Karlsruhe Institute of Technology
Department of Informatics

FZI Research Center for Information Technology
Information Process Engineering
Prof. Dr. R. Studer

Replica Framework

A Framework for Ontology Sharing and Distributed Ontology Systems

Seminar paper

of

Jan Novacek

Submitted on April 15, 2012

to

Institute of Applied Informatics and
Formal Description Methods (AIFB)
Karlsruhe Institute of Technology

Examiner: Prof. Dr. Rudi Studer
Supervising tutor: Jens Wissmann, M.Sc.

Study-Address: Bussardweg 94
67346 Speyer
novacek@fzi.de

Jan Novacek
Bussardweg 94
67346 Speyer

Hereby I assure that I made this seminar paper at hand without third parties' help only with the specified sources and aids. All figures, which were gathered from the sources, were marked as such. This seminar paper, in same or similar form, has not been available to any audit authority yet.

Karlsruhe, April 15, 2012

(Signature)

Jan Novacek

Acknowledgements

I want to thank Jens Wissmann for his continuous thoughtful support and from whom I have learned a lot throughout the writing of this seminar paper.

Special thanks also to my mother and to my brother for helpful advice and honest criticism.

Abstract

This seminar paper presents a novel framework for developing a Distributed Ontology System (DOS) and tools for Collaborative Ontology Development (COD). Both topics have much in common but have been dealt with rather separately up to this point, the framework is an initial step to met requirements of both fields. In addition to the presentation of the concept, an implementation of the Replica Framework is also discussed in this work.

First, the work is motivated, then the reader will be taught fundamentals of the Semantic Web and related technologies. The terms DOS and COD are explained and aspects of both fields relevant to the Replica Framework. The presentation of the framework model and components as well as the presentation of the backend and frontend implementation follows. Finally conclusions are drawn and an outlook of future work is given.

Der vorliegende Text ist auf Basis des Latex-Templates zu [T. Gockel] 49 erstellt.

Contents

Chapter 1

Introduction

1.1 Motivation, objectives and contributions

There is a need for collaborative ontology development tools. In fact, first contributions to the field of collaborative ontology development were made in the outcomes of the *Knowledge Sharing Effort* initiated by the Defense Advanced Research Projects Agency (DARPA) back in 1990 with the *Ontolingua Server*. Nowadays the development of large complex ontologies such as the Biomedical Grid Terminology (BiomedGT)[1] involve many scientists from all around the globe and requires collaborative tools to assist in the process of creation which is organized in workgroups with members from many different countries.

In the vision of the semantic web [Berners-Lee 01] there will not be a single large ontology but rather many ontologies scattered throughout the web with content specific to their creators domain but interlinked as to the principles of Linked Data. Knowledge will be fragmented in smaller pieces of information rather than residing in one global knowledge base accessible from a single host. Therefore distributed ontology systems are required.

There have been several successful approaches for Collaborative Ontology Development and Distributed Ontology Systems, but they have been dealt with rather separately. The concept of a distributed ontology is in fact quite similar to the situation in Collaborative Ontology Development where many different authors develop specific parts of an ontology: A large ontology is divided into smaller parts either physically or logically which together form the ontology.

This seminar paper presents a novel framework designed to met requirements of both fields as an initial step to improve the process of building semantic web applications with a need for large scale distributed ontologies and collaboration tools.

[1] http://biomedgt.nci.nih.gov/

Chapter 2

Semantic Web Fundamentals

This chapter gives an introduction to the Semantic Web and to related technologies that are relevant to the subject of this seminar paper.

2.1 The Semantic Web

One problem with information on the internet today is that information representations are designed to be used by humans. Consider you want to find all Wikipedia articles about musical artists who were born in Germany and lived before 1900. While Wikipedia covers this information implicitly in its articles you can not easily express a query for this and would have to read all articles to find those you are looking for.

Within the semantic web effort languages for expressing such knowledge in a machine processable form are developed. By adding semantic data to articles it would be possible for a machine to recognize and deal with the content. The approach of the Semantic Web is to augment the existing web with machine processable meta information [Berners-Lee 98].

For a comprehensive introduction to the sematic web refer to [Antoniou 08].

2.2 Ontologies

The term *ontology* is originating from the philosophical branch of metaphysics the study of the nature of being, existence and reality. A major subject of ontology is the analysis of categories of being and their relations. Categorization of being means to determine categories of entities. There are many views amongst philosophers on what may be the most fundamental ones - Classes, Physical objects, Relations or Space and Time to name a few. The definition of categories means also to identify differences or similarities between the entities the categories contain.

In computer science an ontology is a "formal, explicit specification of a shared conceptualization" [Gruber 93]. It is used to represent knowledge in a machine readable form where knowledge is understood as a set of concepts and the relations between these concepts.

While the primary purpose of an ontology is to express machine understandable knowledge the degree of formality of an ontology can vary. For example the WordNet[1] thesaurus has word-sense pairs where the lexical definitions are expressed in natural language. The Gene Ontology[2] a community driven bioinformatics initiative holds standardized gene representations with definitions in natural language but also gene product associations and attributes. And last but not least CYC[3] provides formalized theories for many aspects of human knowledge facts in its own formal language CycL.

Ontologies are used in the fields Semantic Web, artificial intelligence, systems engineering, software engineering, biomedical informatics, library science, enterprise bookmarking and for building an enterprise architecture framework. Generally in every area where knowledge representation is required.

For an overview of ontologies refer to [Horrocks 08].

2.2.1 Ontologies vs. Databases

There is an obvious analogy to databases but there are important differences between ontologies and databases. In contrast to databases, ontologies have an *open world assumption*: The truth-value of a statement unknown to an observer is assumed to be true. Information is assumed to be incomplete by default. Consider a query to a database for a certain telephone number. If there is no entry for the name you are looking for in the database the query will evaluate to `false` which would denote there is no such number. In comparison to that when asking an ontology you could not tell whether there is a number or not because it is just unknown.

Moreover unlike databases ontologies do not assume that instances have a unique name. An instance can be refered to with more than one name. When talking about a *cape* we can also refer to it by saying *cloak* but we mean the same thing. In a database these would be two different entities whereas in an ontology it can be expressed that both names refer to the same concept.

Finally whilst database schemas behave as constraints on the structure of data defining legal database states, ontology axioms behave like logical implications and entail implicit information. For more information on the topic refer to [Horrocks 08].

RDF

The *eXtensible Markup Language* (XML) is the key framework for interchange of data and meta information between applications. But XML does not define any semantics just syntax.

The *Resource Description Framework* (RDF) is a *data-model* developed in 2004. It has clearly defined semantics which can be used to describe concepts and for modeling of information and provides therefore the basic means to express an ontology. The concept of a *resource* is fundamental to RDF. A resource can be anything and is identified by a uniform resource identifier (URI). Resource is the base class of all other RDF classes which include containers

[1] WordNet developed at Princeton University, http://wordnet.princeton.edu/
[2] Gene Ontology http://www.geneontology.org/
[3] CYC Knowledge Base http://cyc.com/cyc/technology/whatiscyc_dir/whatsincyc

of alternatives, unordered containers, and ordered containers as well as a class representing a list.

The key idea of RDF is to express knowledge in *statements*. A statement is a triple consisting of a subject a predicate and an object. As subject and object are resources information can be represented as resource graphs in RDF where edges correspond to predicates.

Overview of Ontology Languages

This section gives an overview of the ontology languages which led to the development of OWL.

The first popular language to express an ontology was F-Logic[4] developed by Michael Kifer at New York State University and Georg Lausen at the University of Mannheim. F-Logic was originally developed for deductive databases and combines the advantages of conceptual modeling with object-oriented, frame-based languages but also well-defined semantics of a logic-based language.

RDF-Schema (RDF-S) developed by the W3C can be used to express an ontology. The first version was published in 1998[5]. It was intended to structure RDF resources.

DAML-Ont (Darpa Agent Markup Language - Ontology) started in 1999 is an ontology language based on XML and RDF. It was developed as an extension to XML and RDF which were already widespread standards. While the predominant markup language for web pages HTML could primarily be used to describe the look of web pages, by then it did not define any semantics. DAML-Ont defines semantic annotations in form of markups while depending on existing web technologies. That is why it was a major step for the Semantic Web.

OIL (Ontology Inference Layer or Ontology Interchange Language) published in 2001 based on concepts developed in Description Logic and frame-based systems, a proposal for a standard in specifying and exchanging ontologies which is also grounded on Web languages. OIL is based on XML and RDFS which means that any RDFS ontology is also a valid OIL ontology.

DAML+OIL developed by a joint committee of United States and European Union members and published in March 2001[6] combines features of both and is the predecessor to OWL.

2.2.2 OWL

The most popular and recent of ontology languages is the *Web Ontology Language* (OWL) developed by the W3C and published in 2004. OWL is grounded on XML and RDF and is designed not only to formulate but also to exchange and reason with knowledge about a domain of interest.

OWL poses three levels of expressiveness OWL Lite, OWL DL and OWL Full. OWL Full is the most expressive version of OWL. It is however possible to state undecidable problems in OWL Full which is a problem for fully automatic reasoning. OWL DL is a fragment of OWL that is based on decidable Description Logics (DLs) that guarantee to sound and

[4]Frame Logic, refer to [Kifer 95]
[5]RDF-S first version http://www.w3.org/TR/1998/WD-rdf-schema-19980409/
[6]The DARPA Agent Markup Language Homepage, http://www.daml.org/

complete inferences and is thus especially useful in the context of automatic reasoning. As the complexity of reasoning in OWL DL might be high, OWL Lite is a further restricted fragment that makes further guarantees to the tractability of reasoning problems and is meant to be implemented easily. In this work we focus on the ontology language OWL DL.

Description logics (DLs) are a family of knowledge representation languages which can be used to represent the terminological knowledge of an application domain in a structured and formally well-understood way. DLs have been applied in many domains, such as medical informatics, software engineering, configuration of technical systems, natural language processing, databases, and web-based information systems[7]. The name description logic refers, on the one hand, to *concept descriptions* used to describe a domain, on the other hand, to the *logic-based semantics* of DLs. They originated from research in semantic networks [Quillian 67] and frame systems [Minsky 81], and efforts to give these a formal basis [Woods 75, Brachman 77, Hayes 77, Hayes 79].

DLs formalize the vocabulary of a domain in terms of *individuals*, *concepts*, and *roles*. The "Handbook of Description Logics" [Baader 03] discusses several DLs, most of which extend the basic description logic $\mathcal{ALC}$ (Attribute Language with Complements) that was introduced by [Schmidt-Schauß91]. DLs are named according to the constructs they provide to describe entities, and so, to their expressive power [Schmidt-Schauß91][8].

Definition 2.2.1 ($\mathcal{ALC}$ syntax) *Let N_I, N_C and N_R be disjunct sets of* individual names, concept names *and* role names. *Then the set of* $\mathcal{ALC}$-concept descriptions *is defined inductively as follows:*

1. *Each concept name $A \in N_C$ is an $\mathcal{ALC}$-concept description.*
2. *The most general concept $\top$ and the unsatisfiable concept $\bot$ are $\mathcal{ALC}$-concept descriptions.*
3. *If C, D are $\mathcal{ALC}$-concept descriptions, and $R \in N_R$, then the concept conjuntion $C \sqcap D$, the concept disjunction $C \sqcup D$, the concpet negation $\neg C$ and the concept role quantifications $\forall R.C$ and $\exists R.C$ are also $\mathcal{ALC}$-concept descriptions.*

This syntax definition is often also given in a Bachus-Naur-style form

$$C, D \rightarrow A \mid \top \mid \bot \mid C \sqcap D \mid C \sqcup D \mid \neg C \mid \forall R.C \mid \exists R.C \tag{2.1}$$

Concepts can be used in axioms to capture their relationships. Terminological axioms (TBox) are can have the form of subsumptions $C \sqsubseteq D$ or equivalences $C \equiv D$. Assertional axioms (ABox) can be type assertions $C(a)$ or property assertions $R(a, b)$. An ontology then is a set of axioms.

By $\Sigma(\mathcal{O})$ we denote the signature of an ontology (O), i.e. the set of class, property and individual names in the ontology. $\Sigma(\mathcal{O})$ consists of axioms which are definitions of classes and the relationships between them. There are various languages for writing axioms with syntactical and semantical differencesa an example of an $\mathcal{ALC}$ axiom with $Pizza, CheeseTopping \in N_C$, $hasTopping \in N_R$ is:

$$(Pizza \sqcap \forall hasTopping.CheeseTopping) \sqsubseteq CheeseyPizza \tag{2.2}$$

[7] For details on these and other applications, see [Baader 03, Part 3].

[8] Evgeny Zolin provides a web-interface that allows to explore different possible DL constructors and their computational properties: http://www.cs.man.ac.uk/~ezolin/dl/

Which is the implication that the conjunction of the *Pizza* and *CheeseTopping* concept is a *CheeseyPizza*. For an example for writing an axiom in OWL refer to listing A.3. The example expresses that the *CheeseTopping* class is a sub class of *PizzaTopping* along with its portuguese label. If the class *PizzaTopping* has not yet been defined anywhere else in the ontology it is implicitly asserted to exist. The current version of OWL is OWL 2 which is an extension to the first version and which adds new functionality. OWL 2 is also based on RDF/XML and the relationships between the Direct and RDF-based semantics have not changed. For more information on that topic refer to the OWL 2 Web Ontology Language Primer [Hitzler 09].

2.2.3 Logical Reasoning with OWL

As already stated in section 2.2.2 OWL DL is a fragment of first-order logic. Thus decision procedures for sentential logics and proof procedures for formulas of first-order logic can be used to infer additional information from an OWL ontology. Interesting inferences are for example whether the ontology is consistent, whether a class subsumes another class whether two classes are equal or disjoint or if an individual belongs to a certain class. The answers to each of these questions have important implications. For example determining class subsumptions reveals the structure of the class hierarchy of the ontology and knowing whether a certain individual belongs to a certain class allows to identify all individuals which belong to that class. Other examples for important questions are whether two classes are equal or if they are disjoint.

The existence of many description logic reasoning tools was one of the key motivations for basing OWL on a DL [Horrocks 08].

Reasoners

Modern reasoners like HermiT[9] or Pellet[10] implement sophisticated approaches and apply heuristics for performance optimization. Most ontology development systems such as Protégé[11] offer users support of a reasoner to identify inconsistent (often called unsatisfiable) class definitions. Such classes cannot have any members and are therefore useless and indicate a fundamental error in the ontology design.

Another function of reasoning systems is the explanation of inferences. Explanation typically involves computing a (hopefully small) subset of the ontology that still entails the inference in question, and if necessary presenting the user with a chain of reasoning steps [Kalyanpur 05]. This can help finding mistakes during the development of the ontology.

[9] HermiT OWL Reasoner http://hermit-reasoner.com/
[10] Pellet: OWL2 Reasoner for Java http://clarkparsia.com/pellet/
[11] Protégé http://protege.stanford.edu/

2.2.4 Modularization

This section offers an introduction to ontology modularization and different ways to achieve it.

Modularization or sometimes *segmentation of ontologies* has a major role in Distributed Ontology Systems (DOS). It is a widely used technique in various areas of computer science such as algorithms, software engineering and programming languages. It is the application of the divide and conquer principle. An intuitive understanding of the concept of module is some subset of a whole which makes sense (i.e., is not an arbitrary subset randomly built) and can somehow exist separated from the whole, although not necessarily supporting the same functionality as the whole [Menken 05]. Modularization is a process and the outcome of this process is generally a *module*. A module encapsulates functionality and defines logical boundaries to other modules.

In the context of ontology engineering an *ontology module* can be understood as a reusable component of a larger or more complex ontology, which is self-contained but bears a definite relationship to other ontology modules [Doran 08]. Another definition is that modularization should be considered as a way to structure ontologies, meaning that the construction of a large ontology should be based on the combination of self-contained, independent and reusable knowledge components [d'Aquin 07]. The goals of ontology modularization include but are not limited to reuse, scalability for information retrieval and reasoning as well as for evolution and maintenance, complexity management, understandability and personalization.

Modularization Approaches

How an ontology module is created depends on the development method. There are at least three ways to create modules.

1. Starting point is a large ontology that is partitioned in smaller modules This approach is most important in Distributed Ontology Systems, where large ontologies are scattered across several nodes to reduce the amount of memory required when storing the ontology as a whole on a single node or to provide redundancy.
2. Starting point is a set of modules, target is a wider ontology consisting of a partition of modules. This is also important in Distributed Ontology Systems especially in concern of distributed query answering.
3. Modularization at the design phase in the lifecycle of an ontology. This approach expects the ontology developers to know about logical ontology modules which are reasonable.

Forms of Ontology modularization

There are several strategies which can be applied when distributing knowledge across modules. For more detailed descriptions of the modularization strategies have a look at [Parent 09].

Disjoint or overlapping modules Disjoint modules have the advantage of easier consistency management. A set of consistent modules is always consistent. A modularization system may have complete control over the knowledge distribution process across modules or users may somehow have the ability to adjust the distribution.

Semantics-driven Strategies Semantics-driven strategies target the semantic part of an application domain from the users/developers point of view. Users have to define to which module knowledge should belong manually. Modularization systems may be used to assist in this process e.g. make proposals.
A simple and traditional way to specify a module is by writing a query for a subset of an ontology. The evaluation of the query then produces the module.

Structure-driven Strategies Structure-driven strategies ignore semantics and look at ontologies as data structures namely graphs of interconnected nodes. Graph decomposition algorithms are used to compute subsets of the graph based on structural properties.

Machine learning Strategies In contrast to human-driven development machine-learning methods analyze the use of ontologies and compute modules based on criteria such as queries.

Monitoring modularization and making it evolve Whatever the modularization strategy may be it is important to evaluate and monitor the reliability and efficiency of the module. Aspects of modules may be adjusted or removed when they prove to be unsatisfactory or not efficient.

Modularization Tools

While there exist many different approaches of ontology modularization, by the time of writing few mature public tools exist which are stable enough to be used in the wild when it comes to OWL ontologies. Some of these tools are presented in the following list:

OWLAPI The OWLAPI[12] facilitates modularization tools.

ModTool The ModTool utilizes the JENA[13] API and is a standalone ontology modularization tool that comes along with GUI.

Galen Segmenter A Jakarta Tomcat web application segmentation service, as described in [Seidenberg 06].

2.3 Collaborative Ontology Development

In some projects, the development of an ontology is very hard or impossible to be achieved by a single person. The ontology may become very large and is difficult to maintain or the complexitiy is too high. Collaborative Ontology Development (COD) is the subject of multi-user ontology development which can be done team based or with anonymous user contributions for example. Other situations may require hierarchical organizations or editors and reviewers.

Especially in scientific projects where many domain experts need to work collaboratively on a task, communication is essential. Working on an ontology is thus not restricted to only editing the ontology but also involves a lot of communication amongst editors and other persons that

[12]The OWL API is a Java API and reference implmentation for creating, manipulating and serialising OWL Ontologies http://owlapi.sourceforge.net/

[13]JENA a open source framework for building semantic web applications http://jena.sourceforge.net/

are part of the development process. That is why a COD system usually supplies something like chats, message boards or other tools which assist communication. If and to which extend these tools are supported varies from COD system to COD system.

2.3.1 Frameworks and Toolkits

This section presents frameworks and toolkits related to COD, points out a significant difference between COD and Distributed Ontology System (DOS) framework characteristics and offers a short presentation of two most popular projects for each.

Most frameworks have much in common in regard to the support of collaborative ontology development aspects. Differences are rather on the technical and architectural side which in practice often has a great impact on usability and performance.

The most preeminent technical difference right now is the nature of the client interface. A developer can use many different technologies to implement an application. We will distinguish between desktop and web applications.

Frameworks intended for building desktop applications

A desktop application is a program installed locally on a computer system intended to be used with a GUI. Such applications can leverage all features that the underlying platform provides including additional hardware.

Collaborative Protégé Collaborative Protégé [14] is an extension to Protégé[15] a popular and widely used ontology editor and knowledge-base framework. It supports synchronous and asynchronous collaboration via two working modes: *multi-user mode* and *standalone mode*. Annotation of ontology components Annotation of changes Discussion threads Proposals and voting Searching and filtering Live discussion (chat).

NeOn Toolkit It can be compared to Protégé[16] and is an outcome of the NeOn project which started in March 2006. NeOn aims to achieve and facilitate the move from feasibility in principle to a concrete solution focusing on cost efficiency and effectiveness of knowledge acquisition for, and design, development and maintenance of largescale, heterogeneous semantic-based applications [Consortium 06].

Frameworks intended for building web interfaces

While there exist several frameworks maily intended for desktop application development, there are also many projects which provide means for building web interfaces. Web interfaces have several advantages above desktop applictions but also some drawbacks. The most important advantage of web interfaces over desktop applications is that they do not require clients to install any software except a browser thus easing usage. On the other hand they

[14] http://protegewiki.stanford.edu/wiki/Collaborative_Protege

[15] http://protege.stanford.edu/

[16] Protégé a free, open source ontology development platform but while the editor of this application has similar features to the ones of Protégé, the background and purpose of the NeOn Toolkit is different. http://protege.stanford.edu/

might limit the functionality because all user interface features depend on and are restricted to the capabilities of the client browser. By the time of writing web applications have become very popular.

OntoVerse *Ontoverse*[17] - *Cooperative knowledge management in the life sciences network* is a research project funded by the German Federal Ministry of Education and Research. It is targeted at the scientific community and provides an ontology editor with additional Wiki functionalities to plan and discuss domain ontologies, particularly for the Life Sciences.

Semantic MediaWiki The *Semantic MediaWiki*[18] is an extension to MediaWiki which is the base code for Wikipedia. It allows users to add meta-information to articles in order to enable processing by machines later.

2.3.2 Aspects of Collaborative Ontology Development

This section summarizes aspects of collaborative ontology development and functionalities of development frameworks as identified in [Tudorache 07], [Tudorache 10], and [Malzahn 07].

Synchronous and asynchronous ontology access In some situations it may be an advantage to edit an ontology simultaneously. On the other hand this might not be pratical in a situation where editors from different countries live in different timezones.

Discussions, chats and annotations of ontology components Users may need to discuss about certain parts of the ontology to find consensus. Thats is why support for discussions and/or chats is needed. It is also important, that the messages are linked direct to the content they refer to, so that a user can see what other users have written about a certain concern.

Access control and provenance of information In a large work group there might be probably be members with an expertise in a certain domain. These users may be allowed to edit corresponding parts of the ontology but may not be allowed to edit other parts. Moreover provenance of information is not only desirable in security concerns but also in regard to collaboration.

Workflow support Collaborative tools should support different tasks, the process for proposing a change reaching to the concensus and the roles that users might play in this process.

Private/shared workspace support There should be a differentiation between a local copy and the publicy available copy. Users should have the opportunity to edit a local copy without making all changes immediately public.

Searching Users should be able to search either in the set of shared ontologies or in the changes that have been made.

Group awareness In large projects developers will probably be organized in more than one work group. The framework should take account of work group support.

Version control Keeping track of document versions offers the possibility to monitor changes and also to roll them back if needed.

[17] http://www.ontoverse.org/
[18] http://semantic-mediawiki.org/

2.4 Distributed Ontology System

At first it is important to mention, that the term *Distributed Ontology System* (DOS) is not used consistently. Some authors use the term in reference to collaborative ontology development systems. Other sources use it in reference to huge ontologies fragmented in smaller pieces which are distributed over several computation nodes. When used in this seminar paper the term refers to the latter.

Not only that Ontologies have become very large, lately the amount of ontologies have also increased dramatically which poses several challenges. For reasoning on very large ontologies the performance of current reasoners is not sufficient [Chen 09]. An initial approach to cope with this problem is to distribute the knowledge and query processing across a set of nodes.

Moreover querying multiple ontologies which were independently developed is problematic. For example a knowledge base used in an IT company may be composed of several ontologies each developed independently for the domain of a certain company department. To pose a query against the whole knowledge base would denote querying each of these ontologies and computing the result of the composition of the individual query results afterwards. When unification of the development methodology is impractical or impossible it is beneficial to use a uniform query mechanism. An easier approach to querying the knowledge base would be to unite the ontologies logically to a single large ontology and pose the query against this large ontology. Thus the process of querying the ontologies becomes transparent to the user.

The key idea of a distributed ontology is that a large ontology is composed of smaller pieces called *fragments*. These fragments are distributed across an arbitrary number of independent nodes. A more formal definiton of this concept is presented in the Framework Architecture chapter.

2.4.1 Modularization

The modularization of an ontology is a very important principle when it comes to Distributed Ontology Systems. It has a major impact on the performance of a DOS. Ontology fragments may contain complete or incomplete A-Boxes, T-Boxes or R-Boxes. A fragment is called a *module* if a query performed against it returns the same result as a query performed against the whole ontology. Therefore it is useful to differentiate between modular and non-modular fragments which is basically the question whether the fragments are disjoint or overlap.

Querying non-modular ontology fragments of an ontology poses several problems for example that there might be intrinsic knowledge which cannot be inferred of a single fragment alone.

2.4.2 Distributed Query Answering

A *query* is a request for either information retrieval or information modification with information systems. Examples for query languages in the context of Semantic Web are the RDF query languages RDQL[19], Sesame RDF Query Language[20] and SPARQL [21]. Posing queries

[19]RDF query Language http://www.w3.org/TandS/QL/QL98/pp/rdfquery.html
[20]SeRQL http://www.openrdf.org/doc/sesame/users/ch06.html
[21]SPARQL Protocol and RDF Query Language http://www.w3.org/TR/rdf-sparql-query/

against a DOS is not a trivial task and may involve querying multiple hosts combining requests or results. A query processing procedure for conjunctive queries is presented in [Chen 08]. An interesting approach to distributed query answering using the MapReduce programming model is presented by [Alvarez 10].

Chapter 3

Framework architecture

This chapter presents the Replica Framework architecture. Requirements are identified at first followed by examples of usage scenarios and formal definitions of shared ontology and distributed ontology system components which make up the Replica Framework.

Afterwards the important aspect of *Change Management* and responsible change processing components are presented in detail followed by a description of behavioral aspects of Replica Framework components. At last a brief description of the Replica Framework configuration is given.

3.1 Requirements

This section characterizes requirements for building a solid architecture that will satisfy the needs of a development platform to build Collaborative Ontology Development (COD) and Distributed Ontology System (DOS) tools.

It is divided in three sub sections for general requirements and two sub sections for requirements specific to COD and DOS. [Tudorache 10] proposed general requirements and the COD requirements.

3.1.1 General Requirements

General requirements are characteristics that are fundamental to the Replica Framework which are not specific to the COD or DOS aspects.

Scalability, reliability, robustness Users will not use tools they cannot rely on, there are also other frameworks. Collaborative applications should also scale with both the size and complexity of the ontology and the amount of collaborators.

Support for various levels of expressiveness Framework tools should perform with simple and complex ontologies equally well. The complexity should ideally not have an impact on performance.

3.1.2 Collaborative Ontology Development requirements

A key requirement of the collaborative ontology development side of the framework is the possibility of *team based ontology development*. This allows the deduction of several other requirements.

Access Control When multiple clients can edit an ontology it is a mandatory requirement of the framework to possess some kind of access control. The granularity of the access control mechanism should be fine enough to enable change filtering based on the user ID and the kind of change. Thus the framework should also supply a component that allows the definition of change filters in a flexible and efficient way.

Provenance of information When it comes to ontology changes, provenance of information is very important. The framework should provide the means to track changes. It is important to know the user ID a change originates from but also the date a change was made on and the reason for the change. Change tracking allows the recording of change histories. The framework should leave the construction and storage of a change history up to the user.

Communication tooling At last the framework should also be flexible enough to support the development of communication tools that suit any application area.

3.1.3 Distributed Ontology System requirements

In addition to those mentioned in section 3.1 there are also requirements for the DOS side of the framework. The following list of requirements is a selection of distributed system requirements proposed in [Hobo 04].

Heterogeneity A key requirement for a DOS is heterogeneity. The system has to be able to run on a wide range of hardware and software platforms to simplify system installation.

Openness Openness means the level of open standards used when the distributed system is built. Simple interfaces and use of widespread standards simplify system integration and increase flexibility.

Transparency Transparency means that the DOS implements a thick abstraction layer hiding a lot of operations and complexity from the user. The system is seen as a whole instead of a lot of different components.

Fault-Tolerance The DOS should be able to handle errors from nodes. If a node becomes unavailable or unstable, the system has to cope with that and remain stable.

3.2 Framework Components

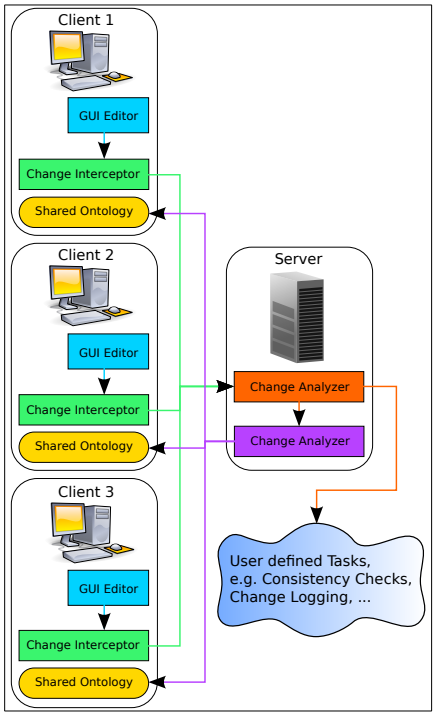
This section describes the main components of the framework. First, a graphical overview presents all components followed by detailed descriptions of the component functions. Implementation details are described in the backend implementation chapter 4.

3.2.1 Usage scenarios

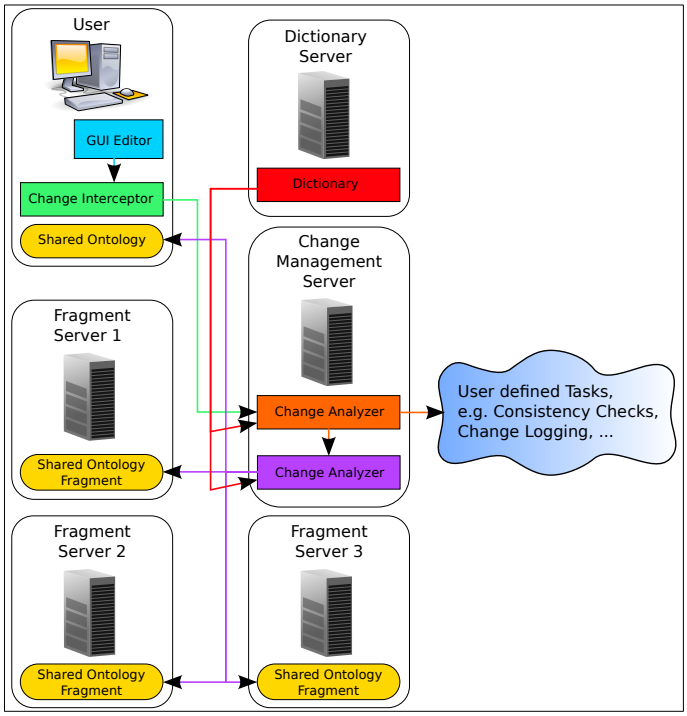
The Replica Framework offers several units meant for implementing either Collaborative Ontology Development (COD) scenarios or Distributed Ontology Systems (DOS). As the framework is configurable, extendable and flexible it can be adapted exactly to the developers needs. The use cases are not limited to simple multi-user scenarios where multiple editors edit an ontology synchronously as shown in figure 3.1a. Instead of that it is possible to create complex mixures of such scenarios and DOS where multiple users with different roles and access rights can edit certain fragments of an ontology which is limited by pre-defined policies and controlled by multiple servers responsible for analyzing changes, protocolling changes and performing consitency checks and a dictionary server holding meta information of all fragments as shown in figure 3.1b.

Figure 3.1: Usage scenarios

(a) Usage scenario 1



(b) Usage scenario 2



3.2.2 Shared Ontology

The key concept of the framework is the concept of a *shared ontology*. A shared ontology is an ontology which can be accessed and modified by multiple clients. Working on a shared ontology is done synchronously which means that changes are visible to all clients immediately after they have been made. The defintion of a *shared ontology* is based on the definition in [Díaz 06].

Definition 3.2.1 (Shared Ontology) *If an ontology is developed through by publishing ontological contribution in contrast to developing it through externalization we call it a* Shared Ontology*.*

A shared ontology may be fragmented in smaller pieces which together form the whole ontology. So a fragment is a subset of statements of the ontology and is therefore an ontology itself. Fragments can either be modular or non-modular. Using modular fragments exposes several advantages which has been described before in section 2.2.4.

Definition 3.2.2 (Shared Ontology Fragment) *For a Shared Ontology o a set of axioms φ is called a* Shared Ontology Fragment *of o when $\varphi \subseteq o$.*

3.2.3 Dictionary

The global dictionary contains meta information about fragments. It covers fragment identifiers as well as fragment ontology signatures and the *identifier function* to look up a fragment by signature and vice versa and is part of a distributed ontology system as proposed in [Chen 09].

Definition 3.2.3 (Identifier Function) *For a set $\mathcal{I}$ of ontology identifiers and a set $\mathcal{F}$ of Shared Ontology Fragments a bijective function $\phi : \mathcal{I} \to \mathcal{F}$ that unambiguously assigns an identifier to every fragment is called an* identifier function*.*

Definition 3.2.4 (FragmentID) *For a set $\mathcal{I}$ of ontology identifiers, a set $\mathcal{F}$ of Shared Ontology Fragments and a fragment $\varphi \in \mathcal{F}$, an identifier ι is called a* FragmentID *of φ if an identifier function ϕ exists such that $\exists \iota \in \mathcal{I} : \phi(\iota) = \varphi$.*

Each Shared Ontology Fragment has a unique ID.

Definition 3.2.5 (Dictionary) *For a set of FragmentIDs $\mathcal{I}_\varphi$, a set of Shared Ontology Fragments $\mathcal{F}$ and signature $\Sigma = \{\sigma(\varphi) \mid \varphi \in \mathcal{F}\}$, and ϕ an identifier function from $\mathcal{I}_\varphi$ to $\mathcal{F}$, a triplet $\mathcal{D} = \{\mathcal{I}_\varphi, \Sigma, \phi\}$ is called* Dictionary*.*

This offers the possibility to look up which fragment holds required information when performing a query against a distributed ontology which is also the primary purpose of the dictionary. The dictionary can be further extended to contain other information about fragments if needed such as the fragment size or structural properties.

3.2.4 Distributed Ontology System

We adapt the concept of a *Distributed Ontology System* (DOS) proposed by [Chen 09]. A DOS consists of a large shared ontology a set of ontology fragments the ontology is composed of and a dictionary to look up fragments.

Definition 3.2.6 (Distributed Ontology System) *For a large shared ontology o, a set of ontology fragments $\mathcal{F}$ which o is composed of and a dictionary $\mathcal{D}$, a triplet $\{o, \mathcal{F}, \mathcal{D}\}$ is called* Distributed Ontology System.

3.2.5 Change Management

This sub section presents components related to change management in the Replica Framework.

When it comes to the distribution of a shared ontology the Replica Framework has to provide the means for managing changes. Each node provides special components which allocate all necessary functionalities for *change management* from intercepting changes to analyzing and propagating them which are also the three stages in order in which change is processed.

Change Model

The framework does not force the user into using a certain change distribution model. By exhausting the full functionality of the change management the Replica Framework offers the flexibility to implement change management according to user's demands.

The forms of ontology modifications are diverse. Users can add or remove axioms to shared ontologies or certain fragments, add annotations or change the ontology ID. Moreover users may want to apply a set of these modifications. Keeping track of modifications is essential in many applications for example in versioning systems.

Definition 3.2.7 (Change) *For the set Δ of ontology modifications, user ID υ, time stamp τ, comment ω, the 4-tuple $\nu = \{\Delta, \upsilon, \tau, \omega\}$ is called* change.

The Replica Framework uses a special container called *change* to carry a set of modifications, a user ID, a timestamp and a comment which is meant to report the cause of the change.

Definition 3.2.8 (Criterion) *A* criterion *is a function $Change \rightarrow Boolean$ that* matches *a change.*

Note that a *Change* can contain a set of modifications.

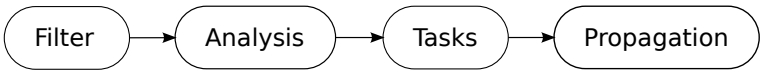
Definition 3.2.9 (Change Filter) *For $n \in \mathbb{N}$, Criteria γ_i, $\Theta = \bigcup_{i=0..n} \gamma_i$ is called* Change Filter.

A *Change Filter* is meant to be used at the first step of the change processing but also in any other component where changes are handled.

Change Processing

Change processing in the Replica Framework follows a scheme which can be modified if needed. Developers can implement custom modifications and other tasks in any component. Figure 3.2 presents the change processing work flow.

Figure 3.2: Change processing work flow



Whenever a change is applied this procedure is triggered. Next change processing is described in algorithm 1 which describes the change processing in detail.

Algorithm 1 ApplyChange(ν)

Require: Shared Ontology o, Change Filter Θ
  if not Filter(ν, Θ) **then**
    $\alpha \leftarrow$ Analyse(ν)
    Trigger(α)
    Propagate(α, ν)
    return applied changes on o
  else
    // ignore change
  end if

The other procedures used in algorithm 1 are:

Algorithm 2 Filter(ν, Θ)

Require: Criteria $\chi_i \in \Theta \quad i = 1, 2, ..., n \quad i, n \in \mathbb{N}$
  for $i = 1$ to n **do**
    if $\chi_i(\nu)$ **then**
      return true
    end if
  end for
  return false

Analyse is a developer-defined analysis procedure that examines a change and returns an analysis as a result. The Replica Framework poses no restrictions on the subject and content of this analysis.

Trigger is also a developer-defined algorithm that depends on the analysis. This algorithm may for example be a consistency check of the ontology or protocolling a change.

Propagate is the function that sends a change to other peers. To which peers the change is sent depends on the developers demands. The change can also be modified at this stage.

Change Interceptor

The *Change Interceptor* is responsible for intercepting and if required filtering changes on the shared ontology. Filters are be based on the node a change originates from and also the kind of change.

This component is local on a node. After a change has been intercepted it is either modified and forwarded to the change analyzer or filtered.

Change Analyzer

The *Change Analyzer* component is used to inspect the change type and possibly trigger user defined tasks connected to certain outcomes of the analysis. A user defined task may for example be a consistency check or some other kind of check for constraints. This offers workflow support.

Changes can also be filtered at this component. The main difference to the change interceptor is that the change analyzer can either be distributed thus local on any node or at certain user specified hosts. This allows centralizing change analysis and easy change tracking for building change histories.

Change Propagator

At last the *Change Propagator* component is required to send the shared ontology change to all or some other nodes. The Change Propagator should know the analysis of the change produced by the Change Analyzer. In addition to that a change propagation strategy should be configurable. The strategy determines which changes will be sent to specified nodes. It is therefore a very important aspect of the framework. The strategy can either be configured to propagate changes to all or just some nodes. The target node(s) of a certain change can also depend on the kind of change which is about to be propagated or other developer defined information.

3.2.6 Communication Node

The framework relies on a client and server communication model. Clients connect to a server and can then retrieve shared ontologies managed by the server they are connected to as well as changes and communication data from other clients. As centralization is not always desired the framework also allows for building decentralized network structures by employing the concept of a *communication node*. A communication node is a peer that provides client as well as server functionality.

The concept of a communication node is similar but not equal to the peer concept of a peer-to-peer network. Table 3.1 offers a comparison between the two ideas which is based on the definition of a peer-to-peer network by [Schollmeier 01]. The major difference between both network models is that in opposite to a self-organizing peer-to-peer network communication node management has to be configured manually.

Table 3.1: Comparison of peer-to-peer and Replica Framework peer concepts

Characteristic	**Peer-to-Peer**	**Replica Framework**
Heterogenity of network connection speed, performance, reliability	yes	yes
Peers provide services and resources and request services of other peers	yes	yes
Services and resources can be shared across all peers	yes	yes
Peers make up an overlay network	yes	yes
Peers are autonomous	yes	yes
The network is self-organizing	yes	no

3.3 Behavioral Aspects

This section outlines an important behavioral aspect of the Replica Framework architecture. *Change Management* is described, the concept and a simple standard management strategy that a framework implementation should provide as a starting point.

3.3.1 Change Management

In some usage scenarios not all users may be allowed to submit changes or enforcing restrictions is necessary. Furthermore changes may need to be modified before applying them to the shared ontology or it may be necessary to trigger an event when a certain type of change is submitted. Performing a consistency check before applying a change is an example for such a situation where change management is required.

To offer a maximum of flexibility a change can be filtered and modified at any stage in change processing: (1) interception (2) analysis (3) tasks and (4) propagation.

Change distribution

A fundamental aspect of change management is the distribution of changes. The easiest change distribution model is plain synchronous unmodified change distribution. All changes made on a node are instantly transmitted without modification. By defining a change propagation strategy for the *change propagator* component presented in section 3.2.5.

3.4 System Configuration

Each Replica Framework component may require a configuration supplied by the user. Whenever possible reasonable defaults are used which can then be overridden by supplied configuration values. To simplify the configuration process a bundle of the individual component configurations can be created and be read at a central place. The Replica Framework takes care of distributing and applying the configurations to corresponding components.

Chapter 4

Implementation of the Backend

To implement the backend side of the framework a combination of object-oriented programming (OOP) and aspect-oriented programming (AOP) was chosen. AOP was applied with the intention to improve the development of the change interception mechanism of the shared ontology described in 4.3 and security related concerns as AOP is suited well for implementing cross-cutting concerns.

To support the development procedure unit tests for all modules were implemented to ensure components remain functional in the course of the development.

The code convention was adapted to the OWLAPI's to ease entry for beginners that are familiar with the OWLAPI.

4.1 Technologies

This section describes the technologies used in this project, reasons why they have been chosen and experiences in the course of the development.

4.1.1 AspectJ

AspectJ[1] is an open-source project of the eclipse foundation and extends Java with AOP. AspectJ is great for implementing tracing logging or other security related concerns.

In the context of the Replica Framework it was meant to implement change interception at first. During the course of the project the AspectJ change interception mechanism was replaced with native Java code for practical reasons. Although AspectJ is not thoroughly used anymore in the project it is still included as it can be used to implement security and logging issues if needed by the developer. The main interface of the shared ontology is prepared to allow easy creation of aspects in regard to shared object access. All shared object methods have a special annotation defined by the marker interface *ProxyMethod*. Additionally methods which modify the shared ontology object carry the *PropagatingMethod* annotation as these methods require change propagation as shown in listing A.1.

[1] AspectJ http://eclipse.org/aspectj/

A major drawback of using AspectJ in this project was the impact on performance and memory usage when implementing change interception and propagation with AspectJ.

4.1.2 Eclipse Communication Framework

The Eclipse Communication Framework[2] (ECF) is a mature framework for building distributed servers, applications and tools. In contrast to other communication frameworks does ECF not implement a single communication protocol but relies on protocol independence instead. It is therefore an abstraction of many well known communication techniques and protocols. This framework implements a sophisticated extendable model which hides underlying protocols with unifying APIs for communication functionalities. These include currently APIs for file transfer, implementing remote services, presence, service discovery, messaging and object replication. ECF is actively developed, has a lot of components and features, along with the fact that the protocol independence and extendability principle offers a lot of flexibility this was the communication framework of choice for this project.

A single but important shortcoming of ECF is the fragmented and incomplete documentation. For many components there is neither an official nor an unofficial documentation available which had a significant impact on the progress of the project development.

4.2 Modules

Modules represent a separation of concerns by packing functionality in logical units. Modular Programming therefore allows to build large and complex applications while improving maintainability.

The framework components are organized in modules which are implemented as OSGi[3] Bundles. Bundles are a collection of Java source files together with libraries and meta information. A special file *MANIFEST.MF* holds meta information such as bundle name and version, vendor name, required execution environment, exported packages and dependencies.

Dependency management is a very important feature of the OSGi framework. This section offers an overview of all Replica Framework modules and describes the *Core* and *Communication* modules in detail. These modules are entry points to the Replica Framework.

4.2.1 Core module

The core module is the one on which all other modules depend on. It contains an implementation of the Shared Ontology described in section 3.2.2 and the class *OWLReplicaManager* which is the entry point to the framework. *OWLReplicaManager* has the same functionality as the *OWLManager* in the OWLAPI.

[2]Eclipse Communication Framework http://www.eclipse.org/ecf/

[3]The OSGi Alliance is a worldwide consortium of technology innovators that advances a proven and mature process to create open specifications that enable the modular assembly of software built with Java technology.

Moreover the core module contains a builder which generates an abstract shared ontology implementation. This builder can be used from the command line and was made for the purpose of reducing maintenance costs in case the OWLAPI is updated.

4.2.2 Communication module

The communication module contains all communication-related components of the framework. The *CommManager* can be used to create connection instances and is required on both the client and server side.

In addition to that does the communication module include the *channel* package. This package contains an implementation of the *SignalChannel* which is a slight addition to the ECF *Datashare API* message concept. *SignalChannel* is meant to be used in application development when communication requires transmission of events or other relatively small typed messages.

4.2.3 Overview

This sub section offers an overview of all Replica Framework modules in table 4.1.

Table 4.1: Overview of all Replica Framework plugins

Bundle Postfix, **Module Name**	**Description**	**Dependencies**
owlapi	The OWLAPI lib Plug-in plugin contains the patched version of the OWLAPI library.	-
core	The core plugin contains the Shared Ontology Implementation and Shared Ontology builder which is required when applying OWLAPI updates.	*owlapi*
app	The application plugin contains interfaces and classes required to build applications like server or client instances.	*core*
comm	The communication module contains all communication related components of the Replica Framework. Especially the *CommManager* is included in this plugin.	*core*
dictionary	The dictionary plugin plugin holds interfaces and classes related to the Dictionary concept of the Replica Framework.	*core*
fragments	The fragments plugin bundle incorporates the Shared Ontology Fragment concept related components of the Replica Framework.	*core*
changes	The Change Management plugin holds special implementations of the *OWLReplicaOntology* interface that actualize all necessary parts of the Change Management concept of the framework.	*core*, *fragments*
neonplugin	The NeOn Toolkit plugin includes the implementation of the NeOn Toolkit plugin.	*core*, *comm*
plicies	This plugin contains components required for *policy management*. It has not yet been implemented at the time of writing.	*core*
query	This plugin holds Query Management related parts. It has not yet been implemented at the time of writing.	*core*
demo	This plugin contains an implementation of a standalone demonstrator application that presents features of the Replica Framework.	*core*, *comm*

4.3 Shared Ontology Implementation

The shared ontology object is a fundamental component of the framework. The *OWLReplicaOntology* interface is a facade combining the OWLAPI *OWLOntology* interface with the ECF *ISharedObject* interface. To provide implementations of this interface was a challenging task. As Java supports multiple inheritance only for interfaces it is impossible to extend from existing implementations of both respective interfaces. Consequently implementations can only extend from either the *OWLOntology* or *ISharedObject* implementations exclusively. The problem is that the effort of implementing all methods of both interfaces and maintaining these over time with updates of both the OWLAPI and ECF is way too high.

The solution is to create an implementation that extends *TransactionSharedObject* a reference implementation of the ECF ISharedObject interface in combination with the proxy pattern. As the implementation has to provide all *OWLOntology* methods as well it contains an ontology as an attribute. The method calls are then delegated to this inner ontology object. To simplify the process of creating such a class in case of an OWLAPI update the framework provides a builder for an abstract class with the structure described before.

4.3.1 Change Interception

Intercepting changes on the ontology instance is essential for the implementation of a shared ontology and change management in the context of distributed ontology system. At first AspectJ was used to leverage the strength of crosscutting-concerns in AOP. The idea was to intercept changes by declaring pointcuts containing all join points that denote ontology changes. Filtering, modification and propagation was implemented in corresponding advices then.

When implementing change management the complexity of the AOP approach became unbearable. Aside from that crosscutting-concerns broke with the principle of separation of concerns fundamental to modular programming. Mixing both approaches was therefore problematic and impractical so falling back to a traditional object-oriented programming solution was the proper alternative. The solution was to intercept changes in the outer ontology proxy methods responsible for applying changes before passing them to the inner ontology object as shown in listing A.1. To avoid cycles when propagating changes, another method is required which bypasses change processing and a distinction between callers as shown in listing A.2.

This approach brought a slight performance improvement as the AspectJ runtime was not required anymore and the amount of method invocations was reduced.

4.4 Communication

To implement a useful collaborative ontology development system it is not enough to provide only a shared ontology. The framework should also provide the means to build other components that support in the process of collaborative ontology development.

Furthermore communication should be reliable and ideally fault-tolerant. This was accomplished by transactional message propagation. ECF provides a special *ISharedObject*

implementation called *TransactionSharedObject* for all-or-nothing, in-order messaging. Using this feature ensures that changes are either applied correctly or dropped. No partial changes can be applied which greatly reduces the risk of raising inconsistencies in case of communication errors.

4.4.1 Connection Management

The main component which is used for connection management is the *communication manager*. It can be used to create multiple connections. A connection provides access to the basic communication facilities. These include the ECF interfaces for managing shared objects and messaging as well as the *signal channel manager*.

Connection Activity

To avoid blocking methods when communicating asynchronously a mechanism is needed which allows sending messages and reacting to a response or a timeout later without waiting for the result. A *connection activity* represents a procedure in the context of a connection.

It contains a list of states and user-defined transitions between these states.

4.4.2 Signals

In some situations messaging may be restricted to sending simple typed messages that trigger certain predefined tasks. For example refreshing the screen or rebuilding an repository index. For this purpose *Signals* have been introduced which are asynchronous messaging based and allow for sending plain typed messages without content or typed messages with content.

4.5 Change Management

Change management implementations are hidden behind the *OWLReplicaOntology* interface. The principle is that changes are intercepted before submitting them to the Ontology object within the OWLReplicaOntology object and processing them. The various components of that processing chain are described in section 3.2.5. At first AspectJ was used to realize ontology change interception. A special marker interface called *PropagatingMethod* indicates which methods in the *OWLReplicaOntology* have a modifying impact and are therefore part of the change management.

4.6 Applications

The Replica Framework comes with three applications that make it possible to use the framework. This section describes the communication hosts briefly that the Replica Framework includes. These are the *server*, *client* and *node* hosts.

4.6.1 Server

The *server* application of the Replica Framework is a central spot for clients to connect to and to hold meta information of all clients and shared ontologies. That is why the server can also be seen as some kind of *shared ontology repository*.

Servers also manage the shared ontologies. For example when clients create a shared ontology they need to retrieve a shared ontology ID from the server before. Whenever a client needs to access a shared ontology it has to connect to a server.

4.6.2 Client

The *client* application of the Replica Framework can retrieve lists of shared ontology IDs from the server. It is used to communicate with the server for accessing shared ontologies.

In addition to that it allows to retrieve and add *shared ontology groups* which are meant to simplify the implementation of shared ontology management. Such a group consists of a set of shared ontologies. A shared ontology can also occur in more than one group.

4.6.3 Node

A *node* is a special host that combines the functionality of servers and clients. Its primary purpose is to ease the development of peer-to-peer applications in the context of Distributed Ontology Systems. The implementation is basically an instance of a server and a client hidden behind the *ReplicaOntologyNode* interface.

4.6.4 System Configuration

To configure the Replica Framework a configuration has to be specified. This configuration consists of sections for each component. Wherever possible reasable defaults are used so that the supplied configuration overwrites configuration values. This principle is meant to ease system configuration.

Support of runtime changes of configuration values is preferred wherever possible. For example the configuration for the creation of connections with the *CommManager* of the communication module described in section 4.2.2 can be changed after the *CommManager* has been started. This configuration sets defaults for new connections but specific parts of it or the whole configuration of a connection can also be supplied to the *CommManager* when a connection is about to be created.

Chapter 5

Implementation of Demonstrations

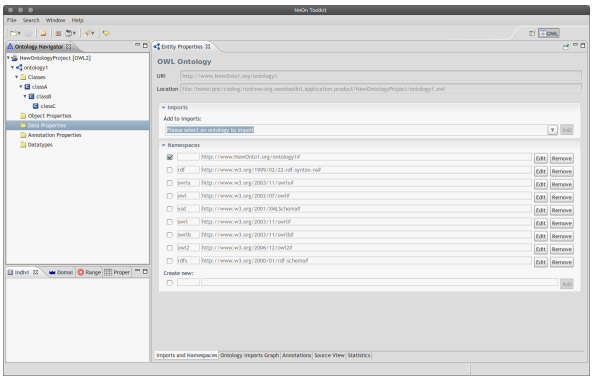
5.1 NeOn Toolkit Plugin Implementation

This section describes the NeOn Toolkit the Replica Framework NeOn Toolkit plugin implementation and presents screenshots of the plugin.

5.1.1 NeOn Toolkit Platform

The NeOn Toolkit is an ontology engeneering environment implemented as a desktop application built on the code-base of OntoStudio[1], which is in turn based on the popular IDE Eclipse. The application is implemented as an Eclipse application with the advantage that the application uses the proven Eclipse application model and tools of the Eclipse platform. Technically an Eclipse application is a special plugin which is started by the eclipse platform. In contrast to usual plugins the lazy-loading principle does not apply to it and menus and toolbars can be customized programmatically.

Figure 5.1: NeOn Toolkit editor GUI



Apart from basic ontology creation the NeOn Toolkit features many other ontology engineering tools that support in the ontology development lifecycle.

[1] OntoStudio is a commerical engineering environment of ontoprise, http://www.ontoprise.de/

5.1.2 Plugin Implementation

The NeOn Toolkit is implemented as an Eclipse[2] application. Like the IDE the application architecture relies on the OSGi module system and service platform[3]. Thus the NeOn Toolkit plugin was implemented as an OSGi bundle containing the necessary meta information required to integrate in the Eclipse application system.

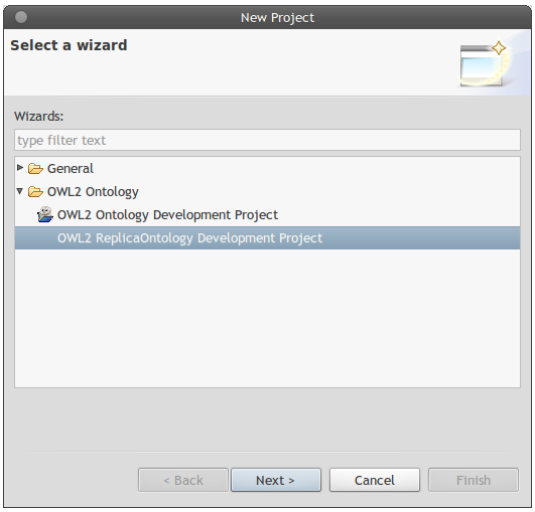
In addition to the main plugin the NeOn Toolkit bundle containing the OWLAPI library had to be modified because the current Replica Framework implementation requires a patched version.

The plugin integrates smoothly into NeOn Toolkit and extends its functionality by adding a Replica Framework project creation wizard and a wizard for creating shared ontologies as shown in figure 5.2. The shared ontology creation wizard also takes care of starting a local Replica Framework server and initializing an empty shared ontology.

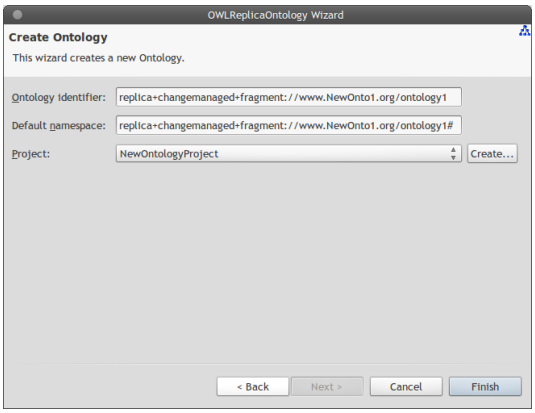
When a shared ontology has been created within a Replica Framework project the NeOn Toolkit editor shown in figure 5.1 can be used to modify the ontology. All changes of the ontology are then synchronized with all other connected editors which enables simultaneous team based development.

Figure 5.2: Shared ontology creation wizard

(a) Shared ontology project creation wizard.



(b) Shared ontology project creation wizard.



[2]Eclipse a popular free, open source IDE http://www.eclipse.org/

[3]The Eclipse development community therefore also provides an implementation of the OSGi specification call Equinox http://www.eclipse.org/equinox/.

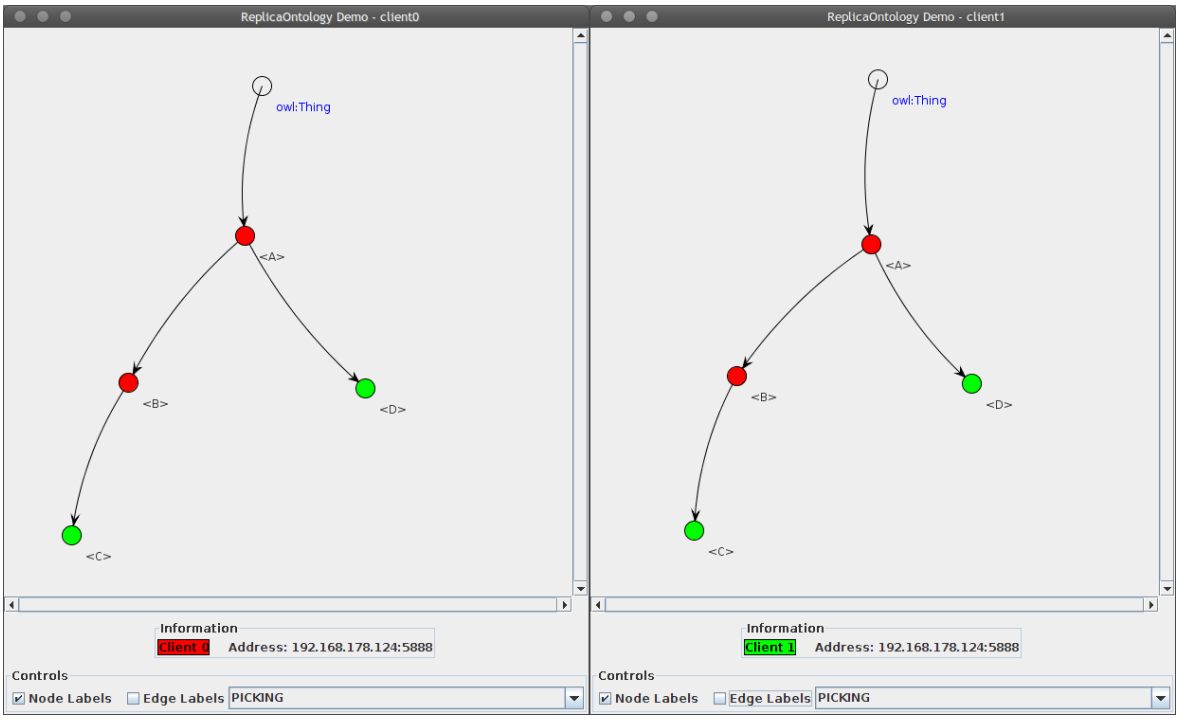
5.2 Replica Framework demonstrator

The Replica Framework demonstrator is a standalone application which uses the JUNG[4] network/graph framework to provide an interactive demonstration of the framework's capabilities.

The demonstrator GUI has a graph view section, an information section and a control section. Nodes in the graph correspond to OWLAPI classes. Each client can add a sub node to any node in the graph. The color of the node indicates the client the node originates from. Every window functions as a client. When the demo starts up two clients connect to a server instance.

Figure 5.3 shows two demonstrator instances, **Client 0** and **Client 1** with a class hierarchy edited by both clients. The node colors indicate the client the node originates from.

Figure 5.3: Replica Framework demonstrator



[4]JUNG — the Java Universal Network/Graph Framework http://jung.sourceforge.net/

Chapter 6

Conclusions

The Replica Framework provides means for implementing Collaborative Ontology Development (COD) tools and Distributed Ontology Systems (DOS). It allows synchronous working on an ontology with multiple users. It is first of its kind that combines approaches from COD and DOS in a unified framework. The ECF-based architecture provides a reliable basis for distributed communication. The framework is flexible to realize a range of COD, DOS and hybrid scenarios. The modular structure of the framework has been created with extensibility in mind and the shared ontology builder allows migration to a new OWLAPI version to reduce maintenance costs when keeping the framework up-to-date which also happened in the course of the development. In addition, every component can be configured individually.

Unit tests for all major functionalities of each module of the code base are provided as a means to foster rapid development of stable applications and framework extensions.

Initial experience shows that the speed of change processing was sufficient in unit tests and the demonstrator but there are various possibilities to improve it. For example, collecting changes and applying them all at once was a lot faster than applying each change on its own.

Many features remain to be implemented in future work. For example *query management* has been omitted as well as *policy management*. While the current implementation of the Replica Framework supports distribution of OWL ontologies, import dependencies between ontologies are not yet supported.

Another aspect that has not been addressed is how different shared ontology fragments are synchronized when they are connected to a logical shared ontology. This could be done for example by computing a logical diff of the ontologies such as CEX and MEX [Konev 08] (implemented in OWLDiff[1]) using the result to gain synchronicity.

Equally important is the question of fault tolerance. While the Replica Framework implementation relies on transactional message communication error correction has not been addressed heavily and for example collecting changes locally and re-applying them later on when the connection is lost has not been implemented yet.

The Replica Framework implementation offers the possibility to configure every component from a central place. This is currently done programmatically and will be extended to file-based configuration in the future.

[1] http://krizik.felk.cvut.cz/km/owldiff/

Concerning the Replica Framework implementation not all of the aspects of COD have been addressed. For example, private/shared workspace support could be integrated together with asynchronous ontology access. The framework model forms a platform for implementation of many of these aspects. A search function and version control can be implemented by using the *change management* components that can also be used to incorporate workflow support and by functions of the communication module. Implementing tools that assist in the communication is easy by using the communication module or the underlying ECF methods directly.

The NeOn plugin and demonstrator presented in chapter 5 is a proof of concept and has shown that the Replica Framework is functional and stable. In the future, we plan further tests of the demonstrator with use case ontologies.

Appendix A

Source code

```
@ProxyMethod
@PropagatingMethod
public List<OWLOntologyChange> applyChange(OWLOntologyChange change) {
    final SharedObjectMsg msg = SharedObjectMsg.createMsg("applyChangeSilent",
        change);
    try {
      // Send change to everyone
      sendSharedObjectMsgTo(null, msg);
    } catch (IOException e) {
      // Error processing
      e.printStackTrace();
    }
    if(calledByOntologyManager()) {
      return ontology.applyChange(change);
    }
    return ontology.getOWLOntologyManager().applyChange(change);
  }
```

Listing A.1: The *applyChanges* method of the OWLReplicaOntology implementation

```
    public void applyChangeSilent(OWLOntologyChange change) {
  change.setOntology(ontology);
  ontology.getOWLOntologyManager().applyChange(change);
}
```

Listing A.2: The *applyChangesSilent* method of the OWLReplicaOntology implementation

```
  <owl:Class rdf:about="#CheeseTopping">
    <rdfs:label xml:lang="pt">CoberturaDeQueijo</rdfs:label>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#PizzaTopping"/>
    </rdfs:subClassOf>
  </owl:Class>
```

Listing A.3: Example of OWL axioms

Appendix B

Glossary

AOP Aspect Oriented Programming.

COD Collaborative Ontology Development.

DAML Darpa Agent Markup Language.

DOS Distributed Ontology System.

ECF Eclipse Communication Framework.

EPL Eclipse Public License.

GUI Graphical User Interface.

IDE Integrated Development Environment.

OOP Object Oriented Programming.

OWL Web Ontology Language.

RDF Resource Description Framework.

RDFS Resource Description Framework Schema.

URI Uniform Resource Identifier.

XML eXtensible Markup Language.

Bibliography

[Alvarez 10] Juan Esteban Maya Alvarez. Query engine for massive distributed ontologies using MapReduce. Dissertation, Information and Media Technologies, July 2010.

[Antoniou 08] Grigoris Antoniou, Frank van Harmelen. A Semantic Web Primer. MIT Press, Cambridge, MA, 2. Auflage, 2008.

[Baader 03] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P.F. Patel-Schneider, Hrsg. The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2003.

[Berners-Lee 01] T. Berners-Lee, J. Hendler, O. Lassila. The semantic web.

[Berners-Lee 98] Tim Berners-Lee. The semantic web road map.

[Boulos 09] Maged N. Boulos. Semantic Wikis: A Comprehensible Introduction with Examples from the Health Sciences.

[Brachman 77] Ronald J. Brachman. What's in a concept: Structural foundations for semantic networks. Int. Journal of Man-Machine Studies, 9(2):127–152, 1977.

[Chen 08] Xueying Chen, Michel Dumontier. Conjunctive query answering in distributed ontology systems for ontologies with large owl aboxes. In Rinke Hoekstra, Peter F. Patel-Schneider, Hrsg., Tagungsband: OWLED, Band 529 of CEUR Workshop Proceedings. CEUR-WS.org, 2008.

[Chen 09] Xueying Chen, Michel Dumontier. A framework for distributed ontology systems. 2009.

[Consortium 06] NeOn Consortium. Neon: Lifecycle support for networked ontologies. technical annex. 2006.

[Doran 08] Paul Doran. Ontology reuse via ontology modularisation.

[Díaz 06] Alicia Díaz, Guillermo Baldo, Gérôme Canals. Co-protégé: Collaborative ontology building with divergences. Seiten 156–160, 2006.

[d'Aquin 07] Mathieu d'Aquin, Anne Schlicht, Heiner Stuckenschmidt, Marta Sabou. Ontology modularization for knowledge selection: Experiments and evaluations. Database and Expert Systems Applications, Seiten 874–883, 2007.

[Fensel 08] D Fensel, F Van Harmelen, B Andersson, P Brennan, H Cunningham, E Della Valle, F Fischer, Z Huang, A Kiryakov, TK Lee. Towards larkc: a platform for web-scale

reasoning.

[Gruber 93] T. Gruber. A translation approach to portable ontology specifications.

[Hadzic 09] Maja Hadzic, Pornpit Wongthongtham, Tharam Dillon, Elizabeth Chang, Maja Hadzic, Pornpit Wongthongtham, Tharam Dillon, Elizabeth Chang. Introduction to Ontology, Band 219 of Studies in Computational Intelligence. Springer Berlin / Heidelberg, 2009.

[Hayes 77] Patrick J. Hayes. In defense of logic. Tagungsband: Proc. of the 5th Int. Joint Conf. on Artificial Intelligence (IJCAI'77), Seiten 559–565, 1977.

[Hayes 79] Patrick J. Hayes. The logic of frames. In D. Metzing, Hrsg., Frame Conceptions and Text Understanding, Seiten 46–61. Walter de Gruyter and Co., 1979.

[Hitzler 09] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, Sebastian Rudolph. OWL 2 Web Ontology Language Primer.

[Hobo 04] Remco Hobo. Distributed system requirements. 2011.03.08.

[Horrocks 08] Ian Horrocks. Ontologies and the semantic web.

[Kalyanpur 05] A. Kalyanpur, B. Parsia, E. Sirin, J. Hendler. Debugging unsatisfiable classes in OWL ontologies. Journal of Web Semantics, 3(4):268–293, 2005.

[Kifer 95] M. Kifer, Lausen. G., J. Wu. Logical foundations of object-oriented and frame-based languages. Journal of the ACM, 42(4):741–843, 1995.

[Konev 08] Boris Konev, Carsten Lutz, Dirk Walther, Frank Wolter. Logical difference and module extraction with cex and mex. In Franz Baader, Carsten Lutz, Boris Motik, Hrsg., Tagungsband: Description Logics, Band 353 of CEUR Workshop Proceedings. CEUR-WS.org, 2008.

[Malzahn 07] Nils Malzahn, Stefan Weinbrenner, Peter Hüsken, Jürgen Ziegler, H. Ulrich Hoppe. Collaborative ontology development - distributed architecture and visualization. Open-Archive-Publikation.

[Menken 05] Maarten Menken, Heiner Stuckenschmidt, Holger Wache (vrije, Giorgos Stoilos, Vassilis Tzouvaras, Technology Hellas, Contact Person Dieter Fensel, Contact Person Alain Leger. D2.1.3.1 report on modularization of ontologies coordinated by stefano spaccapietra (ecole polytechnique fédérale de lausanne) with contributions from:.

[Minsky 81] Marvin Minsky. A framework for representing knowledge. In J. Haugeland, Hrsg., Mind Design. The MIT Press, 1981.

[Parent 09] Christine Parent, Stefano Spaccapietra. An overview of modularity.

[Quillian 67] M. Ross Quillian. Word concepts: A theory and simulation of some basic capabilities. Behavioral Science, 12:410–430, 1967.

[Schmidt-Schauß91] Manfred Schmidt-Schauß, Gert Smolka. Attributive concept descriptions with complements. Artificial Intelligence, 48(1):1–26, 1991.

[Schollmeier 01] Rüdiger Schollmeier. A definition of peer-to-peer networking for the classifi-

cation of peer-to-peer architectures and applications.

[Seidenberg 06] Julian Seidenberg, Alan Rector. Web ontology segmentation: analysis, classification and use. Tagungsband: WWW '06: Proceedings of the 15th international conference on World Wide Web, Seiten 13–22, New York, NY, USA, 2006. ACM.

[Stuckenschmidt 04] Heiner Stuckenschmidt, Michel Klein. Structure-Based Partitioning of Large Concept Hierarchies.

[Tudorache 07] Tania Tudorache, Natasha Fridman Noy. Collaborative protégé. 2009.02.21.

[Tudorache 10] Tania Tudorache, Natalya Noy, Samson Tu, Mark Musen. Supporting Collaborative Ontology Development in Protégé. In Amit Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, Krishnaprasad Thirunarayan, Hrsg., The Semantic Web - ISWC 2008, Band 5318 of Lecture Notes in Computer Science, Kapitel 2, Seiten 17–32. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[Woods 75] William A. Woods. What's in a link: Foundations for semantic networks. In D. G. Bobrow, A. M. Collins, Hrsg., Representation and Understanding: Studies in Cognitive Science, Seiten 35–82. Academic Press, 1975.

[T. Gockel] Form der wissenschaftlichen Ausarbeitung. Springer-Verlag, Heidelberg, 2008.

Subject index