontology may come in different versions, one just defining the used vocabulary, and maybe their explicitly stated taxonomic relations, and a second ontology adding much more knowledge, such as disjointness or domain and range axioms. If the simple ontology is properly built, it can lead to an ontology which often yields the same results to queries as the complete ontology (depending, naturally, on the actual queries). The additional axioms can be used in

order to check the consistency of the ontology and the knowledge base with regards to the higher axiomatized version, but for querying them the simple ontology may suffice.

We investigate the relationship of heavy- to lightweight ontologies, and how they can interplay with regards to ontology evaluation in Section 9.2.1. We then move to an exemplary formalism going well beyond the expressivity of OWL, by adding rules to the ontology for evaluating it in Section 9.2.2. Autoepistemic operators lend themselves also to be used in the testing of ontologies, especially with regards to their (relative) completeness, since they are a great way to formalize the introspection of ontologies (Section 9.2.3). We also regard a common error in ontology modeling with description logics based languages, and try to turn this error into our favor in Section 9.2.4.

### 9.2.1 Expressive consistency checks

Ontologies in information systems often need to fulfill the requirement of allowing reasoners to quickly answer queries with regards to the ontology. Light weight ontologies usually fulfill this task best. Also, many of the more complex constructors of OWL DL often do not add further information, but rather are used to restrict possible models. This is useful in many applications, such as ontology mapping and alignment, or information integration from different sources.

For example, a minimal cardinality constraint will, due to the open world assumption, hardly ever lead to any inferred statements in an OWL DL ontology. Nevertheless the statement can be useful as an indicator for tools that want to offer a user interface to the ontology, or for mapping algorithms that can take this information into account.

Further expressive constraints on the ontology, such as disjointness of classes, can be used to check the ontology for consistency at the beginning of the usage, but after this has been checked, a light weight version of the ontology, that potentially enables reasoners to derive answers with a better response time, could be used instead.

Formally, we introduce a test ontology $C$ for an ontology $O$, including additional axioms of the entities used in $O$ that constrain the possible models of the ontology, and check for the satisfiability of the merged ontology $O \cup C$.

For example, consider the following ontology $O$ about a family:

```
PropertyAssertion(father Adam Seth)
PropertyAssertion(mother Eve Seth)
```

Now consider ontology $C$, formalizing further constraints on the terms used in the ontology:

```
SubPropertyOf(father parent)
SubPropertyOf(mother parent)
PropertyDomain(parent Person)
PropertyRange(father Male)
```

```
PropertyRange(mother Female)
DisjointClasses(Male Female)
```

Now, $O \cup C$ will be recognized as inconsistent by a reasoner. This is because of the definitions of the properties `father` and `mother` showing that it should point *to* the father respectively the mother, whereas in $O$ it is used to point *from* the father respectively the mother. This is recognized because $C$ adds range axioms on both, thus letting us infer that `Seth` has to be an instance of both `Female` and `Male`, which is not possible due to the `DisjointClasses` axiom in $C$.

**Method 22 (Increasing expressivity)**
An ontology $O$ can be accompanied by a highly axiomatized version of the ontology, $C$. The merged ontology of $O \cup C$ has to be consistent, otherwise the inconsistencies point out to errors in $O$.

### 9.2.2 Consistency checking with rules

The consistency checks with context ontologies do not need to be bound by the expressivity of OWL, but can instead be using languages with a different expressivity, such as SWRL (Horrocks *et al.*, 2003; Brockmans and Haase, 2006). For our example, we will use the transformation of the ontology to a logic programming language like datalog (Grosof *et al.*, 2003) as implemented by the OWL tools (Motik *et al.*, 2005) wrapping KAON2 (Motik, 2006). We can then add further integrity constraints to the resulting program. This may exceed the expressivity available in the original ontology language. Consider constraints on the `parent` relationship that state that a parent has to be born before the child. In datalog we can concatenate the translation of the ontology to datalog (i.e. $LP(O)$) with the constraint program $C$, and test the resulting program for violations of the integrity constraints. Note that $LP(O)$ cannot fully translate arbitrary complex ontologies $O$, but for our use case it is sufficient to only regard the resulting logic program. We can simply ignore the rest.

Consider (a repaired version of) the family ontology $O$ given in Section 9.2.1. We also add years of birth for `Adam` and `Seth`:

```
PropertyAssertion(father Seth Adam)
PropertyAssertion(mother Seth Eve)
PropertyAssertion(birthyear Adam "-3760"^^xsd:gYear)
PropertyAssertion(birthyear Seth "-3890"^^xsd:gYear)
```

Translating it to datalog will yield the following result $LP(O)$:

```
father(Seth, Adam)
mother(Seth, Eve)
birthyear(Adam, -3760)
birthyear(Seth, -3890)
```

We also translate the constraint ontology $C$ from Section 9.2.1 as $LP(C)$:

```
parent(X, Y) ← father(X, Y)
parent(X, Y) ← mother(X, Y)
Person(X) ← parent(X, Y)
Male(Y) ← father(X, Y)
Female(Y) ← mother(X, Y)
← Male(X) ∧ Female(X)
```

Now we regard further integrity constraints such as the following $R$ (< meaning before):

```
← parent(X, Y) ∧ birthyear(X, BX) ∧ birthyear(Y, BY) ∧ (BY < BX)
```

$R$ states that if the year of birth of the parent's child is before the year of birth of the parent, then we have an inconsistency. We can now concatenate $LP(O)$, $LP(C)$ and $R$ and check for inconsistencies – and indeed, one will be raised since `Adam` was born 130 years after `Seth`, and thus the ontology must be inconsistent.

**Method 23 (Inconsistency checks with rules)**
Translate the ontology to be evaluated and possible constraint ontologies to a logic program. This translation does not have to be complete. Formalize further constraints as rules or integrity constraints.

Concatenate the translated ontologies and the further constraints or integrity constraints. Run the resulting program. If it raises any integrity constraints, then the evaluated ontology contains errors.

### 9.2.3 Use of autoepistemic operators

One approach using increased expressivity is to describe what kind of information an ontology is supposed to have, using *autoepistemic constructs* such as the **K**- and **A**-operators (Grimm and Motik, 2005). In this way we can, for example, define that every person in our knowledge base needs to have a known name and an address, and

put this into a description. Note that this is different than just using the existential construct. An axiom like

```
SubClassOf(Human SomeValuesFrom(parent Human))
```

tells us that every human has a human parent, using the **K** operator we could require that every humans' parent has to be explicitly given in the knowledge base or else an inconsistency would be raised. Autoepistemic operators allow for a more semantic way to test the completeness of knowledge bases than the syntactic XML based schema definitions described in Section 5.3. This way the ontology can be evaluated with regards to its *data completeness*. Data completeness is defined with regards to a tool that uses the data. The tool needs to explicate which properties it expects when being confronted with an individual belonging to a certain class. The operators allow to map completeness checks to satisfiability checks, and use a common language to express these checks. Otherwise the completeness checks have to be checked programmatically by the tool.

In (Grimm and Motik, 2005) an extension of OWL DL with autoepistemic operators is described. The axiom above translated to DL would be

$\texttt{Human} \sqsubseteq \exists \texttt{parent}.\texttt{Human}$

Using the **K**- and **A**−operators instead, we would define that

$\mathbf{K}\texttt{Human} \sqsubseteq \exists \mathbf{A}\texttt{parent}.\mathbf{A}\texttt{Human}$

i.e. for every human a parent must be known who also must be known to be human (i.e. either stated explicitly or inferrable) in the ontology, or else the ontology will not be satisfiable.[3] Thus we are able to state what *should* be known, and a satisfiability check will check if, indeed this knowledge is present.

On the Semantic Web, such a formalism will prove of great value, as it allows to simply discard data that does not adhere to a certain understanding of completeness. For example, a crawler may gather event data on the Semantic Web. But instead of simply collecting all instances of event, it may decide to only accept events that have a start and an end date, a location, a contact email, and a classification with regards to a certain term hierarchy. Although this will decrease the recall of the crawler, the data will be of a higher quality, i.e. of a bigger value, as it can be sorted, displayed, and actually used by calendars, maps, and email clients in order to support the user.

The formalization and semantics of autoepistemic operators for the usage in Web ontologies is described in (Grimm and Motik, 2005).

[3] The example follows ex. 3.3 in (Donini *et al.*, 2002)

### 9.2.4 Domain and ranges as constraints

When novice ontology engineers have a background in programming, they often find the semantics of domain and range confusing (Allemang and Hendler, 2008). They expect domain and ranges to be used like type constraints in programming languages, i.e. if they say that `father` is a relation with the range `Male` and one applies it between `Seth` and `Eve`, who is a *Woman*, they expect an inconsistency to be raised. Instead, `Eve` will be classified as a *Man* by the reasoner (for the sake of the example no disjointness axiom is stated between `Male` and `Female`). When programming in a stronlgy typed language like Java or C++, if we had a function defined as `father(Male dad, Person child)`, calling the function with an object instance of `Female` will raise an exception (or, due to polymorphism, actually not find a suitable function. This assumes that `Female` is not a subclass of `Male`).

In order to simulate a mechanism similar to the expectations of programmers, we need to introduce two new axiom types, `PropertyDomainConstraint` to describe a constraint on a domain of a property, and `PropertyRangeConstraint` for the range respectively. Besides their name they have the same syntax as the `PropertyDomain` respective `PropertyRange` axioms. Using autoepistemic logic, the axiom

```
PropertyDomainConstraint(R C)
```

translates to the following semantics

$$\exists \mathbf{K}\mathtt{R}.\top \sqsubseteq \mathbf{A}\mathtt{C}$$

stating that everything that is known to be in the domain of `R` has to be a known instance of `C`.

Instead of using a new axiom type, we could also use the fact that ontology engineers often add domain and range axioms erroneously, not in order to add more inferences but with the meaning intended by the new axiom types, i.e. as type constraints. Based on that we suggest to consciously misinterpret the semantics of the already existing domain and range axioms for the sake of ontology evaluation (note, that this explicitly is not meant as an reinterpretation for using the ontology, but merely for evaluating it before usage). The evaluator will quickly figure out if this is a useful misinterpretation for a given ontology or not. We checked both the Watson EA and the Watson 130 corpora (see Section 11.3) and we did not find a single case where the domain axiom of a property added an inferred class assertion to an individual using that property with the individual not being already explicitly asserted to be an instance of that class. This indicates that this approach could indeed provide fruitful.

# Part III

# Application

10 Collaborative ontology evaluation in Semantic MediaWiki 167

11 Related work 185

12 Conclusions 197

## Chapter 10

# Collaborative ontology evaluation in Semantic MediaWiki

> Given enough eyeballs,
> all bugs are shallow.
>
> *(Eric Raymond, b. 1957, The Cathedral and the Bazaar (Raymond, 2001))*

Wikis have become popular tools for collaboration on the Web, and many vibrant online communities employ wikis to exchange knowledge. For a majority of wikis, public or not, primary goals are to organize the collected knowledge and to share information. Wikis are tools to manage online content in a quick and easy way, by editing some simple syntax known as wikitext. This is mainly plain text with some occasional markup elements. For example, a link to another page is created by enclosing the name of the page in brackets, e.g. by writing `[[Semantic Web]]`.

But in spite of their utility, the content in wikis is barely machine-accessible and only weakly structured. In this chapter we introduce *Semantic MediaWiki* (SMW) (Krötzsch *et al.*, 2007c), an extension to MediaWiki (Barret, 2008), a widely used wiki software. SMW enhances MediaWiki by enabling users to annotate the wiki's contents with explicit information. Using this semantic data, SMW addresses core problems of today's wikis:

- **Consistency of content**: The same information often occurs on many pages. How can one ensure that information in different parts of the system is consistent, especially as it can be changed in a distributed way?
- **Accessing knowledge**: Large wikis have thousands of pages. Finding and *comparing* information from different pages is challenging and time-consuming.

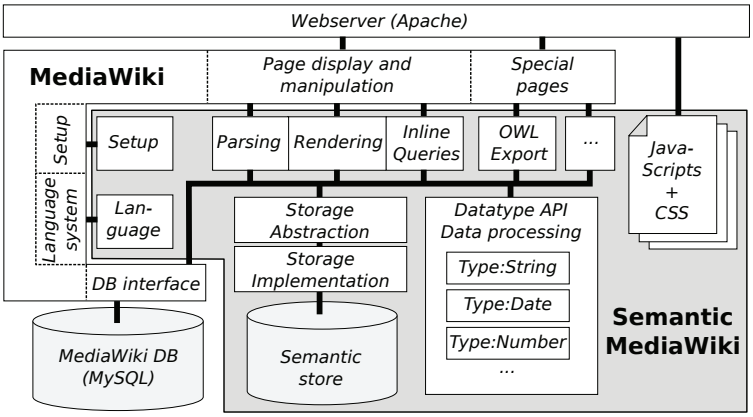


Figure 10.1: Architecture of SMW's main components in relation to MediaWiki.

- **Reusing knowledge**: Many wikis are driven by the wish to make information accessible to many people. But the rigid, text-based content of classical wikis can only be used by reading pages in a browser or similar application.

SMW is a free and open source extension of MediaWiki, released under the GNU Public License. Figure 10.1 provides an overview of SMW's core components and architecture. The integration between MediaWiki and SMW is based on MediaWiki's extension mechanism: SMW registers for certain events or requests, and MediaWiki calls SMW functions when needed. SMW thus does not overwrite any part of MediaWiki, and can be added to existing wikis without much migration cost. Usage information about SMW, installation instructions, and the complete documentation are found at SMW's homepage.[1]

Section 10.1 explains how structural information is collected in SMW, and how this data relates to the OWL ontology language. Section 10.2 surveys SMW's main features for wiki users: semantic browsing, semantic queries, and data exchange on the Semantic Web. Queries are the most powerful way of retrieving data from SMW, and their syntax and semantics is presented in detail. In Section 10.3 we survey related systems. Based on our definition of ontology, it is clear that SMW is an ontology engineering tool that aims at a massively collaborative usage. Based on that, Section 10.4 discusses how ontology evaluation can be performed collaboratively within SMW using some of the approaches introduced in Part II of this thesis.

[1] http://semantic-mediawiki.org

## 10.1 Annotation of wiki pages

The main prerequisite of exploiting semantic technologies is the availability of suitably structured data. For this purpose, SMW introduces ways of adding further structure to MediaWiki by means of *annotating* the textual content of the wiki. In this section, we recall some of MediaWiki's current means of structuring data (Section 10.1.1), and introduce SMW's annotations with properties (Section 10.1.2). Finally, a formal semantic interpretation of the wiki's structure in terms of OWL is presented (Section 10.1.3).

### 10.1.1 Content structuring in MediaWiki

The primary method for entering information into a wiki is *wikitext*, a simplified markup language that is transformed into HTML pages for reading. Accordingly, wikitext already provides many facilities for describing formatting, and even some for structuring content. For defining the interrelation of pages within a wiki, hyperlinks are arguably the most important feature. They are vital for navigation, and are sometimes even used to classify articles informally. In Wikipedia, for example, articles may contain links to pages of the form `[[as of 2010]]` to state that the given information might need revalidation or updates after that year.

The primary structural mechanism of most wikis is the organization of content in wiki pages. In MediaWiki, these pages are further classified into *namespaces*, which distinguish different kinds of pages according to their function. Namespaces cannot be defined by wiki users, but are part of the configuration settings of a site. A page's namespace is signified by a specific prefix, such as `User:` for user homepages, `Help:` for documentation pages, or `Talk:` for discussion pages on articles in the main namespace. Page titles without a known namespace prefix simply belong to the main namespace. Most pages are subject to the same kind of technical processing for reading and editing, denoted *Page display and manipulation* in Figure 10.1. The major exception are so-called *special pages* – built-in query forms without user-edited content – that use `Special:` as a namespace prefix.

Many wiki engines generally use links for classifying pages. For instance, searching for all pages with a link to the page `[[France]]` is a good way to find information about that country. In MediaWiki, however, this use has been replaced by a more elaborate category system (Schindler and Vrandečić, 2010). Every page can be assigned to one or many categories, and each category is represented with a page in the `Category:` namespace. Category pages in turn can be used to browse the classified pages, and also to organize categories hierarchically. Page categories and their hierarchy can be edited by all users via special markup within the wiki. Overall, the category system probably is the one function of MediaWiki that is closest in spirit to the extensions of SMW.

Another structuring problem of large wikis are synonymous and homonymous titles. In case of synonyms, several different pages for the same subject may emerge in a decentralized editing process. MediaWiki therefore has a *redirect* mechanism by which a page can be caused to forward all requests directly to another page. This is useful to resolve synonyms but also for some other tasks that suggest such forwarding (e.g. the mentioned articles `[[as of 2005]]` are redirects to the page about the year 2005). Homonyms in turn occur whenever a page title is ambiguous, and may refer to many different subjects depending on context. This problem is addressed by so-called *disambiguation pages* that briefly list the different possible meanings of a title. Actual pages about a single sense then either use an unique synonym or are augmented with parentheses to distinguish them, e.g. in the case of `[[1984 (book)]]`.

A final formatting feature of significance to the structure of the wiki is MediaWiki's *template system*. The wiki parser replaces templates with the text given on the template's own page. The template text in turn may contain parameters. This can be used to achieve a higher consistency, since, for example, a table is then defined only once, and so all pages using this table will look similar. The idea of capturing semantic data in templates has been explored inside Wikipedia[2] and in external projects such as DBpedia (Auer and Lehmann, 2007).

In addition to the above, MediaWiki knows many ways of structuring the textual content of pages themselves, e.g. by sections or tables, presentation markup (e.g. text size or font weights), etc. SMW, however, aims at collecting information about the (abstract) concept represented by a page, not about the associated text. The layout and structure of article texts is not used for collecting semantic annotations, since they should follow didactic considerations.

### 10.1.2 Semantic annotations in SMW

Adhering to MediaWiki's basic principles, semantic data in SMW is also structured by pages, such that all semantic content explicitly belongs to a page. Using the terms from Section 2.3, every page corresponds to an ontology entity (including classes and properties). This locality is crucial for maintenance: if knowledge is reused in many places, users must still be able to understand where the information originated. Different namespaces are used to distinguish the different kinds of ontology entities: they can be individuals (the majority of the pages, describing elements of the domain of interest), classes (represented by *categories* in MediaWiki, used to classify individuals and also to create subcategories), properties (relationships between two individuals or an individual and a data value), and *types* (used to distinguish different kinds of properties). Categories have been available in MediaWiki since 2004, whereas properties and types were introduced by SMW.

[2] See, e.g., `http://de.wikipedia.org/wiki/Hilfe:Personendaten`.

```
'''London''' is the capital city of [[England]] and of the [[United Kingdom]].  As of
[[2005]], the population of London was estimated 7,421,328.  Greater London covers an
area of 609 square miles.                                          [[Category:City]]
```

```
'''London''' is the capital city of [[capital of::England]] and of the [[capital of::United
Kingdom]]. As of [[2005]], the population of London was estimated [[population::7,421,328]].
Greater London covers an area of [[area::609 square miles]].       [[Category:City]]
```

Figure 10.2: Source of a page about London in MediaWiki (top) and in SMW (bottom).

SMW collects semantic data by letting users add annotations to the wiki source text of pages via a special markup. The processing of this markup is performed by the components for *parsing* and *rendering* in Figure 10.1. While the annotation syntax is most relevant (and most visible) to wiki editors, it is but a small part of the overall SMW system. The underlying conceptual framework, based on *properties* and *types* is rather more relevant. We also expect the annotation syntax to become more hidden, as exemplified by a number of extensions on top of SMW.

Properties in SMW are used to express binary relationships between one individual (as represented by a wiki page) and some other individual or data value. Each wiki-community is interested in different relationships depending on its topic area, and therefore SMW lets wiki users control the set of available properties. SMW's property mechanism follows standard Semantic Web formalisms where binary properties also are a central expressive mechanism. But unlike RDF-based languages, SMW does not view property statements (subject-predicate-object triples) as primary information units. SMW rather adopts a page-centric perspective where properties are a means of augmenting a page's contents in a structured way.

MediaWiki offers no general mechanism for assigning property values to pages, and a surprising amount of additional data becomes available by making binary relationships in existing wikis explicit. The most obvious kind of binary relations in current wikis are hyperlinks. Each link establishes *some* relationship between two pages, without specifying what kind of relationship this is, or whether it is significant for a given purpose. SMW allows links to be characterized by properties, such that the link's target becomes the value of a user-provided property. But not all properties take other wiki pages as values: numeric quantities, calendar dates, or geographic coordinates are examples of other available types of properties.

For example, consider the wikitext shown in Figure 10.2 (top). The markup elements are easy to read: triple quotes `'''...'''` are used for text that should appear bold-faced, and text within square brackets `[[...]]` is transformed into links to the wiki page of that name. The given links to `[[England]]`, `[[United Kingdom]]`, and `[[2005]]` do not carry any machine-understandable semantics yet. To state that Lon-


Figure 10.3: A semantic view of London.

don is the capital of England, one just extends the link to `[[England]]` by writing `[[capital of::England]]`. This asserts that `London` has a property called `capital of` with the value `England`. This is even possible if the property `capital of` has not been introduced to the wiki before.

Figure 10.2 (top) shows further interesting data values that are not corresponding to hyperlinks, e.g. the given population number. A syntax for annotating such values is not as straightforward as for hyperlinks, but we eventually decided for using the same markup in both cases. An annotation for the population number therefore could be added by writing `[[population::7,421,328]]`. In this case, `7,421,328` is not referring to another page and we do not want our statement to be rendered as a hyperlink. To accomplish this, users must first *declare* the property `population` and specify that it is of a numerical type. This mechanism is described below. If a property is not declared yet, then SMW assumes that its values denote wiki pages such that annotations will become hyperlinks. An annotated version of the wikitext for `London` is shown in Figure 10.2 (bottom), and the resulting page is displayed in Figure 10.3.

Properties are introduced to the wiki by just using them on some page, but it is often desirable to specify additional information *about* properties. SMW supports this by introducing wiki pages for properties. For example, a wiki might contain a page `[[Property:Population]]` where `Property:` is the namespace prefix. A property page can contain a textual description of the property that helps users to employ it consistently throughout the wiki, but it also can specify semantic features of a property. One such feature is the aforementioned *(data)type* of the property. In the case of `[[Property:Population]]` one would add the annotation `[[has type::Number]]` to describe that the property expects numerical values. The property `has type` is a built-

in property of SMW with the given special interpretation. It can also be described on its property page but it cannot be modified or deleted.

SMW provides a number of datatypes that can be used with properties. Among those are *String* (character sequences), *Date* (points in time), *Geographic coordinate* (locations on earth), and the default type *Page* that creates links to other pages. Each type provides own methods to process user input, and to display data values. SMW supplies a modular *Datatype API* as shown in Figure 10.1 that can also be extended by application-specific datatypes. Just like properties, types also have dedicated pages within the wiki, and every type declaration creates a link to the according page. To some extent, it is also possible to create new customized datatypes by creating new type pages. These pages of course cannot define the whole computational processing of a data value, but they can create parameterized versions of existing types. The main application of this is to endow numerical types with conversion support for specific units of measurement. For example, the property `Area` in Figure 10.2 (bottom) might use a custom type that supports the conversion between $km^2$ and square miles (as can be seen in Figure 10.3). Unit conversion is of great value for consolidating annotations that use different units, which can hardly be avoided in a larger wiki.

### 10.1.3 Mapping to OWL

The formal semantics of annotations in SMW, as well as their mapping for the later export (see Section 10.2.3) is given via a mapping to the OWL ontology language. Most annotations can easily be exported in terms of OWL, using the obvious mapping from wiki pages to OWL entities: normal pages correspond to individuals, properties in SMW correspond to OWL properties, categories correspond to OWL classes, and property values can be abstract individuals or typed literals. Most annotations thus are directly mapped to simple OWL statements.

OWL further distinguishes object properties, datatype properties, and annotation properties. SMW properties may represent any of those depending on their type. Types themselves do not have OWL semantics, but may decide upon the XML Schema type used for literal values of a datatype property. Finally, containment of pages in MediaWiki's categories is interpreted as class membership in OWL.

SMW offers a number of built-in properties that may also have a special semantic interpretation. The above property `has type`, for instance, has no equivalent in OWL and is interpreted as an annotation property. Many properties that provide SMW-specific meta-information (e.g. for unit conversion) are treated similarly. MediaWiki supports the hierarchical organisation of categories, and SMW can be configured to interpret this as an OWL class hierarchy (this may not be desirable for all wikis (Voss, 2006)). Moreover, SMW introduces a special property `subproperty of` that can be used for property hierarchies. Overall, the schematic information representable in SMW is intentionally shallow, since the wiki is not intended as a general purpose


Figure 10.4: Inverse search in SMW, here giving a list of everyone born in London.

ontology editor that requires users to have specific knowledge about semantic technologies.

## 10.2 Exploiting semantics

However simple the process of semantic annotation may be, the majority of users will neglect it as long as it does not bear immediate benefits. In the following we introduce several features of SMW that show contributors the usefulness of semantic markup.

### 10.2.1 Browsing

As shown in Figure 10.3 the rendered page may include a so called *factbox* which is placed at the bottom of the page to avoid disturbing normal reading. The factbox summarizes the given annotations, provides feedback on possible errors, e.g. if a given data value does not fit a property's type, and offers links to related functions. Note that most SMW instances do not display the factbox but rather choose to customize the users experience by using inline queries to display the semantic data.

These links can be used to browse the wiki based on its semantic content. The page title in the factbox heading leads to a *semantic browsing* interface that shows not only the annotations within the given page, but also all annotations where the given page

is used as a value. The magnifier icon behind each value leads to an *inverse search* for all pages with similar annotations (Figure 10.4). Both of those user interfaces are realized as *special pages*, architecturally similar to the special page *OWL Export* in Figure 10.1. In addition, the factbox shows links to property pages, which in turn list all annotations for a given property. All those browsing features are interconnected by appropriate links, so that users can easily navigate within the semantic knowledge base.

### 10.2.2 Querying

SMW includes a query language that allows access to the wiki's knowledge. The query language can be used in three ways: either to directly query the wiki via a special query page, to add the answer to a page by creating an *inline query* (Figure 10.1), or by using *concepts*.

**Inline queries** enable editors to add dynamically created lists or tables to a page, thus making up-to-date query results available to readers who are not even aware of the semantic capabilities of the underlying system. Figure 10.6 shows a query result as it might appear within an article about Switzerland. Compared to manually edited listings, inline queries are more accurate, easier to create, and easier to maintain.

**Concepts** are the intensional counterparts to MediaWiki's extensional categories. They occupy a new namespace (`Concept:`) and allow there to define a query that describes the class. Individual pages can not be tagged explicitly with a concept, instead an individual instantiates a concept implicitly by fulfilling the query description. This allows to define concepts such as `ISWC Conference` as

```
[[Category:Conference]] [[series::ISWC]]
```

All conferences that are properly annotated will then automatically be recognized as ISWC conferences. Concepts can be used in queries just as normal categories, and allow a higher abstraction than categories do.

The syntax of *SMW's query language* is closely related to wiki text, whereas its semantics corresponds to specific class expressions in OWL.[3] Each query is a disjunction of conjunctions of conditions. Fundamental conditions are encoded as *query atoms* whose syntax are similar to that of SMW's annotations. For instance,

```
[[located in::England]]
```

is the atomic query for all pages with this annotation. Queries with other types of properties and category memberships are constructed following the same principle. Instead of single fixed values one can also specify ranges of values, and even specify nested query expressions.

A simplified form of SMW's query language is defined in Figure 10.5 (top). The main control symbols used to structure queries are: `OR` and `||` as the disjunction

[3] SMW's query language has never been officially named, but some refer to it as **AskQL** (Ell, 2009)

```
QUERY ::= CONJ ('OR' CONJ)*
 CONJ ::= ATOM (ATOM)*
 ATOM ::= SUB | PROP | CAT | PAGE
  SUB ::= '<q>' QUERY '</q>'
 PROP ::= '[[' TITLE '::'  VALUE ('||' VALUE)* ']]'
VALUE ::= '+' | SUB | (('>'|'<'|'!')?  STR )
  CAT ::= '[[Category:'  TITLE ('||' TITLE)* ']]'
 PAGE ::= '[[:'  FULLTITLE ('||' FULLTITLE)* ']]'
```

```
QUERY ::= 'UnionOf(' CONJ (CONJ)* ')'
 CONJ ::= 'IntersectionOf(' ATOM (ATOM)* ')'
 ATOM ::= SUB | PROP | CAT | PAGE
  SUB ::= QUERY
 PROP ::= 'SomeValuesFrom(' TITLE ' UnionOf(' VALUE (VALUE)* '))'
VALUE ::= 'owl:Thing' | SUB | (>=|<=|!=)?  STR ')'
  CAT ::= 'UnionOf(' TITLE (TITLE)* ')'
 PAGE ::= 'OneOf(' FULLTITLE (FULLTITLE)* ')'
```

Figure 10.5: Production rules for SMW queries (top) and according OWL descriptions (bottom).

operators (depending on the context), `<q>` and `</q>` as (sub)query delimiters, `+` as the empty condition that matches everything, and `<`, `>`, and `!` to express comparison operators $\leq$, $\geq$, and $\neq$ (note that the given grammar assumes a way of expressing these comparison on literal values directly in OWL, whereas they actually need to be defined using appropriate concrete domain definitions). Some nonterminals in Figure 10.5 are not defined in the grammar: `TITLE` is for page titles, `FULLTITLE` is for page titles with namespace prefix, and `STR` is for Unicode strings. In those, we do not permit symbols that could be confused with other parts of the query, e.g. page titles must not start with `<`. SMW provides means to escape such characters.

The following is an example query for all cities that are located in an EU-country or that have more than 500,000 inhabitants:

```
[[Category:City]]
<q>
    [[located in::<q>[[Category:Country]] [[member of::EU]]</q>]]
  ||
    [[population:: >500,000]]
</q>
```

The formal semantics of such queries is given by a mapping to class descriptions in OWL, i.e. a query retrieves all inferred members of the according OWL class. It is

not hard to see that every SMW query emerges from a unique sequence of production steps, and we can follow the same steps in the grammar in the lower part of Figure 10.5. The result is a complex class description. The given example is translated to:

```
 1  UnionOf(
 2    IntersectionOf(
 3      UnionOf(City)
 4      UnionOf(
 5        SomeValuesFrom(located_in
 6          UnionOf(
 7            IntersectionOf(
 8              UnionOf(Country)
 9              SomeValuesFrom(member_of OneOf(EU)))))
10        SomeValuesFrom(population UnionOf(>=500,000)))))
```

The result is a syntactic form that is close to the functional style syntax used in this thesis, but which is not fully correct OWL yet. It is not hard to obtain a correct OWL class description from this expression. First, the identifiers used here still correspond to names of wiki entities – for a formal interpretation, they must be replaced by proper URIs. Second, the `UnionOf` descriptions in the lines 1, 3, 6, 8, and 13 contain only a single class description, and are thus not useful (and not allowed in OWL). One can easily remove `UnionOf` in such cases. Third, the pseudo expression `(>=500,000)` must be transformed into a valid OWL data range. This can be done by using OWL's `DatatypeRestriction` feature. In the case of object properties (SMW properties of type *Page*), `OneOf` can be used instead. Moreover, the exact shape of the relevant description also depends on the datatype used for a given data property, since restrictions like "greater or equal" are realized by suitable facets of the according XML Schema datatype (Motik *et al.*, 2009b). We omit this easy but tedious description of the formal details here.

Just like OWL, SMW's query language does not support explicit variables, which essentially disallows cross-references between parts of the query. This ensures that all queries are tree-like. For instance, it is not possible to ask for all people who died in the city they were born in. This restriction makes query answering tractable, which is essential for SMW's usage in large wikis. In contrast, when variables are allowed querying is at least NP-hard, and it becomes harder still even for tractable fragments of OWL 2 (Krötzsch *et al.*, 2007a).

SMW queries as introduced above merely define a result set of pages. In order to retrieve more information about those results, SMW allows so-called *print requests* as parts of queries. For instance, adding `?has capital` as a query parameter will cause all values of the property `has capital` to be displayed for each result. Figure 10.6 shows

| | Capital | Population | Languages |
|---|---|---|---|
| Aargau | Aarau | 573,654 | German language |
| Uri | Altdorf, Switzerland | 35,000 | German language |
| Appenzell Innerrhoden | Appenzell (town) | 15,000 | German language |
| Basel-City | Basel | 186,700 | German language |
| Ticino | Bellinzona | 319,800 | |
| Canton of Berne | Berne | 947,100 | |
| Graubünden | Chur | | German language<br>Italian language<br>Romansh |
| Canton of Jura | Delémont | 69,100 | French language |

Figure 10.6: A semantic query for all cantons of Switzerland, together with their capital, population, and languages. The data stems from an automatically annotated version of Wikipedia.

a typical output for a query with multiple print requests. Using further parameters in query invocation, result formatting can be controlled to a large degree. In addition to tabular output, SMW also supports various types of lists and enumerations, interactive timelines using SIMILE's timeline code,[4] and many further custom formats.

### 10.2.3 Giving back to the Web

The Semantic Web is all about exchanging and reusing knowledge, facilitated by standard formats that enable the interchange of structural information between producers and consumers. Section 10.1.3 explained how SMW's content is grounded in OWL, and this data can also be retrieved via SMW's Web interface as an *OWL export*. As shown in Figure 10.1, this service is implemented as a special page. It can be queried for information about certain elements. The link *RDF feed* within each factbox also leads to this service (see Figure 10.3).

Exported data is provided in OWL/RDF encoding, using appropriate URIs as identifiers to prevent confusion with URLs of the wiki's HTML documents. The semantic data is not meant to describe the HTML-document but rather its (intended) subject. The generated OWL/RDF is "browseable" in the sense that URIs can be used to locate further resources, thus fulfilling the linked data principles (as described in Section 4.1.1). All URIs point to a Web service of the wiki that uses content negotiation to redirect callers either to the OWL export service or to the according wiki page. Together with the compatibility to both OWL and RDF this enables a maximal reuse

[4] http://simile.mit.edu/timeline/

of SMW's data. Tools such as Tabulator (Berners-Lee *et al.*, 2006a) that incrementally retrieve RDF resources during browsing can easily retrieve additional semantic data on user request. SMW furthermore provides scripts for generating the complete export of all data within the wiki, which is useful for tools that are not tailored towards online operation such as the faceted browser Longwell.[5] Sample files of such export are found at `http://semanticweb.org/RDF/`.

SMW makes sure to generate valid URIs for all entities within the wiki. It does not burden the user with following the rules and guidelines for "cool URIs" (Sauermann and Cyganiak, 2008), but generates them automatically from the article name. Users can at any time introduce new individuals, properties, or classes. Because of that it does not make sense to use a hash namespace (see Section 4.1.2) as the returned file would be an ever-growing and -changing list of entity names. Instead a slash namespace is used, so that SMW can basically use the local name as a parameter in creating the required export of data.

## 10.3 Related systems

Before SMW, some other semantic wikis have been created, but most of them have been discontinued by now (Campanini *et al.*, 2004; Souzis, 2005; Nixon and Simperl, 2006). Many of the early semantic wikis emphasized the *semantic* side and disregarded some of the strengths of wikis such as their usability and low learning curve. One of the major design paradigms of SMW have been that SMW is a wiki first, and a semantic system second.

The most notable (and stable) related system currently is *KiWi* (Schaffert *et al.*, 2009), previously known as *IkeWiki* (Schaffert, 2006). KiWi is similar to SMW with respect to the supported kinds of easy-to-use inline wiki annotations, and various search and export functions. In contrast to SMW, KiWi introduces the concept of *ontologies* and (to some extent) *URIs* into the wiki, which emphasizes use-cases of collaborative ontology editing that are not the main focus of SMW. KiWi uses URIs explicitly to identify concepts, but provides interfaces for simplifying annotation, e.g. by suggesting properties.

Besides text-centered semantic wikis, various collaborative database systems have appeared recently. Examples of such systems include *OntoWiki* (Auer *et al.*, 2006), *OpenRecord*,[6] *freebase*,[7] and *OmegaWiki*.[8] Such systems typically use form-based editing, and are used to maintain data records, whereas SMW concentrates on text, the main type of content in wikis. OntoWiki draws from concepts of semantic technolo-

[5] `http://simile.mit.edu/wiki/Longwell`
[6] `http://www.openrecord.org`
[7] `http://www.freebase.com`
[8] `http://www.omegawiki.org`

gies and provides a built-in faceted (RDF) browser. The other systems have their background in relational databases. There are two extensions to SMW that help with making SMW more similar to such a form-based editing system, Semantic Forms developed by Yaron Koren[9] and Halo developed by ontoprise.[10]

SMW has become the base for a number of further research works in the area of semantic wikis. (Rahhal *et al.*, 2009) describes a Peer2Peer extension of SMW that allows the distributed editing of the semantic wiki. (Bao *et al.*, 2009a) describes the usage of SMW as a light weight application model, implementing two applications on top of it. The MOCA extension (Kousetti *et al.*, 2008) to SMW fosters the convergence of the emerging vocabulary within an SMW instance. A convergent vocabulary is not only a requirement for the proper usage of a semantic wiki, but also for the automatic content checks described in Section 10.4.

## 10.4 Collaborative ontology evaluation

Semantic wikis have shown to be feasible systems to enable communities to collaboratively create semantically rich content. They enable users to make the knowledge within the wiki explicit. This also allows the wiki to automatically check and evaluate the content. In this section we present a number of approaches in order to provide facilities to ensure the content quality of a wiki, including the application of constraint semantics and autoepistemic operators in ways that are easy accessible for the end user. Wikis such as Wikipedia do not work solely because of their underlying software, but due to their rather complex socio-technical dynamics that work due to often implicit community processes and rules (Ayers *et al.*, 2008). We first introduce a number of technical implementation for some exemplary evaluations that can be performed automatically on top of knowledge formalized within an SMW (Vrandečić, 2009c): *concept cardinality* in Section 10.4.1, *class disjointness* in Section 10.4.2, and *property cardinality constraints* in Section 10.4.3. We close this chapter with a discussion of the social aspects of collaborative ontology evaluation in Section 10.4.4.

### 10.4.1 Concept cardinality

Concept cardinality states how many results a query within the wiki should have. Besides exact numbers also minimal and maximal cardinalities are allowed. For the description of the implementation we assume that the query is captured by a concept. Then `Template:Cardinality` can be added to the concept page.

```
{{#ifeq:{{#ask:[[Concept:{{PAGENAME}}]]|format=count}}
       |{{{1}}}|OK|Not OK}}
```

[9] http://www.mediawiki.org/wiki/Extension:Semantic_Forms
[10] http://smwforum.ontoprise.com

The template assumes one parameter, the cardinality. Thus the template can be instantiated on a concept page such as `US states` as follows:

```
{{Cardinality|50}}
```

The format `count` returns simply the number of results. The `#ifeq` MediaWiki parser function checks if the first parameter, i.e. the query result, is equal to the second parameter, i.e. the first parameter of the template call (in our example 50). If they are equal, the third parameter will be returned (`OK`), otherwise the fourth parameter will be printed (`Not OK`). In a production setting the resulting text should be more refined to make sure that the user understands the raised issue.

In case we want not to check exact cardinality, but rather maximal or minimal cardinality we can use the MediaWiki parser function `#ifexpr` that checks if an expression is true or not.[11]

### 10.4.2 Class disjointness

Class disjointness states a page should not be categorized by the two specified categories at the same time. For this case we suggest the `Template:Disjoint`:

```
Testing {{{1}}} and {{{2}}} -
{{#ask: [[Category:{{{1}}}]] [[Category:{{{2}}}]]
 |default=OK|Intro=Inconsistent individuals: }}
```

The triple-brackets are replaced with the first and second parameter to the template call, respectively. The inline query will return `OK`, if no results exists, or the list of individuals prepended with the `Intro`, as shown below. This template can be called on any wiki page like this:

```
{{Disjoint|Man|Woman}}
```

The template call will result in a little report that could look like this:

```
Testing Man and Woman - Inconsistent individuals:
[[Hermes]], and [[Aphrodite]]
```

Since the inconsistent individuals are already linked, the user can quickly naviagte to their pages and check them.

Note that the template is not necessary neither in this case nor in the previous. We could simply add the query directly on any wiki page. The template is merely used to allow other users to easily add tests to pages without having to understand how the SMW query language works.

[11] http://www.mediawiki.org/wiki/Help:Extension:ParserFunctions#.23ifexpr

### 10.4.3 Property cardinality constraints

Property cardinalities are statements about how often a property should be used on a specific page or point to a specific page. We can start with a similar approach as for concept cardinalities, introducing `Template:Property cardinality`:

```
{{#ifexpr:{{#ask:[[{{{Property}}}::{{{Value}}}]]|format=count}}
          <= {{{Maximal cardinality}}}|OK|Not OK}}
```

We could now call it for any page to check if that page's individual fulfills the constraint or not. The following template call checks if the individual `Seth` has indeed a maximum of one `father`.

```
{{Property cardinality
 |Property=Father
 |Maximal cardinality=1
 |Value=Seth
}}
```

We would prefer not to call this manually for every individual, but rather for all individuals of a category at once. To achieve this we can use the query format "template". It applies the given template on every query result. We first define the formatting template `Template:Property cardinality format`:

```
{{{1}}} ({{Property cardinality
 |Property=Father
 |Maximal cardinality=1
 |Value={{{1}}}
}})
```

The parameter `{{{1}}}` means the first parameter, in this case each result of the query. Now we can use this template to format all results of a query, whereby it evaluates the given cardinality restrictions. The query could look like this:

```
{{#ask:[[Category:Person]]
 |link=none
 |format=template
 |template=Property cardinality format
 |sep=, }}
```

This query will return a comma-separated list of all persons, followed by the evaluation result for each person in brackets.

Note that the expressivity of SMW itself is more restricted than OWL, but exports from SMW have been used in combination with more expressive background ontologies (as suggested in Section 9.2.1) and then evaluated by external OWL inference engines (Vrandečić and Krötzsch, 2006). It has to be noted that the above checks do not have the normal OWL semantics. Concept and property cardinality in SMW do not relate to nominals and OWL property cardinalities, but follow rather the semantics of autoepistemic operators as discussed in Section 9.2.3. Within the wiki, we assume a closed world and unique names if not otherwise stated. The deviation from OWL semantics has to be carefully considered when exporting data from the wiki.

Not all possible evaluations can be expressed within the wiki. For example, domain and range constraints in the sense of Section 9.2.4 can not be obviously implemented within SMW.[12]

### 10.4.4 Social and usability aspects

An evaluation framework such as the one available in SMW allows the organic growth and addition of further content evaluations as deemed necessary by the wiki community. We expect that some wikis will introduce specific evaluation approaches that only apply to the domain at hand. Making the above constraints in the wiki must be simple enough to allow contributors to actually apply these features. The given selection is based on the fact that they can be represented within the wiki simply and unambiguously, i.e. contributors will always know where to look for a specific piece of information. This is a necessary requirement in order to keep a wiki maintainable.

This is not the case for the actual evaluations themselves. They can be put on any page – in particular, users can create their own private pages where they define their own evaluation criteria and run their own tests. Or they can be collaboratively run and maintained on a more central page. This way each member of the wiki community can decide on their own how and what part of the ontology to monitor.

A major decision embedded in the given ontology evaluation framework is to indeed allow contributors to introduce inconsistencies. When a page is modified, the wiki could check if that edit would turn it inconsistent and then cancel the edit. But even disregarding if a real time check would be computationally feasible, this seems to heavily conflict with the wiki paradigm. Instead of prohibiting edits that lead to inconsistencies we introduce means to report discovered problems and allow to repair them efficiently by linking to the respective pages.

Having all the evaluations in the wiki being implementable ad-hoc by any contributor, without the requirement of programming PHP and having access to the server

[12] We currently assume that SMW together with parser functions is Turing-complete. If this is the case, any evaluation that can be done automatically at all could be expressed within SMW. But it can turn out that an actual implementation is too hard to be feasible. This is what we mean with *obviously implementable*.

and the underlying code, we expect numerous evaluations to bloom. The prize of this freedom, though, is a high computational tax. Executing multiple intervowen layers of template calls and inline queries by the MediaWiki parser is computationally expensive. We expect that some template combinations will turn out to be both useful and expensive, in which case new extensions can be written natively in PHP to replace these template calls with calls to new parser functions. This has happened before in Wikipedia with several features (Schindler and Vrandečić, 2010). We regard this as a prime example for the Web Science process (Berners-Lee *et al.*, 2006b), where technological and social approaches interplay heavily with each other in order to come up with a Web-based solution to a problem.

Chapter 11

# Related work

> Wanting connections, we found connections – always, everywhere, and between everything. The world exploded in a whirling network of kinships, where everything pointed to everything else, everything explained everything else ...
>
> *(Umberto Eco, b. 1932, Foucault's Pendulum (Eco, 1988))*

This thesis presents a framework for ontology evaluation. In this chapter we discuss other frameworks with a similar intention, and relevant aspects in the creation of the underlying framework in Section 11.1. We discuss further methods and approaches to ontology evaluation in Section 11.2 and show how they fit into our framework. We expect that future work – especially newly developed evaluation methods, further implementations of existing methods, and evaluations regarding existing methods – can be similarly integrated in our framework. Section 11.3 then closes this chapter by discussing the corpora used for ontology evaluation within this thesis.

Note that the relation to our own previously published work is given in Section 1.4.

## 11.1 Frameworks and aspects

There are already a number of frameworks for ontology evaluation. The main goal of the framework in this thesis is the assessment of the qualities of one given ontology.

The other frameworks often have slightly different, although related goals. The most common such goals are ontology ranking and ontology selection.

**Ontology ranking** has the goal of sorting a given set of ontologies based on some criteria. Often these criteria can be parameterized with a context, usually a search term. Ontology search engines such as FalconS (Cheng *et al.*, 2008), Watson (d'Aquin *et al.*, 2007b), SWeSE (Harth *et al.*, 2009), or Swoogle (Ding *et al.*, 2004) have to perform the task of ontology ranking when deciding which ontologies to display in what order to the user who searched for ontologies. Other ontology ranking frameworks are (Alani and Brewster, 2006) and (Tartir *et al.*, 2005).

**Ontology selection** can be regarded as a specialization of ontology ranking as it selects only a single result, often for being reused in an ontology engineering task. The Cupboard system (d'Aquin and Lewen, 2009) presents a similar, though more fine-grained approach that allows not the selection of whole ontologies but rather the selection of single axioms. It incorporates a topic-specific open rating system (Lewen *et al.*, 2006) for assessing the ontologies and then provides further algorithms to present the individual axioms, but the final decision lies with the ontology engineer who selects the axioms to reuse.

The framework presented here differs from ranking and selection frameworks as it does not regard and sort sets of given ontologies, but only assesses individual ontologies by themselves. Many of the methods presented here can also be used in a ranking or selection framework, but some have to be adopted: a number of the methods do not yield a numerical score but rather a list of problematic parts of an ontology (such as Method 1 on page 67 or Method 11 on page 86), others may yield a number but this number is not simply proportional to ontology quality but may have a complex relation to it (if at all – compare the metrics introduced in Method 12 on page 101).

Some ontology evaluation frameworks are based on defining several criteria or attributes; for each criterion, the ontology is evaluated and given a numerical score. Additionally a weight is also assigned (in advance) to each criterion, and an overall score for the ontology is then computed as a weighted sum of its per-criterion scores. (Burton-Jones *et al.*, 2005) proposes an approach of this type, with ten simple evaluation methods (called *attributes*) such as lawfulness, interpretability, comprehensiveness, etc. The methods are grouped in so called *metric suites*, comparable to ontology aspects in our framework. The four suites are *syntax*, *semantics*, *pragmatics*, and *social*, but even though the names are partially equal, the metric suites do not correspond to the individual aspects in our framework, i.e. they are differently defined. (Burton-Jones *et al.*, 2005) further assumes that every method is a metric $\mathcal{M} : \mathcal{O} \rightarrow [0, 1]$. Each metric suite is a weighted average of its attributes, and the overall ontology quality is a weighted average over the results of the metric suites, thus achieving a simple, overall quality measure for ontologies between 0 and 1, with 1 denoting the perfect ontology.

(Fox and Gruninger, 1998) proposes an alternative set of criteria for ontology evalua-

tion: functional completeness, generality, efficiency, perspicuity, precision granularity, and minimality. These criteria can mostly be mapped to our catalog: functional completeness is within completeness, generality in adaptability, efficiency within computational efficiency, perspicuity in clarity, and minimality in conciseness. Precision granularity asks: *"Is there a core set of ontological primitives that are partitionable, or do they overlap in meaning? Does the representation support reasoning at various levels of abstraction and detail?"* The further description of this criterion indicates that this is overlapping with our criteria of accuracy, clarity, and completeness. The set is fairly compatible to our set, the main difference being that the criteria in (Fox and Gruninger, 1998) are not desiderata used to group evaluation methods, but fully defined as reachable goals with a metric to measure them (and thus would rather be methods in our framework than criteria really).

(Lozano-Tello and Gómez-Pérez, 2004) defines an even more detailed set of 117 criteria, organized in a three-level framework. The criteria cover various aspects of the formal language used to describe the ontology, the contents of the ontology, the methodology used, the costs (hardware, software, licensing, etc.) of using the ontology, and the tools available.

Our methodology for selecting the ontology evaluation criteria is presented in Section 3.6. For the selection of criteria for the ontology evaluation framework presented in this thesis we have focused on ontology evaluation literature. There are a number of related research fields, such as information and data quality, software engineering (especially the evaluation of software architectures and software models), and database engineering (especially in the field of database schemas and data modeling).

In systems engineering, quality attributes as non-functional requirements are sometimes called *ilities* due to the suffix many of the words share (Voas, 2004). In data and information quality research many of theses *ilities* are regarded as quality attributes: accessibility, accountability, accuracy, auditability, availability, credibility, compatibility, effectiveness, extensibility, etc. Data is defined as having a high quality *"if they are fit for their intended uses in operations, decision making and planning"* (Juran and Godfrey, 1999). A standard for data quality is being developed as ISO 8000, currently covering the criteria conformance to a specification (ISO 8000-110, 2009), vocabulary (ISO 8000-102, 2009), provenance (ISO 8000-120, 2009), accuracy (ISO 8000-130, 2009), and completeness (ISO 8000-140, 2009).

We regard the criteria catalog in Section 3.6 not as a fixed, unchanging list, but as the current state of the art in ontology evaluation research based on our literature survey. The criteria are used to categorize ontology evaluation methods and to help users to find methods relevant for their tasks quickly. The list may be extended or changed and thus may potentially include further relevant criteria originating in related fields of studies or in further work in ontology evaluation.

As noted in Section 3.4 this thesis covers more the field of *ontology verification* (as

opposed to *ontology validation*). A complementing work covering the area of ontology validation is provided in (Obrst *et al.*, 2007). It provides a concise overview of many evaluation methods and techniques not discussed within this thesis. They are:[1]

- the evaluation of the use of an ontology in an application (see Section 11.2.2).
- the comparison against a source of domain data (see Section 11.2.3).
- assessment by humans against a set of criteria. Human experts are used to gauge an ontology against a set of principles *"derived largely from common sense"*.
- natural language evaluation techniques. This evaluates the ontology within a natural language processing application such as information extraction, question answering or abstracting.
- using reality as a benchmark. Here the notion of a *"portion of reality"* (POR) is introduced, to which the ontology elements are compared.

It is not claimed that the list of techniques is a complete list. We further point out that the list is not even a list of techniques: one of the points describes a dimension of evaluation methods (whether the evaluation is performed automatically, semi-automatically, or manually, i.e. by humans), one is a specialization of the other (natural language evaluation techniques specialize the application-based evaluation), and the last one is describing a methodological condition for ontology evaluation techniques (using reality as a benchmark). As shown in Section 3.5 we do not commit to a strong theory of reality, i.e. an accessible objective ideal that can be used to be compared with an ontology. We furthermore do not think that ontologies should be restricted in specifying conceptualizations of domains within reality, but should be also allowed to specify conceptualizations of fictional domains, such as the family tree of the mythological Greek gods, the history of Tolkien's Middle-earth, or scientific theories about the luminifereous aether.

Whereas we disagree with the framework described in (Obrst *et al.*, 2007), we agree that many of the listed methods and techniques are important for a comprehensive ontology evaluation, especially for ontology validation. Ontology validation is usually the only way to ensure the correctness of the knowledge encoded in the ontology. But most validation approaches require the close cooperation of domain and ontology engineering experts. Validation often can not be performed automatically. Since this thesis focuses on automatic evaluation approaches, we leave it to (Obrst *et al.*, 2007) to provide an overview of validation approaches.

[1] There is also a section on *ontology accreditation, certification, and maturity model*, but it is made clear that this is a discussion about the future of ontology evaluation and not describing a technique *per se*

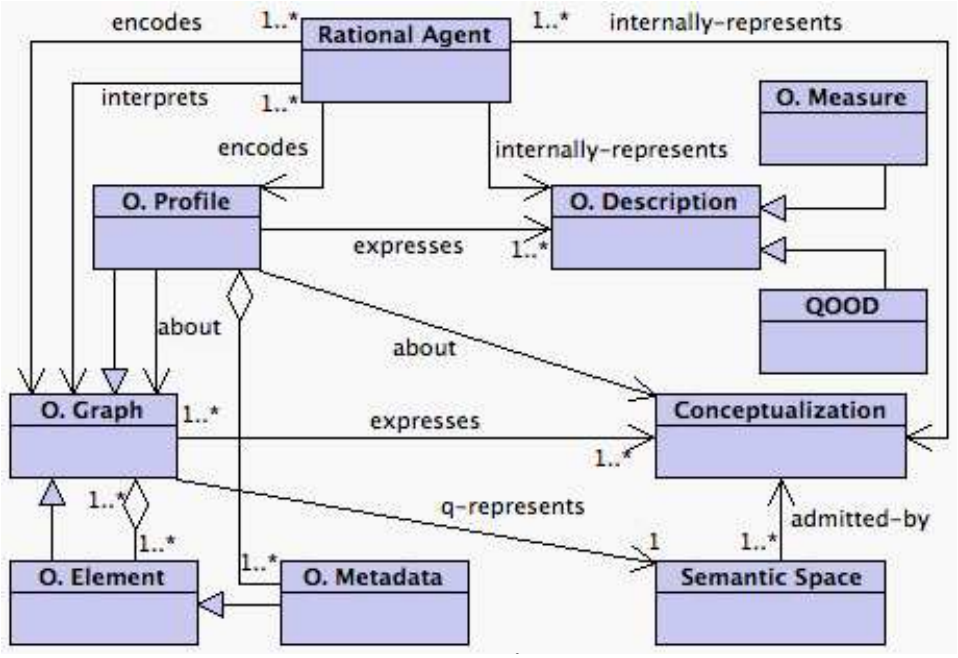


Figure 11.1: UML diagram of the main notions from $O^2$ (from Gangemi *et al.*, 2005).

As discussed in Chapter 3, our framework is strongly influenced by the $O^2$ framework presented in (Gangemi *et al.*, 2005; Gangemi *et al.*, 2006a; Gangemi *et al.*, 2006b). Figure 11.1 gives an overview of the $O^2$ ontology. Here we specify the main differences of our framework given in Section 3.1 on page 37) compared to $O^2$: $O^2$ does not use the term **ontology** explicitly but rather assumes that the **ontology graph** equals the ontology. This leads to a number of ambiguities regarding the proposed metrics in (Gangemi *et al.*, 2005), exemplarily discussed in depth in Section 6.1.2. Since in $O^2$ an **ontology** is an **ontology graph**, it also consists of **ontology elements** that are graphs as well (i.e. nodes and arcs). In our framework an ontology is structurally defined as a set of **axioms**, and the possible axioms and their constituents are fully described in the meta-ontology, which is not the case in $O^2$. **Semantic space** in $O^2$ corresponds to **model** in our framework. $O^2$ does not include **evaluation methods** explicitly. And **ontology description** in $O^2$ corresponds to **ontology evaluation** in our framework. Furthermore, every **ontology description** is expressed by an **ontology profile** which in turn is an ontology, i.e. an ontology measure expresses its result as metadata in an ontology about the measured ontology. Although this has a number of advantages, we consider this as too restrictive. In our framework, descriptions *may* be expressed as ontologies, but this is not required (see Section 3.1 for an example).

## 11.2 Methods and approaches

Work in ontology evaluation has grown considerably during the last few years, also due to two workshops on ontology evaluation (Vrandečić *et al.*, 2006a; Vrandečić *et al.*, 2007a). Even though we already included numerous methods within our framework in Chapters 4–9, the list cannot be exhaustive. We hope that future work will regard this framework as an orientation and localize new methods within the framework. In the following, we regard a number of methods that have not been mentioned explicitly in this thesis and are here placed within our framework, defining the two dimensions of aspects and criteria for each of the methods.

A major line of research that has been hardly discussed in this thesis is the **logical consistency** or **satisfiability** of an ontology and the debugging of inconsistencies (Parsia *et al.*, 2005). Other papers extend the notion of satisfiability. For example, (Gómez-Pérez, 2004) defines that class hierarchy cycles or partition problems also mean inconsistent ontologies. Such issues are often addressed by specific ontology repair tools such as SWOOP[2] or ODEval.[3] **Inconsistency explanation** (Ji *et al.*, 2009) deals with the issue of creating user-understandable explanations of inconsistencies so that the user can deal with the given problem. Satisfiability is an issue of the *representation* aspect, and is covered by the criteria *consistency* and *computational efficiency*.

**Integrity constraints** are extended on top of Section 6.3.4 in (Briggs, 2008; Sirin and Tao, 2009) and can also be regarded as extending the definition of satisfiable ontologies. They often belong to the *context* aspect since they add the integrity constraints as an external artifact, and further the *accuracy* and *consistency* criteria.

Some newer ontology engineering methodologies such as DILIGENT (Tempich *et al.*, 2005) and HCOME (Kotis *et al.*, 2005) take into account the fundamental role of **discourse and agreement** between different stakeholders. They identify the **sharedness** of the ontology as an explicit value and provide processes to guarantee that sharedness is achieved. Even though we do not regard sharedness as an ontology quality criterion (Section 3.6) of its own, it is an integral part of our conceptualization (Section 3.5). Sharedness regards the aspect of *semantics* and touches a number of criteria: *accuracy*, *clarity*, and *organizational fitness*.

The larger a group that commits to an ontology (and the shared conceptualization it purports), the harder it is to reach a consensus – but also the larger the potential benefit. Thus the status of the ontology with regards to relevant standardization bodies in the given domain is a major criteria when evaluating an ontology. Ontologies may be **standardized or certified** by a number of bodies such as W3C, Oasis, IETF, and other organizations that may standardize ontologies in their area of expertise (Obrst

[2] http://www.mindswap.org/2004/SWOOP/
[3] http://minsky.dia.fi.upm.es/odeval

*et al.*, 2007) – or just a group of peers that declare an ontology to be a standard (see the history of RSS (Beged-Dov *et al.*, 2000) or FOAF (Brickley and Miller, 2005) for examples). In certain cases, **adoption** by relevant peers (especially business partners) may be even more important than standardization. Tools like Swoogle (Ding *et al.*, 2004) allow to check how often an ontology is instantiated, and thus to measure the adoption of the ontology on the Web. The standardization refers to the *vocabulary* aspect and the criterion of *organizational fitness*.

A related question is the **grounding** of the terms in an ontology (Jakulin and Mladenić, 2005), i.e. how will the terms in the ontology be understood by the users of the ontology (either directly, or via a certain tool). Since there is no way to encode the meaning of a term (besides the very weak understanding of meaning as the model-theoretic formal semantics) we need to make sure that a term such as `foaf:Person` is well grounded, usually through the documentation and shared understanding. In certain cases, the meaning can be completely grounded in a computer: since computers are able to recognize and handle XML files, for example, the `XML file` class can be fully "understood" by the computer (Oberle *et al.*, 2006). But for classes relating to the world outside of the computer this often remains a challenge. Evaluating the grounding is an evaluation of the *vocabulary* aspect addressing the criterion of *clarity*.

In (Stojanović, 2004) the theory and practice of **ontology evolution** is discussed. Ontology change operations and ontology evolution strategies are introduced. Based on this, (Haase and Stojanović, 2005) extends this work for OWL DL ontologies, and investigates the evolution of ontologies with regards to consistency, implemented in the so called evOWLution framework. As the theoretical work allows generic and user defined consistency checks, the ideas presented here could be regarded as a number of ways to formalize further aspects of the ontology, and enable more expressive consistency checks beyond simple logical satisfiability. This way we can extend the evolution framework over all aspects of the ontology, clearly improving the *adaptability* of the ontology.

The most widely evaluated aspect in current research is the *context* aspect. Frequently ontologies are paired with a context and then evaluated with regards to this context. In the survey of ontology evaluation methods provided by (Brank *et al.*, 2005) the evaluation methods are classified into four approaches:

- comparing the ontology to a golden standard,
- task-based ontology evaluation, i.e. using the ontology with an application and evaluating the application
- data-driven ontology evaluation, and
- evaluations performed by humans against a set of predefined criteria, standards or requirements.

We think that the last point does not belong to this list as it describes a dimension of the ontology evaluation method independent of the other approaches. The other approaches, though, are all part of what the framework in this thesis defines as being an evaluation of the context aspect, with the context being a golden standard, a task or application, or a set of external data respectively. In the following, we will regard evaluation methods that belong to these three categories of context.

### 11.2.1 Golden standard – similarity-based approaches

The evaluation based on a golden standard builds on the idea of using similarity measures to compare an ontology with an existing ontology that serves as a reference. This approach is particularly useful to evaluate automatically learned ontologies with a golden standard. The similarity between ontologies can be calculated using similarity functions (Ehrig *et al.*, 2005). A similarity function for ontologies is a real-valued function

$$sim : \mathcal{O} \times \mathcal{O} \rightarrow [0, 1]$$

measuring the degree of similarity between two ontologies. Typically, such measures are reflexive and symmetric. The similarity function to compare the ontologies can be used directly as the evaluation function, if we keep one of the arguments – the golden standard $GS$ – fixed.

(Ehrig *et al.*, 2005) shows how similarity functions for ontologies based on individual similarity measures for specific aspects and elements of the ontologies can be defined. We present specific similarity measures for some aspects of an ontology in the following.

On the vocabulary aspect, the similarity between two strings can be measured by the Levenshtein edit distance (Levenshtein, 1966), normalized to [0, 1]. A string matching measure between two sets of strings is then defined by taking each string of the first set, finding its similarity to the most similar string in the second set, and averaging this over all strings of the first set. In an evaluation setting, the second set is a "golden standard" set of strings that are considered a good representation of the classes of the problem domain under consideration. The golden standard could be another ontology, as in (Maedche and Staab, 2002), based on a document-corpus, or provided by experts.

The vocabulary can also be evaluated using precision and recall, as known in information retrieval. In this context, precision is the fraction of the labels that also appear in the golden standard relative to the total number of labels. Recall is the percentage of the golden standard lexical entries that also appear as labels in the ontology, relative to the total number of golden standard lexical entries. A disadvantage of these definitions is that they are strict with respect to spelling (e.g. different use of hyphens in multi-word phrases would not match, etc.).

(Velardi *et al.*, 2005) describes an approach for the evaluation of an ontology learning system which takes a body of natural-language text and tries to extract from it relevant

domain-specific classes (terms and phrases), and then find definitions for them (using Web searches and WordNet entries) and connect some of the classes by subsumptions. Part of their evaluation approach is to generate natural-language glosses for multiple-word terms. The glosses are of the form *"x y = a kind of y, definition of y, related to the x, definition of x"*, where $y$ is typically a noun and $x$ is a modifier such as an adjective. A gloss such as this would then be shown to human domain experts, who would evaluate it to see if the word sense disambiguation algorithm selected the correct definitions of $x$ and $y$. An advantage of this kind of approach is that domain experts might be unfamiliar with formal languages in which ontologies are commonly described, and thus it might be easier for them to evaluate the natural-language glosses. Of course, the disadvantage of this approach is that it nevertheless requires a lot of work on part of the domain experts. The precision of this disambiguations defines the ontology evaluation function.

(Maedche and Staab, 2002) proposes several measures, such as the semantic cotopy of two hierarchies, for comparing the structural aspect of two ontologies. With a golden standard, these measures can be used for ontology evaluation.

Given a golden standard, evaluation of an ontology on the semantic aspect can also be based on precision and recall measures, just like on the lexical layer. (Spyns, 2005) discusses an approach for automatically extracting a set of lexons, i.e. triples of the form `(term1 property term2)` from natural-language text. The result can be interpreted as an ontology, with terms corresponding to classes or individuals.

### 11.2.2 Task-based evaluations

The output of an ontology-based application, or its performance on a given task, will be better or worse depending on the utility of the ontology used in it. Often ontologies are tightly interwoven with an application, so that the ontology cannot be simply exchanged. It may drive parts of the user interface, the internal data management, and parts of it may be hard-coded into the application. On the other hand, the user never accesses an ontology directly but always through some application. Often the application needs to be evaluated with the ontology, regarding the ontology as merely another component of the used tool. Such a situation has the advantage that well-known software evaluation methods can be applied, since the system can be regarded as an integrated system where the fact that an ontology is used is of less importance.

A utility-based evaluation is presented in (Porzel and Malaka, 2004). There a scenario is described where the ontology is used primarily to determine how closely related the meaning of two classes is. The task is a speech recognition problem, where there may be several hypotheses about what a particular word in the sentence really means; a hypotheses should be coherent, which means that the interpretations of individual words should be classes that are relatively closely related to each other. Thus the ontology is used to measure distance between classes and thereby to assess the coher-

ence of hypotheses (and choose the most coherent one). The correctness of the results directly maps to the quality of the ontology with regards to its use in this scenario.

The evaluation function presented in (Haase and Sure, 2005) captures the intuition that the quality of an ontology built for searching is determined by how efficiently it allows the users to obtain relevant individuals. To measure the efficiency, a cost model is introduced to allow us to quantify the user effort necessary to arrive at the desired information. For the case of navigating a class graph, this cost is determined by the complexity of the hierarchy in terms of its breadth and depth. The *breadth* here means the number of choices (sibling nodes of the correct class) the user has to consider to decide for the right branch to follow: The broader the hierarchy, the longer it takes to make the correct choice. The *depth* means, how many links does the user need to follow to arrive at the correct class, under which the desired individual is classified: The deeper the hierarchy, the more "clicks" need to be performed. To minimize the cost, both depth and breadth need to be minimized, i.e. the right balance between them needs to be found.

(Sabou *et al.*, 2007) create custom-tailored ontologies on the fly from the formulation of a task (Alani, 2006) and evaluate them afterwards. Several problems are encountered, ranging from broken links to incompatible axioms due to different contexts and points of views.

Utility-based approaches often have drawbacks:

- They allow one to argue that the ontology is good or bad when used in a particular way for a particular task, but it is difficult to generalize this observation (what if the ontology is used for a different task, or differently for the same task?)
- the evaluation may be sensitive in the sense that the ontology could be only a small component of the application and its effect on the outcome may be relatively small (or depend considerably on the behavior of the other components)
- if evaluating a large number of ontologies, they must be sufficiently compatible that the application can use them all (or the application must be sufficiently flexible)

### 11.2.3 Data-driven evaluation – fitting the data set

An ontology may also be evaluated by comparing it to existing data about the domain to which the ontology refers. This can, e.g., be a collection of text documents. For example, (Jakulin and Mladenić, 2005) proposed the ontology grounding process based on the data representing individuals within the ontology classes. A set of errors in ontology grounding is shown to the user to help in the ontology refinement process. Ontology grounding can be used in the construction of a new ontology or in the data driven ontology evaluation.

Domain completeness is given when an ontology covers the complete domain of interest. This can be only measured automatically if the complete domain is accessible automatically and can be compared to the ontology. A way to assess the completeness of an ontology with respect to a certain text corpus is to use natural language processing techniques to detect all relevant terms in a corpus (Velardi *et al.*, 2005). The learned terms are then compared to the evaluated ontology to measure the coverage of the corpus i.e. the domain.

(Patel *et al.*, 2003) proposed an approach to determine if the ontology refers to a particular topic, and to classify the ontology into a directory of topics: one can extract textual data from the ontology (such as names of classes and properties, and other suitable natural-language strings) and use this as the input to a text classification model. The model itself can be trained by standard machine learning algorithms from the area of text classification; a corpus of documents on a given subject can be used as the input to the learning algorithm.

(Brewster *et al.*, 2004) suggested using a data-driven approach to evaluate the degree of structural fit between an ontology and a corpus of documents. (1) Given a corpus of documents from the domain of interest, a clustering algorithm is used to determine a probabilistic mixture model of hidden "topics" such that each document can be modeled as having been generated by a mixture of topics. (2) Each class $C$ of the ontology is represented by a set of terms including its name in the ontology and the hypernyms of this name, taken from WordNet. (3) The probabilistic models obtained during clustering can be used to measure, for each topic identified by the clustering algorithm, how well the class $C$ fits that topic. (4) At this point, if we require that each class fits at least some topic reasonably well, we obtain a technique for lexical-layer evaluation of the ontology. Alternatively, we may require that classes associated with the same topic should be closely related in the ontology. This would indicate that the structure of the ontology is reasonably well aligned with the hidden structure of topics in the domain-specific corpus of documents.

In the case of more extensive and sophisticated ontologies that incorporate a lot of factual information such as Cyc (Lenat, 1995), the corpus of documents could also be used as a source of "facts" about the external world, and the evaluation measure is the percentage of these facts that can also be derived from information in the ontology or that are consistent with the axioms of the ontology.

Many techniques for the automated generation of ontologies, e.g. ontology learning algorithms, provide different kinds of evidences with respect to the correctness and the relevance of ontology elements for the domain in question. For instance, in order to learn subsumptions, Text2Onto (Cimiano and Völker, 2005) applies a variety of algorithms exploiting the hypernym structure of WordNet (Fellbaum, 1998), matching Hearst patterns (Hearst, 1992) in the corpus, and applying linguistic heuristics (Velardi *et al.*, 2005). Based on the evidences one can compute confidences which model the certainty about whether a particular axiom holds for a certain domain.

## 11.3 Watson corpus

For a number of experiments throughout this thesis we have used corpora derived from the Watson collection. This section describes the corpora and how we created them.

In order to test our approach on a realistic corpus of Web ontologies we have created and made available a corpus of ontologies based on the Watson corpus. Watson is a search engine developed by the Knowledge Media Institute (d'Aquin *et al.*, 2007b). The complete corpus is simply called the Watson corpus and contains roughly 130,000 ontologies. It is available as part of the Billion Triple Challenge corpus. We used the 2008 edition.[4] We have further sampled randomly two subcorpora. Each ontology has a name based on a hash-sum of the ontology's URI. We defined two subcorpora based on these names: the Watson EA corpus (all ontologies where the hash started with EA) and the Watson 130 corpus (all ontologies where the hash started with 130). Watson 130 contains 35 ontologies, Watson EA 515 ontologies.

Some of the evaluations are based on an earlier corpus. We received an early copy of the Watson corpus in spring 2007, containing 5873 files. We filtered these ontologies to receive only valid OWL DL ontologies, so that only 1331 ontologies remained (checking using KAON2 (Motik, 2006)). These ontologies are made available online,[5] including metadata about the ontologies extracted during the experiments.[6] The set of valid OWL ontologies is called the Watson OWL corpus.

The ontologies were given short labels for easier reference (`A00` to `N30`). All ontologies can be retrieved in order to examine the results in this thesis within the context of the complete ontology. The metadata about the ontologies offers several key metrics about the ontology, e.g. the number of class names, the number of axioms, etc. Using the metadata file one can easily filter and select ontologies with specific properties. This corpus is by no means meant to be a full view of the Semantic Web, but just a partial, representative snapshot that should allow us to draw conclusions about current OWL DL ontology engineering practice on the Web. We assume that Watson is a random sample of the Web

The experiments were partially run using the RDFlib[7] Python library in version 2.4.0 instead of KAON2 that was used for DL checking. As of now, the KAON2 SPARQL engine allows only for conjunctive queries on the ABox of the ontology, but does not allow to query the TBox. Since most, though not all, ontology engineering patterns are indeed expressed in the TBox we had to resort to another SPARQL engine for some of the experiments.

[4] `http://challenge.semanticweb.org/`
[5] `http://www.aifb.uni-karlsruhe.de/WBS/dvr/research/corpus`
[6] `http://www.aifb.uni-karlsruhe.de/WBS/dvr/research/corpus/meta.rdf`
[7] Available from `http://rdflib.net/`

## Chapter 12

# Conclusions

> Uh huh. Uh huh. Okay. Um, can you repeat the part of the stuff where you said all about the... things. Uh... the things?
>
> *(Homer Simpson*
> *The Simpsons,*
> *Season 7, Episode 17*
> *(Swartzwelder, 1996))*

When we started with this thesis, we had the naïve goal of achieving a simple, fully automatically computable, real-valued quality function:

$$Q : \mathcal{O} \rightarrow [0, 1]$$

Given two ontologies $O_1$ and $O_2$ we wanted to be able to use the measure, get results such as $Q(O_1) = 0.73$ and $Q(O_2) = 0.62$, and thus not only being able to state that one ontology is better than the other, but also how much better it is.

As said, it was a naïve goal. In (Burton-Jones *et al.*, 2005) such a measure is indeed defined, but there are so many shortcomings with a measure like this: can a simple measure really capture the many dimensions of an ontology? How does this number help in engineering and maintaining ontologies? Just having this number does not tell us how to improve the ontology, nor does it point out to the problems an ontology may have. We had to redefine our goal: instead of such a measure we aimed for methods that help us tell how good an ontology is, to assess the *quality* of an ontology.

But also this quest for quality did not lead to a satisfying result, especially since *"Quality cannot be defined"* (Pirsig, 1984). So once again we changed our goal: instead of aiming for evaluation methods that tell us if an ontology is good, we settled for the

goal of finding ontology evaluation methods that tell us if an ontology is bad, and if so, in which way. This turned out to be the most useful approach in order to get closer to our goals: improving the quality of ontologies on the Web in general and thus gaining advantages from better ontologies, increasing the availability of ontologies by providing usable methods to test ontologies before release, and lower the maintenance costs for ontologies by providing methods that point out possible errors well in advance. But now it should be clear that none of the methods, neither alone nor in combination, can guarantee a good ontology.

This final chapter summarizes the achievements of this thesis in Section 12.1 and lists the many open research questions and development challenges in Section 12.2.

## 12.1 Achievements

The result of this thesis is a comprehensive framework for the evaluation of ontologies. The framework organizes ontology evaluation methods in two dimensions: ontology quality criteria (accuracy, adaptability, clarity, completeness, computational efficiency, conciseness, consistency, and organizational fitness) and ontology aspects (vocabulary, syntax, structure, semantics, representation, and context). For all criteria and for all aspects we presented methods to evaluate the given criteria or aspect. We added a number of new techniques to the toolbox of an ontology engineer, such as stable metrics, XML based ontology validation, reasoning over a meta-ontology, and others.

Unlike other evaluation frameworks and methods we separated an ontology into the given aspects, thus making it clear what is actually being evaluated. A common error in current research is to mix up semantics and structure. Chapters 6–8 show how to keep these levels separate, and offers the tool of normalization in order to assess exactly what the metrics engineer claims to assess. This will clarify the conceptualization surrounding the evaluation of ontologies, and help with describing new ontology evaluation methods and what their benefits will be.

The framework in this thesis is also novel as far as it puts some emphasis on the evaluation of the "lower aspects" of the ontology, i.e. vocabulary, syntax, and structure (Chapters 4–6). Only recently, with the strong shift towards *Linked Data*, have these lower levels gained increased scrutiny. This is not yet reflected so much in research work but rather in informal groups such as the *Pedantic Web*.[1] Other evaluation frameworks in published research almost exclusively focus on the aspects of semantics and context. But our extensive survey of existing ontological data shows that many ontologies have easily reparable issues on those low levels already. Without means to evaluate those aspects it is hard to fix them. We hope that our framework will show to be a more comprehensive and inclusive framework that takes into account both parts of ontology evaluation, and will ultimatively help with improving the overall quality

[1] http://pedantic-web.org

of ontologies on the Web. This will hopefully not only improve the availability and usefulness of semantic data on the Web, but also point out a path to reconcile the two research streams on linked data and expressive ontologies, highlighting the mutual benefit they can gain from each other.

## 12.2 Open questions

In the following we list a number of open questions and research challenges raised by this thesis. Many of the methods have their own, specific list of open issues: XML schema validation can be extended with more powerful schema languages, normalization may offer benefits in other areas besides ontology evaluation, and it would be interesting to investigate if it is possible to define a method that turns a normal metric into a stable metric. We pointed out to specific research questions throughout this thesis. Here we will list more general research questions that pertain to the framework as a whole.

It is obvious that a domain- and task-independent verification, as discussed here, provides some common and minimum quality level, but can only go so far. In order to properly evaluate an ontology, the evaluator always needs to come up with methods appropriate for the domain and task at hand, and decide on the relative importance of the evaluation criteria. But the minimum quality level discussed here will at least provide the ontology engineer with the confidence that they eliminated many errors and can publish the ontology. Providing a framework for creating and understanding domain- and task-aware evaluation methods, integrating the rich work in this field, remains an open issue.

As we have seen, numerous quality evaluation methods have been suggested in literature. But only few of them have been properly designed, defined, implemented, and experimentally verified. The relation between evaluation methods and ontology quality criteria is only badly understood and superficially investigated, if at all. For example, (Gangemi *et al.*, 2005) discusses a number of quality criteria and how some of the metrics may serve as indicators for certain criteria, either positive or negative. But they do not report any experiments investigating the correlations between the results of the methods and the criteria. Recent experiments in turn point to some indeed counterintuitive relations: a higher *tangledness* actually increases efficiency when using the ontology in a browsing task (Yu *et al.*, 2007). This contradicts the more intuitive relationship described in (Gangemi *et al.*, 2005) where tangledness is a *negative* indicator for *cognitive ergonomics*, which includes exploitability of the ontology by the user. The lack of experimental evaluations matching methods and criteria will hinder meaningful ontology evaluations. A better understanding of the connection between the features of an ontology and quality criteria remains easily the most important open research challenge in ontology evaluation. Also evaluations using scientific tools

from the field of psychology are expected to further show their usefulness in evaluating ontologies. (Yamauchi, 2007) provides an example, where the difference between two possibilities to represent formal models is evaluated, but we expect this to be merely the beginning towards a better understanding of human conceptualizations, which, in the end, form the foundation for every ontology.

Most of the presented methods in this thesis are only prototypically implemented, be it as tools of their own (like the XML schema-based validation) or be it as part of a toolset (like the structural metrics implemented in the KAON2 OWL tools). What this thesis did not achieve is the implementation of a comprehensive application that applies the various described evaluation methods and provides a summarizing report, either as a part of an ontology development environment or as a stand-alone application. We expect such a validation tool to be of great use. Also many of the current implementations are not efficient. We have defined the results formally, but for a number of our prototypical implementations we do not expect them to scale to realistically sized ontologies. We expect that future research will realize efficient implementations of those methods that have proven useful.

We have implemented a number of the evaluation methods within a collaborative semantic authoring system, Semantic MediaWiki. SMW was developed and implemented during the creation of this thesis. We expect the field of collaborative ontology evaluation to become an increasingly important topic for collaborative knowledge construction. But what we see today is just the beginning of this interesting, new research track. We expect the close future to show hitherto unknown levels of cooperation between groups of humans and federations of machine agents, working together to solve the wicked problems we face today.

# Part IV

# Appendix

List of Methods 203

List of Tables 205

List of Figures 207

Bibliography 209

Full Table of Contents 230

# List of Methods


1 Check used protocols . . . 67
2 Check response codes . . . 69
3 Look up names . . . 71
4 Check naming conventions . . . 73
5 Metrics of ontology reuse . . . 74
6 Check name declarations . . . 75
7 Check literals and data types . . . 78
8 Check language tags . . . 79
9 Check labels and comments . . . 81
10 Check for superfluous blank nodes . . . 82
11 Validating against an XML schema . . . 86
12 Ontology complexity . . . 101
13 Searching for Anti-Patterns . . . 114
14 OntoClean meta-property check . . . 125
15 Ensuring a stable class hierarchy . . . 140
16 Measuring language completeness . . . 141
17 Explicitness of the subsumption hierarchy . . . 145
18 Explicit terminology ratio . . . 146
19 Checking competency questions against results . . . 154
20 Checking competency questions with constraints . . . 155
21 Unit testing with test ontologies . . . 157
22 Increasing expressivity . . . 160
23 Inconsistency checks with rules . . . 161

# List of Tables


1.1 Namespace declaration in this thesis . . . . . 18

2.1 Semantics of OWL axioms. Axiom types noted with * may hold more than the given parameters. . . . . 34
2.2 Semantics of OWL expressions using object properties (datatypes properties are analogous). Expression types with * may hold more parameters. 35

4.1 An overview of what different response codes imply for the resolved HTTP URI reference $U$. $I$ is the information resource that is returned, if any. $L$ is the URI given in the location field of the response. The table covers the most important responses only, the others do not imply any further facts. . . . . 68
4.2 The five hash and slash namespaces with the biggest number of names. 71

6.1 Constraint Violations for Manual Taggings. The lower part of the table shows constraint violations based on taggings from the inter-annotater agreed sets, e.g. $A_2$ / $A_3$ shows the number of violations based on only the taggings where annotator $A_2$ and $A_3$ agreed on. . . . . 123
6.2 Constraint Violations for Automatic Taggings . . . . . 124

7.1 Class and property assertions to calculate the language completeness of the example ontology. . . . . 142

# List of Figures


3.1 Framework for ontology evaluation. The slashed arrow represents the *expresses* relation. . . . 38
3.2 A subsumption axiom (on the left) and its reification. Dotted lines represent instantiation, slashed lines annotations, circles individuals, and squares classes. . . . 40
3.3 The semantic spectrum for ontologies. . . . 45
3.4 Two agents $X$ and $Y$ and their conceptualizations of domain $d$ (the tree), each other, and their respective conceptualizations. . . . 51
3.5 Three agents and an ontology. $Y$'s conceptualization is omitted for space reasons. $Z$ internalizes ontology $O$, thus connecting it to or creating its own conceptualization $C_Z$ of domain $d$, in this case, the tree. 52

4.1 Distribution of the HTTP response codes on the HTTP URIs from the Watson EA corpus. The left hand side shows the slash URIs, the right hand side hash URIs. . . . 69
4.2 The fifteen most often used data types in the Watson corpus. Note the logarithmic scale. . . . 77

6.1 Example for a circular hierarchy path. . . . 103
6.2 Two class hierarchies with identical semantics. Semantic similarity measure *ssm* of $C_1$ and $C_3$ is lower in the left ontology than in the right one. . . . 106
6.3 The partition pattern: class $A$ is partitioned into the subclasses $B_1 \ldots B_n$ 108
6.4 The upper levels of the time ontology. Note that `ProperInterval` and `Instant` are declared disjoint even though they are not sibling classes. 114

8.1 A simple taxonomy before (left) and after (right) normalization. The arrows denote subsumption. . . . 144

9.1 Example class hierarchy. . . . 153

10.1 Architecture of SMW's main components in relation to MediaWiki. . . 168
10.2 Source of a page about London in MediaWiki (top) and in SMW (bottom). 171
10.3 A semantic view of London. . . . 172

10.4 Inverse search in SMW, here giving a list of everyone born in London. 174
10.5 Production rules for SMW queries (top) and according OWL descriptions (bottom). . . . . 176
10.6 A semantic query for all cantons of Switzerland, together with their capital, population, and languages. The data stems from an automatically annotated version of Wikipedia. . . . . 178

11.1 UML diagram of the main notions from $O^2$ (from Gangemi *et al.*, 2005). 189

# Bibliography


Harith Alani and Christopher Brewster. Metrics for ranking ontologies. In Denny Vrandečić, Mari del Carmen Suárez-Figueroa, Aldo Gangemi, and York Sure, editors, *Proceedings of the 4th International Workshop on Evaluation of Ontologies for the Web (EON2006) at the 15th International World Wide Web Conference (WWW 2006)*, pages 24–30, Edinburgh, Scotland, May 2006.

Harith Alani. Position paper: ontology construction from online ontologies. In Les Carr, David De Roure, Arun Iyengar, Carole A. Goble, and Michael Dahlin, editors, *Proceedings of the 15th international conference on World Wide Web (WWW2006)*, pages 491–495, Edinburgh, Scotland, May 2006. ACM.

Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel. *A Pattern Language*. Oxford University Press, New York, NY, 1977.

Dean Allemang and James Hendler. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Morgan Kaufman, San Francisco, CA, 2008.

Aristotle. *Metaphysics*. Oxford University Press, 330 BC. translated by W. D. Ross.

Sören Auer and Jens Lehmann. What have Innsbruck and Leipzig in common? Extracting semantics from wiki content. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, *Proc. 4th European Semantic Web Conference (ESWC)*, 2007.

Sören Auer, Sebastian Dietzold, and Thomas Riechert. OntoWiki – A tool for social, semantic collaboration. In Yolanda Gil, Enrico Motta, Richard V. Benjamins, and Mark Musen, editors, *Proc. 5th Int. Semantic Web Conference (ISWC'05)*, number 4273 in LNCS, pages 736–749. Springer, 2006.

Phoebe Ayers, Charles Matthews, and Ben Yates. *How Wikipedia works*. No Starch Press, San Francisco, CA, October 2008.

Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, New York, NY, USA, 2003.

Jie Bao, Li Ding, Rui Huang, Paul Smart, Dave Braines, and Gareth Jones. A semantic wiki based light-weight web application model. In *Proceedings of the 4th Asian Semantic Web Conference*, pages 168–183, 2009.

Jie Bao, Sandro Hawke, Boris Motik, Peter F. Patel-Schneider, and Axel Polleres. rdf:PlainLiteral: A Datatype for RDF Plain Literals, 2009. W3C Recommendation 27 October 2009, available at http://www.w3.org/TR/rdf-text/.

Daniel J. Barret. *MediaWiki*. O'Reilly, 2008.

Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Abstract Reference, 2004. W3C Rec. 10 February 2004.

Kent Beck. *Extreme Programming*. Addison-Wesley, Reading, MA, 1999.

Dave Beckett and Jeen Broekstra. SPARQL Query Results XML Format. W3C Recommendation 15 January 2008, 2008. available at http://www.w3.org/TR/rdf-sparql-XMLres/.

Dave Beckett. RDF/XML syntax specification (revised). W3C Recommendation, February 2004.

Gabe Beged-Dov, Dan Brickley, Rael Dornfest, Ian Davis, Leigh Dodds, Jonathan Eisenzopf, David Galbraith, R.V. Guha, Ken MacLeod, Eric Miller, Aaron Swartz, and Eric van der Vlist. RDF site summary (RSS) 1.0, December 2000. Available at http://web.resource.org/rss/1.0/spec.

V. Richard Benjamins, Pompeu Casanovas, Jesús Contreras, José Manuel López Cobo, and Lisette Lemus. Iuriservice: An intelligent frequently asked questions system to assist newly appointed judges. In V.R. Benjamins, P. Casanovas, A. Gangemi, and B. Selic, editors, *Law and the Semantic Web*, LNCS, Berlin Heidelberg, 2005. Springer.

Tim Berners-Lee, Larry Masinter, and Mark McCahill. Universal Resource Locators (URL). Technical Report 1738, Internet Engineering Task Force, December 1994.

Tim Berners-Lee, Jim Hendler, and Ora Lassila. The semantic web. *Scientific American*, 2001(5), 2001. available at http://www.sciam.com/2001/0501issue/0501berners-lee.html.

Tim Berners-Lee, Roy Fielding, and Larry Masinter. Uniform Resource Identifier (URI): Generic Syntax. Technical Report 3986, Internet Engineering Task Force, June 2005. RFC 3986 (available at http://www.ietf.org/rfc/rfc3986.txt).

Tim Berners-Lee, Yuhsin Chen, Lydia Chilton, Dan Connolly, Ruth Dhanaraj, James Hollenbach, Adam Lerer, and David Sheets. Tabulator: Exploring and analyzing linked data on the Semantic Web. In Lloyd Rutledge, m.c. schraefel, Abraham Bernstein, and Duane Degler, editors, *Proceedings of the Third International Semantic Web User Interaction Workshop SWUI2006 at the International Semantic Web Conference ISWC2006*, 2006.

Tim Berners-Lee, Wendy Hall, James A. Hendler, Kieron O'Hara, Nigel Shadbolt, and Daniel J. Weitzner. A framework for web science. *Foundations and Trends in Web Science*, 1(1):1–130, 2006.

Tim Berners-Lee. Cool URIs don't change. W3C Style, 1998. available at http://www.w3.org/Provider/Style/URI.html.

Tim Berners-Lee. Notation 3 - a readable language for data on the web, 2006. available at http://www.w3.org/DesignIssues/Notation3.

Diego Berrueta and Jon Phipps. Representing classes as property values on the semantic web, 2005. W3C Working Group Note 5 April 2005, avail. at http://www.w3.org/TR/swbp-vocab-pub/.

Diego Berrueta and Jon Phipps. Best practice recipes for publishing RDF vocabularies, 2008. W3C Working Group Note 28 August 2008, avail. at http://www.w3.org/TR/swbp-vocab-pub/.

Janez Brank, Marko Grobelnik, and Dunja Mladenić. A survey of ontology evaluation techniques. In *Proceedings of 8th International Multi-Conference of the Information Society*, pages 166–169, 2005.

Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin. Namespaces in XML 1.0 (second edition), 2006. W3C Recommendation 16 August 2006, available at http://www.w3.org/TR/REC-xml-names/.

Tim Bray, Jean Paoli, C. Michael Sperberg-McQueen, Eve Maler, and Francois Yergeau. Extensible markup language (XML) 1.0 (fifth edition). W3C Recommendation 26 November 2008, 2008. available at http://www.w3.org/TR/REC-xml.

Christopher Brewster, Harith Alani, Srinandan Dasmahapatra, and Yorick Wilks. Data-driven ontology evaluation. In *Proceedings of the Language Resources and Evaluation Conference (LREC 2004)*, pages 164–168, Lisbon, Portugal, 2004. European Language Resources Association.

Dan Brickley and Libby Miller. The Friend Of A Friend (FOAF) vocabulary specification, July 2005.

Thomas Henry Briggs. *Constraint Generation and Reasoning in OWL*. PhD thesis, University of Maryland, 2008.

Saartje Brockmans and Peter Haase. A Metamodel and UML Profile for Rule-extended OWL DL Ontologies –A Complete Reference. Technical report, Universität Karlsruhe, March 2006. http://www.aifb.uni-karlsruhe.de/WBS/sbr/publications/owl-metamodeling.pdf.

Andrew Burton-Jones, Veda C. Storey, Vijayan Sugumaran, and Punit Ahluwalia. A semiotic metrics suite for assessing the quality of ontologies. *Data and Knowledge Engineering*, 55(1):84–102, October 2005.

Stefano Emilio Campanini, Paolo Castagna, and Roberto Tazzoli. Towards a semantic wiki wiki web. In Giovanni Tummarello, Christian Morbidoni, Paolo Puliti, Francesco Piazza, and Luigi Lella, editors, *Proceedings of the 1st Italian Semantic Web Workshop (SWAP2004)*, Ancona, Italy, December 2004.

Pompeu Casanovas, Núria Casellas, Marta Poblet, Joan-Josep Vallbé, York Sure, and Denny Vrandečić. Iuriservice II ontology development. In Pompeu Casanovas, editor, *Workshop on Artificial Intelligence and Law at the XXIII. World Conference of Philosophy of Law and Social Philosophy*, May 2005.

Pompeu Casanovas, Nuria Casellas, Christoph Tempich, Denny Vrandečić, and Richard Benjamins. OPJK and DILIGENT: ontology modeling in a distributed environment. *Artificial Intelligence and Law*, 15(1), 2 2007.

Werner Ceusters and Barry Smith. A realism-based approach to the evolution of biomedical ontologies. In *Proceedings of the AMIA 2006 Annual Symposium*, November 2006.

Gong Cheng, Weiyi Ge, and Yuzhong Qu. FALCONS: Searching and browsing entities on the semantic web. In *Proceedings of the the World Wide Web Conference*, 2008.

Philipp Cimiano and Johanna Völker. A framework for ontology learning and data-driven change discovery. In *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB'2005)*, 2005.

Philipp Cimiano, Günter Ladwig, and Steffen Staab. Gimme the context: Context-driven automatic semantic annotation with C-PANKOW. In Allan Ellis and Tatsuya Hagino, editors, *Proceedings of the 14th World Wide Web Conference*, pages 332 – 341, Chiba, Japan, MAY 2005. ACM Press.

James Clark and Makoto Murata. RELAX NG Specification, December 2001. OASIS committee specification.

Kendall Grant Clark, Lee Feigenbaum, and Elias Torres. Serializing SPARQL Query Results in JSON. W3C Working Group Note 18 June 2007, 2007. available at http://www.w3.org/TR/rdf-sparql-json-res/.

James Clark. XSL Transformations (XSLT). W3C Recommendation 16 November 1999, 9 1999. available at http://www.w3.org/TR/1999/REC-xslt-19991116.

John Cowan and Richard Tobin. XML information set (second edition). W3C Recommendation 4 February 2004, 2004. available at http://www.w3.org/TR/xml-infoset/.

Anne Cregan, Malgorzata Mochol, Denny Vrandečić, and Sean Bechhofer. Pushing the limits of OWL, Rules and Protégé – A simple example. In Bernardo Cuenca Grau, Ian Horrocks, Bijan Parsia, and Peter Patel-Schneider, editors, *OWL: Experiences and Directions*, Galway, Ireland, 11 2005.

Mathieu d'Aquin and Holger Lewen. Cupboard – a place to expose your ontologies to applications and the community. In *The Semantic Web: Research and Applications, 6th European Semantic Web Conference, ESWC 2009, Heraklion, Crete, Greece, May 31-June 4, 2009, Proceedings*, volume 5554 of *Lecture Notes in Computer Science*, pages 913–918. Springer, Mai 2009.

Mathieu d'Aquin, Claudio Baldassarre, Larian Gridinoc, Sofia Angeletou, Marta Sabou, and Enrico Motta. Characterizing knowledge on the semantic web with Watson. In Denny Vrandečić, Raúl García-Castro, Asunción Gómez-Pérez, York Sure, and Zhisheng Huang, editors, *Proceedings of the Workshop on Evaluation of Ontologies and Ontology-based tools, 5th International EON Workshop (EON2007) at ISWC/ASWC'07*, pages 1–10, Busan, Korea, November 2007.

Mathieu d'Aquin, Claudio Baldassarre, Laurian Gridinoc, Marta Sabou, Sofia Angeletou, and Enrico Motta. Watson: Supporting next generation semantic web applications. In *WWW/Internet conference*, Vila real, Spain, 2007.

Frank Dawson and Tim Howes. vCard MIME Directory Profile. RFC 2426, Internet Engineering Task Force, 9 1998.

Tom DeMarco. *Controlling Software Projects: Management, Measurement & Estimation*. Yourdon Press, New York, 1982.

Rose Dieng and Olivier Corby, editors. *Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management: Methods, Models, and Tools (EKAW 2000)*, volume 1937 of *Lecture Notes in Artificial Intelligence (LNAI)*, Juan-les-Pins, France, 2002. Springer.

Li Ding, Tim Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: a search and metadata engine for the semantic web. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 652–659, New York, NY, USA, 2004. ACM.

Francesco M. Donini, Daniele Nardi, and Riccardo Rosati. Description logics of minimal knowledge and negation as failure. *ACM Transactions on Computational Logic*, 3(2):177–225, 2002.

Umberto Eco. *Foucault's Pendulum*. Secker & Warburg, London, 1988.

Marc Ehrig, Peter Haase, Nenad Stojanović, and Mark Hefke. Similarity for ontologies - a comprehensive framework. In *13th European Conf. on Information Systems*, 2005.

Basil Ell. Integration of external data in semantic wikis. Master thesis, Hochschule Mannheim, December 2009.

Michael Erdmann and Rudi Studer. How to structure and access XML documents with ontologies. *Data Knowl. Eng.*, 36(3):317–335, 2001.

Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Web-scale information extraction in KnowItAll (preliminary results). In *Proceedings of the 13th International Conference on the World Wide Web (WWW 2004)*, pages 100–109, 2004.

Ronald Fagin, Joseph Halpern, Yoram Moses, and Moshe Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, MA, USA, 2003.

David C. Fallside and Priscilla Walmsley. XML schema part 0: Primer second edition, 2004. W3C Rec. 28 October 2004.

Adam Farquhar, Richard Fikes, and James Rice. The Ontolingua Server: A tool for collaborative ontology construction. In *Proceedings of the 10th Banff Knowledge Acquisition for KnowledgeBased System Workshop (KAW'95)*, Banff, Canada, November 1996.

Christine Fellbaum. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. MIT Press, May 1998.

Mariano Fernández-López and Asunción Gómez-Pérez. The integration of OntoClean in WebODE. In *Proceedings of the EON2002 Workshop at 13th EKAW*, 2002.

Mariano Fernández-López, Asunción Gómez-Pérez, Juan Pazoz Sierra, and Alejandro Pazoz Sierra. Building a chemical ontology using Methontology and the Ontology Design Environment. *IEEE Intelligent Systems*, 14(1), January/February 1999.

Roy Fielding, James Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999.

Mark S. Fox and Michael Gruninger. Enterprise modeling. *AI Magazine*, 19:109–121, Fall 1998.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison Wesley, Reading, Massachusetts, 1995.

Aldo Gangemi, Nicola Guarino, Claudio Masolo, Alessandro Oltramari, and Luc Schneider. Sweetening ontologies with DOLCE. In A. Gómez-Pérez and V. R. Benjamins, editors, *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web (EKAW 2002)*, volume 2473 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 166–181, Siguenza, Spain, 2002. Springer.

Aldo Gangemi, Carola Catenacci, Massimiliano Ciaramita, and Jens Lehmann. Ontology evaluation and validation: an integrated formal model for the quality diagnostic task. Technical report, Laboratory of Applied Ontologies – CNR, Rome, Italy, 2005. http://www.loa-cnr.it/Files/OntoEval4OntoDev_Final.pdf.

Aldo Gangemi, Carola Catenacci, Massimiliano Ciaramita, and Jos Lehman. Modelling ontology evaluation and validation. In *Proceedings of the Third European Semantic Web Conference (ESWC)*, Budva, Montenegro, 2006.

Aldo Gangemi, Carola Catenaccia, Massimiliano Ciaramita, and Jos Lehmann. Qood grid: A metaontology-based framework for ontology evaluation and selection. In Denny Vrandečić, Mari del Carmen Suárez-Figueroa, Aldo Gangemi, and York Sure, editors, *Proceedings of the 4th International Workshop on Evaluation of Ontologies for the Web (EON2006) at the 15th International World Wide Web Conference (WWW 2006)*, volume 179 of *CEUR-WS*, pages 8–15, Edinburgh, Scotland, May 2006.

Aldo Gangemi. Ontology design patterns for semantic web content. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *Proceedings of the 4th International Semantic Web Conference (ISWC2005)*, volume 3729 of *LNCS*. Springer Verlag Berlin-Heidelberg, November 2005.

Asunción Gómez-Pérez, Mariano Fernández-López, and Oscar Corcho. *Ontological Engineering*. Advanced Information and Knowlege Processing. Springer, 2003.

Asunción Gómez-Pérez. Ontology evaluation. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies, First Edition*, chapter 13, pages 251–274. Springer, 2004.

Jan Grant and Dave Beckett. RDF test cases. W3C Recommendation, February 2004.

Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. OWL 2: The next step for OWL. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):309–322, 2008.

Stephan Grimm and Boris Motik. Closed world reasoning in the semantic web through epistemic operators. In Bernardo Cuenca Grau, Ian Horrocks, Bijan Parsia, and Peter Patel-Schneider, editors, *Second International Workshop on OWL: Experiences and Directions (OWLED 2006)*, Galway, Ireland, 2005.

Benjamin Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proceedings of the Twelfth International World Wide Web Conference, WWW2003, Budapest, Hungary, 20-24 May 2003*, pages 48–57. ACM, 2003.

William E. Grosso, Henrik Eriksson, Ray W. Fergerson, Samson W. Tu, and Mark A. Musen. Knowledge modeling at the millennium: the design and evolution of PROTEGE-2000. In *Proceedings of the 12th International Workshop on Knowledge Acquisition, Modeling and Mangement (KAW-99)*, Banff, Canada, October 1999.

Thomas R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43(5/6):907–928, 1995.

Michael Grüninger and Mark S. Fox. Methodology for the design and evaluation of ontologies. In *IJCAI95 Workshop on Basic Ontological Issues in Knowledge Sharing*, Montreal, 1995.

Nicola Guarino and Christopher Welty. A formal ontology of properties. In Dieng and Corby (2002), pages 97–112.

Nicola Guarino and Christopher Welty. Evaluating ontological decisions with OntoClean. *Communications of the ACM*, 45(2):61–65, February 2002.

Nicola Guarino and Chris A. Welty. An overview of OntoClean. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies in Information Systems, First Edition*, pages 151–172. Springer, 2004.

Nicola Guarino. Ontology of information objects. FOFIS Deliverable 2, Istituto di Scienze e Tecnologie della Cognizione del Consiglio Nazionale delle Ricerche, Mar 2006.

Peter Haase and Guilin Qi. An analysis of approaches to resolving inconsistencies in DL-based ontologies. In *Proceedings of International Workshop on Ontology Dynamics (IWOD'07)*, pages 97–109, June 2007.

Peter Haase and Ljiljana Stojanović. Consistent evolution of OWL ontologies. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *Proceedings of the Second European Semantic Web Conference*, volume 3532, pages 182–197, Heraklion, Crete, Greece, 2005. Springer.

Peter Haase and York Sure. Usage tracking for ontology evolution. SEKT Deliverable D3.2.1, Institute AIFB, University of Karlsruhe, June 2005.

Peter Haase, Frank van Harmelen, Zhisheng Huang, Heiner Stuckenschmidt, and York Sure. A framework for handling inconsistency in changing ontologies. In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors, *Proceedings of the Fourth International Semantic Web Conference (ISWC2005)*, volume 3729 of *LNCS*, pages 353–367. Springer, November 2005.

Andreas Harth, Aidan Hogan, Jürgen Umbrich, and Stefan Decker. SWSE: Object before documents! In Amit P. Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Tim Finin, and Krishnaprasad Thirunarayan, editors, *Proceedings of the 7th International Semantic Web Conference, Semantic Web Challenge 2008*, volume 5318 of *LNCS*, Karlsruhe, Germany, October 2009. Springer.

Jens Hartmann, York Sure, Peter Haase, Raul Palma, and Mari del Carmen Suárez-Figueroa. OMV – Ontology Metadata Vocabulary. In Chris Welty, editor, *Ontology Patterns for the Semantic Web Workshop*, Galway, Ireland, 2005.

Patrick Hayes. RDF Semantics. W3C Recommendation 10 February 2004, 2004. available at http://www.w3.org/TR/rdf-mt/.

Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th International Conference on Computational Linguistics*, pages 539–545, 1992.

Pascal Hitzler and Denny Vrandečić. Resolution-based approximate reasoning for OWL DL. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *Proceedings of the Fourth International Semantic Web Conference (ISWC'05)*, volume 3729 of *LNCS*. Springer Verlag Berlin-Heidelberg, November 2005.

Pascal Hitzler, Markus Krotzsch, and Sebastian Rudolph. *Semantic Web Foundations*. Springer, 2009.

Jerry R. Hobbs and Feng Pan. An ontology of time for the semantic web. *ACM Transactions on Asian Language Information Processing (TALIP)*, 3(1):66–85, 2004.

Masahiro Hori, Jérôme Euzenat, and Peter F. Patel-Schneider. OWL Web Ontology Language XML presentation syntax, 2003. W3C Note 11 June 2003.

Matthew Horridge, Nick Drummond, John Goodwin, Alan Rector, Robert Stevens, and Hai Wang. The manchester owl syntax. In *OWLED2006 Second Workshop on OWL Experiences and Directions*, Athens, GA, USA, 2006.

Matthew Horridge. The Protégé OWL unit test framework, 2005. Website at `http://www.co-ode.org/downloads/owlunittest/`.

Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL Entailment to Description Logic Satisfiability. *Journal of Web Semantics*, 1(4):7–26, 2004.

Ian Horrocks, Frank van Harmelen, Peter Patel-Schneider, Tim Berners-Lee, Dan Brickley, Dan Connolly, Mike Dean, Stefan Decker, Dieter Fensel, Richard Fikes, Pat Hayes, Jeff Heflin, James A. Hendler, Ora Lassila, Deborah L. McGuinness, and Lynn Andrea Stein. DAML+OIL (March 2001), 2001. Joint Committee, http://www.daml.org/2001/03/daml+oil-index.

Ian Horrocks, Peter Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: a semantic web rule language combining OWL and RuleML, 2003.

ISO 15924. Codes for the representation of names of scripts. Technical report, International Standard ISO, 2004.

ISO 2108. Information and documentation – International standard book number (ISBN). Technical report, International Standard ISO, 2005.

ISO 24824. ISO/IEC 24824-1 (Fast Infoset). Technical report, International Standard ISO, 2007.

ISO 3166. Codes for the representation of names of countries and their subdivisions. Technical report, International Standard ISO, 1999.

ISO 639-1. Codes for the representation of names of languages – Part 1: Alpha-2 code. Technical report, International Standard ISO, 2002.

ISO 639-2. Codes for the representation of names of languages – Part 2: Alpha-3 code. Technical report, International Standard ISO, 1998.

ISO 8000-102. Data quality – Part 102: Master data: Exchange of characteristic data: Vocabulary. Technical report, International Standard ISO, 2009.

ISO 8000-110. Data quality – Part 110: Master data: Exchange of characteristic data: Syntax, semantic encoding, and conformance to data specification. Technical report, International Standard ISO, 2009.

ISO 8000-120. Data quality – Part 120: Master data: Exchange of characteristic data: Provenance. Technical report, International Standard ISO, 2009.

ISO 8000-130. Data quality – Part 130: Master data: Exchange of characteristic data: Accuracy. Technical report, International Standard ISO, 2009.

ISO 8000-140. Data quality – Part 140: Master data: Exchange of characteristic data: Completeness. Technical report, International Standard ISO, 2009.

Ian Jacobs and Norman Walsh. Architecture of the World Wide Web Vol. 1, 2004. W3C Recommendation 15 December 2004, avail. at http://www.w3.org/TR/webarch/.

Aleks Jakulin and Dunja Mladenić. Ontology grounding. In *Proceedings of 8th International Multi-Conference Information Society IS-2005*, pages 170–173, 2005.

Qiu Ji, Peter Haase, Guilin Qi, Pascal Hitzler, and Steffen Stadtmüller. RaDON – repair and diagnosis in ontology networks. In Eero Hyvönen, Robert Stevens, Siegfried Handschuh, Peter Haase, Brian Davis, Kim Viljanen, Christian Meilicke, Enrico Motta, Wolfgang Nejdl, and Heiner Stuckenschmidt, editors, *Proceedings of the 6th European Semantic Web Conference (ESWC 2009)*, pages 863–867, Heraklion, Greece, June 2009.

Joseph M. Juran and A. Blanton Godfrey. *Juran's Quality Handbook*. McGraw-Hill, 5th edition, 1999.

Aditya Kalyanpur, Bijan Parsia, Evren Sirin, Bernardo Cuenca Grau, and James Hendler. Swoop: A web ontology editing browser. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 4(2):144–153, June 2006.

Brian W. Kernighan and P.J. Plauger. *The Elements of Programming Style*. McGraw-Hill, 2nd edition, 1978.

Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42:741–843, 1995.

Graham Klyne and Jeremy Carroll. Resource Description Framework (RDF): Concepts and abstract syntax. W3C Recommendation 10 February 2004, 2004.

Andrew Koenig. Patterns and antipatterns. *Journal of Object-Oriented Programming*, 8(1):46–48, March 1995.

Konstantinos Kotis, George A. Vouros, and Jerónimo P. Alonso. HCOME: A tool-supported methodology for engineering living ontologies. *Semantic Web and Databases*, pages 155–166, 2005.

Chrysovalanto Kousetti, David Millard, and Yvonne Howard. A study of ontology convergence in a semantic wiki. In Ademar Aguiar and Mark Bernstein, editors, *WikiSym 2008*, September 2008.

Markus Krötzsch, Denny Vrandečić, and Max Völkel. Wikipedia and the semantic web – the missing links. In *Proceedings of Wikimania 2005 – The First International Wikimedia Conference*, Frankfurt, Germany, July 2005. Wikimedia Foundation.

Markus Krötzsch, Pascal Hitzler, Denny Vrandečić, and Michael Sintek. How to reason with OWL in a logic programming system. In Thomas Eiter, Enrico Franconi, Ralph Hodgson, and Susie Stephens, editors, *Proceedings of the 2nd International Conferenc on Rules and Rule Markup Languages for the Semantic Web (RuleML2006)*, pages 17–26, Athens, GA, USA, 11 2006. IEEE Computer Society.

Markus Krötzsch, Denny Vrandečić, and Max Völkel. Semantic MediaWiki. In Isabel Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Mike Uschold, and Lora Aroyo, editors, *Proceedings of the 5th International Semantic Web Conference (ISWC2006)*, volume 4273 of *LNCS*, pages 935–942, Athens, GA, USA, November 2006. Springer.

Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. Conjunctive queries for a tractable fragment of OWL 1.1. In Karl Aberer, Key-Sun Choi, and Natasha Noy, editors, *Proc. 6th Int. Semantic Web Conf. (ISWC'07)*. Springer, 2007.

Markus Krötzsch, Sebastian Schaffert, and Denny Vrandečić. Reasoning in semantic wikis. In Grigoris Antoniou, Uwe Assmann, Cristina Baroglio, Stefan Decker, Nicola Henze, Paula-Lavinia Patranjan, and Robert Tolksdorf, editors, *Proceedings of the 3rd Reasoning Web Summer School*, volume 4636 of *LNCS*, pages 310–329, Dresden, Germany, September 2007. Springer.

Markus Krötzsch, Denny Vrandečić, Max Völkel, Heiko Haller, and Rudi Studer. Semantic wikipedia. *Journal of Web Semantics*, 5:251–261, September 2007.

Joey Lam. *Methods for resolving inconsistencies in ontologies*. PhD thesis, University of Aberdeen, 2007.

Cindy Lauper and Rob Hyman. Time after time. In *She's So Unusual*. Epic Records, 1983.

Douglas B. Lenat. CYC: A large-scale investment in knowledge infrastructure. *Communications of ACM*, 38(11):32–38, November 1995.

Vladimir Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

Holger Lewen, Kaustubh Supekar, Natalya F. Noy, and Mark A. Musen. Topic-specific trust and open rating systems: An approach for ontology evaluation. In *Proceedings of the 4th International Workshop on Evaluation of Ontologies for the Web (EON2006) at the 15th International World Wide Web Conference (WWW 2006)*, Edinburgh, UK, Mai 2006.

Rhys Lewis. Dereferencing HTTP URIs, 2007. Draft TAG Finding 31 August 2007, avail. at http://www.w3.org/2001/tag/doc/httpRange-14/2007-08-31/HttpRange-14.html.

Uta Lösch, Sebastian Rudolph, Denny Vrandečić, and Rudi Studer. Tempus fugit – towards an ontology update language. In Lora Aroyo et al., editors, *Proceedings of the 6th European Semantic Web Conference (ESWC 2009)*, volume 5554 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag Berlin Heidelberg, 6 2009.

Adolfo Lozano-Tello and Asunción Gómez-Pérez. OntoMetric: A method to choose the appropriate ontology. *Journal of Database Management Special Issue on Ontological analysis, Evaluation, and Engineering of Business Systems Analysis Methods*, 15(2), 2004.

Adolfo Lozano-Tello. *Métrica de idoneidad de ontologías*. PhD thesis, Universidad de Extremadura, 2002.

Alexander Maedche and Steffen Staab. Measuring similarity between ontologies. In *Proc. Of the European Conference on Knowledge Acquisition and Management - EKAW-2002. Madrid, Spain, October 1-4, 2002*, volume 2473 of *LNCS/LNAI*. Springer, 2002.

René Magritte. The treachery of images, 1929.

Deborah L. McGuiness. Ontologies come of age. In Dieter Fensel, Jim Hendler, Henry Lieberman, and Wolfgang Wahlster, editors, *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press, 2003.

Alistair Miles and Sean Bechhofer. SKOS Simple Knowledge Organization System Reference, 2009. W3C Recommendation 18 August 2009, available at http://www.w3.org/TR/skos-reference/.

Malgorzata Mochol, Anne Cregan, Denny Vrandečić, and Sean Bechhofer. Exploring owl and rules: a simple teaching case. *International Journal of Teaching and Case Studies (IJTCS)*, 1(4):299–318, 11 2008.

Boris Motik and Ian Horrocks. Problems with OWL syntax. In *OWLED2006 Second Workshop on OWL Experiences and Directions*, Athens, GA, USA, 2006.

Boris Motik, Denny Vrandečić, Pascal Hitzler, York Sure, and Rudi Studer. dlpconvert - Converting OWL DLP statements to logic programs. In Heiner Stuckenschmidt, editor, *Poster and Demonstration Proceedings of the ESWC 2005*, 5 2005.

Boris Motik, Peter F. Patel-Schneider, and Bernardo Cuenca Grau. OWL2 Web Ontology Language: Direct semantics, 2009. W3C Recommendation 27 October 2009, available at http://www.w3.org/TR/owl2-semantics/.

Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL2 Web Ontology Language: Structural specification and functional-style syntax, 2009. W3C Recommendation 27 October 2009, available at http://www.w3.org/TR/owl2-syntax/.

Boris Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Universität Fridericiana zu Karlsruhe (TH), Germany, 2006.

Boris Motik. On the properties of metamodeling in OWL1. *Journal of Logic and Computation*, 17(4):617–637, 2007.

Lyndon J. B. Nixon and Elena Paslaru Bontas Simperl. Makna and MultiMakna: towards semantic and multimedia capability in wikis for the emerging web. In Sebastian Schaffert and York Sure, editors, *Proc. Semantics 2006*. Österreichische Computer Gesellschaft, 2006.

Natalya F. Noy, R. Fergerson, and Mark Musen. The knowledge model of Protégé-2000: Combining interoperability and flexibility. In Dieng and Corby (2002), pages 17–32.

Daniel Oberle, Steffen Lamparter, Stephan Grimm, Denny Vrandečić, Steffen Staab, and Aldo Gangemi. Towards ontologies for formalizing modularization and communication in large software systems. *Applied Ontology*, 1(2):163–202, 2006.

Leo Obrst, Werner Ceusters, Inderjeet Mani, Steve Ray, and Barry Smith. The evaluation of ontologies. In Christopher J.O. Baker and Kei-Hoi Cheung, editors,

*Revolutionizing Knowledge Discovery in the Life Sciences*, chapter 7, pages 139–158. Springer, 2007.

Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Debugging OWL ontologies. In *Proceedings of the 14th World Wide Web Conference (WWW2005)*, Chiba, Japan, May 2005.

Blaise Pascal. *Pensées*. 1670.

Chintan Patel, Kaustubh Supekar, Yugyung Lee, and E. K. Park. OntoKhoj: a semantic web portal for ontology searching, ranking and classification. In *Proceedings of Fifth ACM International Workshop on Web information and data management*, pages 58–61, New York, NY, USA, 2003.

Peter F. Patel-Schneider and Boris Motik. OWL2 Web Ontology Language: Mapping to RDF graphs, 2009. W3C Recommendation 27 October 2009, available at http://www.w3.org/TR/owl-mapping-to-rdf/.

Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax, 2004. W3C Recommendation 10 February 2004, available at http://www.w3.org/TR/2004/REC-owl-semantics-20040210/.

Addison Phillips and Mark Davis. Matching of Language Tags. Technical Report RFC 4647, Internet Engineering Task Force, September 2006. RFC 4647 (available at http://www.ietf.org/rfc/rfc4647.txt).

Addison Phillips and Mark Davis. Tags for Identifying Languages. Technical Report RFC 4646, Internet Engineering Task Force, September 2006. RFC 4646 (available at http://www.ietf.org/rfc/rfc4646.txt).

Robert M. Pirsig. *Zen and the Art of Motorcycle Maintenance: An Inquiry into Values*. Bantam, 1984.

Plato. *Phaedrus*. Oxford University Press, 370 BC. translated by Benjamin Jowett.

Jeffery T. Pollock. *Semantic Web for Dummies*. Wiley, 2009.

Robert Porzel and Rainer Malaka. A task-based approach for ontology evaluation. In Paul Buitelaar, Siegrfried Handschuh, and Bernardo Magnini, editors, *Proceedings of ECAI 2004 Workshop on Ontology Learning and Population*, Valencia, Spain, August 2004.

Neil Postman. Language education in a knowledge context. *ETC: A Review of General Semantics*, 37(1):25–37, 1980.

Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. W3C Recommendation 15 January 2008, January 2008. available at http://www.w3.org/TR/rdf-sparql-query/.

Charbel Rahhal, Hala Skaf-Molli, Pascal Molli, and Stéphane Weiss. Multi-synchronous collaborative semantic wikis. In Gottfried Vossen, Darrell D. E. Long, and Jeffrey Xu Yu, editors, *Proceedings of the International Conference on Web Information Systems Engineering (Wise 2009)*, volume 5802 of *LNCS*, Poznan, Poland, October 2009.

Eric Raymond. *The Cathedral and the Bazaar*. O'Reilly, Sebastapol, CA, revised edition, January 2001. Available at http://catb.org/~esr/writings/cathedral-bazaar/.

Alan Rector. Representing specified values in OWL: "value partitions" and "value sets". W3C Working Group Note, May 2005. available at http://www.w3.org/TR/swbp-specified-values/.

Sebastian Rudolph, Johanna Völker, and Pascal Hitzler. Supporting lexical ontology learning by relational exploration. In Uta Priss, Simon Polovina, and Richard Hill, editors, *Proceedings of the Conference on Conceptual Structures: Knowledge Architectures for Smart Applications (ICCS 2007)*, volume 4604 of *LNAI*, pages 488–491, Sheffield, UK, July 2007. Springer.

Marta Sabou, Jorge Gracia, Sofia Angeletou, Mathieu d'Aquin, and Enrico Motta. Evaluating the semantic web: A task-based approach. In Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC2007+ASWC2007)*, volume 4825 of *Lecture Notes in Computer Science*, pages 423–437, Busan, South Korea, November 2007. Springer.

Leo Sauermann and Richard Cyganiak. Cool URIs for the semantic web, March 2008. W3C Interest Group Note, available at http://www.w3.org/TR/cooluris/.

Andrea Schaerf. Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, 13(2):141–176, September 1994.

Sebastian Schaffert, Julia Eder, Szaby Grünwald, Thomas Kurz, Mihai Radulescu, Rolf Sint, and Stephanie Stroka. KiWi - a platform for semantic social software. In Christoph Lange, Sebastian Schaffert, Hala Skaf-Molli, and Max Völkel, editors, *4th Workshop on Semantic Wikis (SemWiki2009) at the European Semantic Web*

*Conference (ESWC 2009*, volume 646 of *CEUR-WS*, Herakleion, Greece, June 2009.

Sebastian Schaffert. IkeWiki: A semantic wiki for collaborative knowledge management. In Robert Tolksdorf, Elena Simperl, and Klaus Schild, editors, *1st International Workshop on Semantic Technologies in Collaborative Applications (STICA 2006) at the 15th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2006)*, pages 388–396, June 2006.

Mathias Schindler and Denny Vrandečić. Introducing new features to Wikipedia: Case studies for Web Sience. *IEEE Intelligent Systems*, 2010. to appear.

Toby Segaran, Jamie Taylor, and Colin Evans. *Programming the Semantic Web*. O'Reilly, July 2009.

William Shakespeare. *Romeo and Juliet*. John Danter, London, 1597.

Evren Sirin and Jiao Tao. Towards integrity constraints in OWL. In Peter Patel-Schneider and Rinke Hoekstra, editors, *Proceedings of the Workshop OWL Experiences and Directions 2009 (OWLED 2009) at the 8th International Semantic Web Conference (ISWC 2009)*, October 2009.

Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, June 2007.

Barry Smith and Christopher Welty. FOIS introduction: Ontology—towards a new synthesis. In *Proceedings of the 2nd international conference on Formal Ontology in Information Systems 2001 (FOIS 2001)*, pages III–IX, Ogunquit, ME, October 2001. ACM Press.

Michael K. Smith, Chris Welty, and Deborah McGuinness. OWL Web Ontology Language Guide, February 2004. W3C Recommendation 10 February 2004, available at `http://www.w3.org/TR/owl-guide/`.

Adam Souzis. Building a semantic wiki. *IEEE Intelligent Systems*, 20(5):87–91, September / October 2005.

Peter Spyns. EvaLexon: Assessing triples mined from texts. Technical Report 09, Star Lab, Brussels, Belgium, May 2005.

Steffen Staab, Michael Erdmann, and Alexander Maedche. Engineering ontologies using semantic patterns. In Alun Preece and Daniel O'Leary, editors, *Proceedings of the Workshop on E-Business & the Intelligent Web at the International Joint Conference on Artificial Intelligence (IJCAI 2001)*, Seattle, WA, August 2001.

Ljiljana Stojanović. *Methods and Tools for Ontology Evolution*. PhD thesis, Universität Karlsruhe (TH), Karlsruhe, Germany, August 2004.

Peter F. Strawson. Entity and identity. In Hywel David Lewis, editor, *Contemporary British Philosophy Fourth Series*. Allen and Unwin, London, England, 1976.

York Sure and Rudi Studer. On-To-Knowledge methodology. In John Davies, Dieter Fensel, and Frank van Harmelen, editors, *On-To-Knowledge: Semantic Web enabled Knowledge Management*, chapter 3, pages 33–46. J. Wiley and Sons, November 2002.

York Sure, Jürgen Angele, and Steffen Staab. OntoEdit: Multifaceted inferencing for ontology engineering. In Stefano Spaccapietra, Salvatore March, and Karl Aberer, editors, *Journal on Data Semantics I*, volume 2800 of *LNCS*, pages 128–152. Springer, October 2003.

York Sure, Christoph Tempich, and Denny Vrandečić. SEKT methodology: Final description including guidelines, best practices, and lessons learned. SEKT Deliverable 7.2.2, Institute AIFB, University of Karlsruhe, January 2007.

Vojtech Svatek. Design Patterns for Semantic Web Ontologies: Motivation and Discussion. In Witold Abramowicz, editor, *Proceedings of the 7th International Conference on Business Information Systems (BIS 2004)*, Poznan, Poland, April 2004.

John Swartzwelder. Homer the Smithers. *The Simpsons*, 7(17), February 1996.

Samir Tartir, I. Budak Arpinar, Michael Moore, Amit P. Sheth, and Boanerges Aleman-Meza. OntoQA: Metric-based ontology quality analysis. In Doina Caragea, Vasant Honavar, Ion Muslea, and Raghu Ramakrishnan, editors, *Proceedings of IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources at Fifth IEEE International Conference on Data Mining (ICDM 2005)*, pages 45–53, November 2005.

Christoph Tempich, Helena Sofia Pinto, York Sure, and Steffen Staab. An argumentation ontology for DIstributed, Loosely-controlled and evolvInG Engineering processes of oNTologies (DILIGENT). In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *Proceedings of the Second European Semantic Web Conference (ESWC 2005)*, volume 3532 of *LNCS*, pages 241–256, Heraklion, Greece, May / June 2005.

Ivan Terziev, Atanas Kiryakov, and Dimitar Manov. Base upper-level ontology (BULO) guidance. SEKT deliverable 1.8.1, Ontotext Lab, Sirma AI EAD (Ltd.), July 2005.

Barbara Tillett. What is FRBR? – A conceptual model for the bibliographic universe. *The Australian Library Journal*, 54(1), February 2005.

Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In Ulrich Furbach and Natarajan Shankar, editors, *Proceedings of the Third International Joint Conference on Automated Reasoning (IJCAR 2006)*, volume 4130 of *LNAI*, pages 292–297, Seattle, WA, August 2006. Springer.

UN M.49. Standard Country or Area Codes for Statistical Use, Revision 4. Technical Report 98.XVII.9, United Nations Statistics Division UNSD, 1998.

The Unicode Consortium. *The Unicode Standard, Version 5.0.0.* Addison-Wesley, Boston, MA, November 2006.

Michael Uschold and Michael Gruninger. Ontologies and semantics for seamless connectivity. *SIGMOD Record*, 33(4):58–64, December 2004.

Paola Velardi, Roberto Navigli, Alessandro Cucchiarelli, and Francesca Neri. Evaluation of OntoLearn, a methodology for automatic population of domain ontologies. In Paul Buitelaar, Philipp Cimiano, and Bernardo Magnini, editors, *Ontology Learning from Text: Methods, Applications and Evaluation*, volume 123 of *Frontiers in Artificial Intelligence and Applications*, pages 92–106. IOS Press, July 2005.

Jeffrey Voas. Software's secret sauce: The "-ilities". *IEEE Software*, 21(6):14–15, November / December 2004.

Max Völkel, Markus Krötzsch, Denny Vrandečić, Heiko Haller, and Rudi Studer. Semantic Wikipedia. In Les Carr, David De Roure, Arun Iyengar, Carole A. Goble, and Michael Dahlin, editors, *Proceedings of the 15th international conference on World Wide Web (WWW2006)*, pages 491–495, Edinburgh, Scotland, May 2006. ACM.

Johanna Völker, Denny Vrandečić, and York Sure. Automatic evaluation of ontologies (AEON). In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *Proceedings of the Fourth International Semantic Web Conference (ISWC 2005)*, volume 3729 of *LNCS*, pages 716–731, Galway, Ireland, November 2005. Springer.

Johanna Völker, Denny Vrandečić, York Sure, and Andreas Hotho. Learning disjointness. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, *Proceedings of the 4th European Semantic Web Conference (ESWC 2007)*, volume 4519 of *LNCS*, pages 175–189, Innsbruck, Austria, June 2007. Springer.

Johanna Völker, Denny Vrandečić, Andreas Hotho, and York Sure. AEON - an approach to the automatic evaluation of ontologies. *Applied Ontology*, 3(1-2):41–62, January 2008.

Jakob Voss. Collaborative thesaurus tagging the Wikipedia way. *CoRR*, abs/cs/0604036, April 2006. Available at http://arxiv.org/abs/cs/0604036.

Denny Vrandečić, H. Sofia Pinto, York Sure, and Christoph Tempich. The DILIGENT knowledge processes. *Journal of Knowledge Management*, 9(5):85–96, October 2005.

Denny Vrandečić and Aldo Gangemi. Unit tests for ontologies. In Mustafa Jarrar, Claude Ostyn, Werner Ceusters, and Andreas Persidis, editors, *Proceedings of the 1st International Workshop on Ontology content and evaluation in Enterprise (OntoContent 2006) at On the Move Federated Conferences (OTM2006)*, volume 4278 of *LNCS*, pages 1012–1020, Montpellier, France, October 2006. Springer.

Denny Vrandečić and Markus Krötzsch. Reusing ontological background knowledge in semantic wikis. In Max Völkel, Sebastian Schaffert, and Stefan Decker, editors, *Proceedings of the 1st Workshop on Semantic Wikis – From Wikis to Semantics (SemWiki2006) at the 3rd European Semantic Web Conference (ESWC 2006)*, volume 206 of *CEUR-WS*, June 2006.

Denny Vrandečić and York Sure. How to design better ontology metrics. In Enrico Franconi, Wolfgang May, and Michael Kifer, editors, *Proceedings of the 4th European Semantic Web Conference (ESWC 2007)*, volume 4519 of *LNCS*, pages 311–325, Innsbruck, Austria, June 2007. Springer.

Denny Vrandečić, Mari del Carmen Suárez-Figueroa, Aldo Gangemi, and York Sure, editors. *Proceedings of the 4th International Workshop on Evaluation of Ontologies for the Web (EON2006) at the 15th International World Wide Web Conference (WWW 2006)*, volume 179 of *CEUR-WS*, Edinburgh, Scotland, May 2006.

Denny Vrandečić, York Sure, and Christoph Tempich. SEKT methodology: Initial lessons learned and tool design. SEKT Deliverable 7.2.1, Institute AIFB, University of Karlsruhe, January 2006.

Denny Vrandečić, Johanna Völker, Peter Haase, Duc Thanh Tran, and Philipp Cimiano. A metamodel for annotations of ontology elements in OWL DL. In York Sure, Saartje Brockmans, and Jürgen Jung, editors, *Proceedings of the Second Workshop on Ontologies and Meta-Modeling*, Karlsruhe, Germany, October 2006. GI Gesellschaft für Informatik.

Denny Vrandečić, Raúl García-Castro, Asunción Gómez-Pérez, York Sure, and Zhisheng Huang, editors. *Proceedings of the 5th International Workshop on Evaluation of Ontologies for the Web (EON2007) at the 6th International Semantic Web Conference (ISWC 2007)*, volume 329 of *CEUR-WS*, Busan, South Korea, November 2007.

Denny Vrandečić, York Sure, Raul Palma, and Francisco Santana. Ontology repositories and content evaluation. Knowledge Web Deliverable D1.2.10v2, Institute AIFB, University of Karlsruhe, June 2007.

Denny Vrandečić, Frank Dengler, Sebastian Rudholph, and Michael Erdmann. RDF syntax normalization using XML validation. In Lalana Kagal, Ora Lassila, and Tim Finin, editors, *Proceedings of the Workshop Semantics for the Rest of Us 2009 (SemRUs 2009) at the 8th International Semantic Web Conference (ISWC 2009)*, Chantilly, VA, November 2009.

Denny Vrandečić. Explicit knowledge engineering patterns with macros. In Chris Welty and Aldo Gangemi, editors, *Proceedings of the Ontology Patterns for the Semantic Web Workshop (OPSW2005) at the 8th International Semantic Web Conference (ISWC 2005)*, Galway, Ireland, November 2005.

Denny Vrandečić. Knowledge leveraging and repair – early prototypes. ACTIVE Deliverable D1.4.1, Institute AIFB, University of Karlsruhe, March 2009.

Denny Vrandečić. Ontology evaluation. In Rudi Studer and Steffen Staab, editors, *Handbook on Ontologies, 2nd edition*, chapter 13. Springer, August 2009.

Denny Vrandečić. Towards automatic content quality checks in semantic wikis. In Mark Greaves, Li Ding, Jie Bao, and Uldis Bojars, editors, *Social Semantic Web: Where Web 2.0 Meets Web 3.0*, AAAI Spring Symposium 2009, Stanford, CA, March 2009. AAAI Press.

Laurence Wachowski and Paul Wachowski. *Matrix Revolutions*. Warner Bros., USA, November 2003.

Taowei David Wang and Bijan Parsia. Ontology performance profiling and model examination: First steps. In Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC2007+ASWC2007)*, volume 4825 of *Lecture Notes in Computer Science*, pages 595–608, Busan, South Korea, November 2007. Springer.

Taowei David Wang, Bijan Parsia, and James A. Hendler. A survey of the web ontology landscape. In Isabel Cruz, Stefan Decker, Dean Allemang Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo, editors, *Proceedings of the Fifth International Semantic Web Conference (ISWC'06)*, volume 4273 of *Lecture Notes in Computer Science*, pages 682–694, Athens, Georgia, November 2006. Springer.

Roger Waters and David Gilmour. Wish You Were Here. In *Pink Floyd – Wish You Were Here*. Harvest Records, September 1975.

Chris Welty. OntOWLClean: Cleaning OWL ontologies with OWL. In Brandon Bennet and Christiane Fellbaum, editors, *Proceedings of the Fourth International Conference on Formal Ontologies in Information Systems (FOIS 2006)*, volume 150 of *Frontiers in Artificial Intelligence and Applications*, Baltymore, MD, November 2006. IOS Press.

Takashi Yamauchi. The semantic web and human inference: A lesson from cognitive science. In Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC2007+ASWC2007)*, volume 4825 of *LNCS*, pages 609–622, Busan, South Korea, November 2007. Springer.

Jonathan Yu, James A. Thom, and Audrey Tam. Ontology evaluation using Wikipedia categories for browsing. In Alberto H. F. Laender, André O. Falcão, Øystein Haug Olsen, Mário J. Silva, Ricardo Baeza-Yates, Deborah L. McGuinness, and Bjorn Olstad, editors, *Proceedings of the ACM Sixteenth Conference on Information and Knowledge Management (CIKM)*, pages 223–232, Lisboa, Portugal, November 2007. ACM.

Evgeny Zolin. Complexity of reasoning in description logics. `http://www.cs.man.ac.uk/~ezolin/dl/`, 2010. Last accessed: 14 January 2010.

# Full Table of Contents

**Acknowledgements** **5**

**Abstract** **7**

**I Foundations** **11**

**1 Introduction** **13**
1.1 Motivation . . . 14
1.1.1 Advantages of better ontologies . . . 15
1.1.2 Increasing ontology availability . . . 15
1.1.3 Lower maintenance costs . . . 16
1.2 Contribution . . . 16
1.2.1 A framework for ontology evaluation . . . 16
1.2.2 Methods for ontology evaluation . . . 17
1.2.3 Implementation . . . 17
1.3 Readers' guide . . . 18
1.3.1 Foundations . . . 18
1.3.2 Aspects . . . 19
1.3.3 Application . . . 20
1.4 Relation to previous publications . . . 20

**2 Terminology and Preliminaries** **23**
2.1 Ontologies . . . 23
2.2 Axioms . . . 25
2.2.1 Facts . . . 25
2.2.2 Class axioms . . . 26
2.2.3 Property axioms . . . 28
2.2.4 Annotations . . . 29
2.3 Entities . . . 29
2.3.1 Individuals . . . 30
2.3.2 Classes . . . 30
2.3.3 Properties . . . 30
2.3.4 Ontologies . . . 31
2.4 Semantics . . . 31

**3 Framework** **37**
3.1 Overview . . . 37
3.2 Meta-ontology . . . 39
3.2.1 Reifying ontologies . . . 39
3.2.2 Reifying URIs . . . 41
3.2.3 Advantages of a meta-ontology . . . 41
3.3 Types of ontologies . . . 42
3.3.1 Terminological ontology . . . 42
3.3.2 Knowledge base . . . 43
3.3.3 Semantic spectrum . . . 44
3.3.4 Classification example . . . 47
3.4 Limits . . . 48
3.5 Conceptualizations . . . 49
3.6 Criteria . . . 53
3.6.1 Accuracy . . . 56
3.6.2 Adaptability . . . 56
3.6.3 Clarity . . . 57
3.6.4 Completeness . . . 57
3.6.5 Computational efficiency . . . 58
3.6.6 Conciseness . . . 58
3.6.7 Consistency . . . 59
3.6.8 Organizational fitness . . . 59
3.7 Methods . . . 60
3.8 Aspects . . . 61

**II Aspects** **63**

**4 Vocabulary** **65**
4.1 URI references . . . 65
4.1.1 Linked data . . . 66
4.1.2 Hash vs slash . . . 70
4.1.3 Opaqueness of URIs . . . 72
4.1.4 URI reuse . . . 73
4.1.5 URI declarations and punning . . . 75
4.2 Literals . . . 75
4.2.1 Typed literals and datatypes . . . 76
4.2.2 Language tags . . . 78
4.2.3 Labels and comments . . . 80
4.3 Blank nodes . . . 81

**5 Syntax** **83**
5.1 Syntactic comments . . . 84
5.2 Qualified names . . . 85
5.3 XML validation . . . 85
5.3.1 Example . . . 87
5.3.2 Motivation . . . 89
5.3.3 Normalizing the serialization . . . 92
5.3.4 Creation of compliant schemas . . . 93
5.3.5 Implementation . . . 95
5.3.6 Related approaches . . . 96
5.3.7 Open questions for XML schema-based RDF validation . . . 98

**6 Structure** **99**
6.1 Structural metrics in practice . . . 100
6.1.1 Maximum depth of the taxonomy . . . 102
6.1.2 Class / relation ratio . . . 104
6.1.3 Relationship richness . . . 104
6.1.4 Semantic similarity measure . . . 106
6.2 SPARQL for finding patterns . . . 107
6.2.1 Translating patterns from OWL to RDF . . . 107
6.2.2 Querying with SPARQL . . . 110
6.2.3 Ontology normalization . . . 110
6.2.4 Incomplete searches . . . 111
6.2.5 Querying for anti-patterns . . . 113
6.3 The AEON approach . . . 115
6.3.1 OntoClean in theory . . . 117
6.3.2 OntoClean meta-properties . . . 118
6.3.3 OntoClean constraints . . . 119
6.3.4 Constraint checking . . . 120
6.3.5 Analysis and Examples . . . 122

**7 Semantics** **127**
7.1 Normalization . . . 128
7.1.1 First normalization . . . 130
7.1.2 Second normalization . . . 131
7.1.3 Third normalization . . . 132
7.1.4 Fourth normalization . . . 133
7.1.5 Fifth normalization . . . 134
7.1.6 Examples of normalization . . . 134
7.2 Stability . . . 136
7.3 Language completeness . . . 141

**8 Representation** **143**
8.1 Ontological metrics . . . 143
8.2 Maximum depth of the taxonomy . . . 145
8.3 Class / relation ratio . . . 146
8.4 Relationship richness . . . 147
8.5 Semantic similarity measure . . . 149

**9 Context** **151**
9.1 Unit tests . . . 152
9.1.1 Formalized competency questions . . . 154
9.1.2 Affirming derived knowledge . . . 156
9.1.3 Asserting agnosticism . . . 158
9.2 Increasing expressivity for consistency checking . . . 158
9.2.1 Expressive consistency checks . . . 159
9.2.2 Consistency checking with rules . . . 160
9.2.3 Use of autoepistemic operators . . . 161
9.2.4 Domain and ranges as constraints . . . 163

**III Application** **165**

**10 Collaborative ontology evaluation in Semantic MediaWiki** **167**
10.1 Annotation of wiki pages . . . 169
10.1.1 Content structuring in MediaWiki . . . 169
10.1.2 Semantic annotations in SMW . . . 170
10.1.3 Mapping to OWL . . . 173
10.2 Exploiting semantics . . . 174
10.2.1 Browsing . . . 174
10.2.2 Querying . . . 175
10.2.3 Giving back to the Web . . . 178
10.3 Related systems . . . 179
10.4 Collaborative ontology evaluation . . . 180
10.4.1 Concept cardinality . . . 180
10.4.2 Class disjointness . . . 181
10.4.3 Property cardinality constraints . . . 182
10.4.4 Social and usability aspects . . . 183

**11 Related work** **185**
11.1 Frameworks and aspects . . . 185
11.2 Methods and approaches . . . 190
11.2.1 Golden standard – similarity-based approaches . . . 192

11.2.2 Task-based evaluations . . . . . 193
11.2.3 Data-driven evaluation – fitting the data set . . . . . 194
11.3 Watson corpus . . . . . 196

**12 Conclusions** **197**
12.1 Achievements . . . . . 198
12.2 Open questions . . . . . 199

**IV Appendix** **201**

**List of Methods** **203**

**List of Tables** **205**

**List of Figures** **207**

**Bibliography** **209**

**Full Table of Contents** **230**