Departamento de Inteligencia Artificial
Facultad de Informática

PhD Thesis

Ontology Metadata Management in Distributed Environments

Author : MSc. Raúl Antonio Palma de León
Advisors : Dr. Oscar Corcho García
Prof. Dr. Asunción Gómez Pérez

2009

To my family

Abstract

In recent years, the widespread use of ontologies in different applications and domains has posed new challenges for their development and maintenance. On the one hand, ontology development has been transformed from a process traditionally performed by one ontology engineer into a process performed collaboratively by a team of ontology engineers, whose members may be distributed and play different roles. For example, editors may propose changes, while authoritative users approve or reject them following a well defined process. This process, however, has only been partially addressed by ontology development methods, methodologies, and tool support.

On the other hand, currently ontologies are normally developed by reusing existing ontologies instead of creating new ones from scratch. However, ontologies are dynamic entities that evolve over time. The management of ontology dynamics has several associated challenges, ranging from the adequate control of ontology changes to the administration of ontology versions. Many of these challenges have been addressed in the past with relevant methodological and technological results. However, in most cases, ontologies are treated as isolated entities instead of as part of a network of complex relationships and dependencies, where ontologies reuse/extend other ontologies, have associated metadata, can be used to integrate heterogeneous knowledge bases, etc. Furthermore, in a distributed environment where ontology editors may be working on local copies of the same ontology, strategies should be in place to ensure that changes in one copy are reflected in all the other copies.

In this thesis we address the **management of ontology changes in distributed environments to support collaborative ontology development**. The first contribution is an **ontology metadata model**, which is a core component of our solution due to several reasons: first, information about changes is just a special kind of ontology metadata as it provides additional data about the ontology, i.e., which changes were applied, when, by whom, etc. Second, the ontology metadata model allows identifying if an ontology has changed by a simple inspection of its metadata instead of analyzing the ontology itself. Third, this model provides a high level overview of how an ontology has changed. For example, additional classes have been defined or the domain has been specialized. And fourth, this model is the base for the other models proposed in this thesis, i.e., they specialize or reuse it. Of course, in a more general context, ontology metadata also plays an important role in the widespread dissemination of ontology-driven technologies as it supports an efficient ontology reuse.

The second contribution consists of **models, methods, and strategies for the management of ontology changes in distributed environments**. By extending our ontology metadata model, we propose a layered model for the representation of ontology changes that provides the basis for the methods and strategies supporting

this task. In particular we address, on the one hand, supporting activities for the manipulation of changes (e.g., capturing and storage) and, on the other hand, the propagation of changes to (i) distributed copies of the same ontology, and (ii) its related metadata.

The third contribution consists of **models and strategies supporting collaborative ontology development**. Our solution is focussed on the process typically followed by organizations to coordinate change proposals and on a distributed control of ontologies and changes. Hence, we propose a model for the formalization of this process that allows describing how changes are applied (e.g., the states of the changes and the policy of curation activities), and we provide strategies to control the collaborative process during the ontology development. Furthermore, based on our strategies for the propagation of changes to distributed copies of the same ontology, we support a distributed control of ontologies and changes.

Finally, we provide the **technological support** to the work here presented, which we have employed to evaluate our contributions.

Resumen

En los últimos años, el creciente uso de ontologías en diferentes aplicaciones y dominios ha generado nuevos retos para su desarrollo y mantenimiento. Por un lado, el desarrollo de ontologías se ha transformado de un proceso ejecutado tradicionalmente por un ingeniero ontológico a un proceso ejecutado colaborativamente por un equipo de ingenieros ontológicos, cuyos miembros pueden estar distribuidos y desempeñar diferentes roles. Por ejemplo, editores pueden proponer cambios, mientras que usuarios autorizados los aprueban o rechazan siguiendo un proceso previamente definido. Sin embargo, este proceso ha sido considerado solamente de forma parcial por los métodos, metodologías y herramientas para el desarrollo de ontologías.

Por otro lado, las ontologías están siendo desarrolladas mediante la reutilización de otras ontologías existentes en vez de crearlas desde cero. Sin embargo, las ontologías son entidades dinámicas que evolucionan a través del tiempo. La gestión de la evolución de las ontologías tiene asociada varios retos, que van desde el control adecuado de los cambios en las ontologías hasta el mantenimiento de sus diferentes versiones. Muchos de estos retos han sido atacados en el pasado con resultados metodológicos y tecnológicos relevantes. Sin embargo, en la mayoría de los casos, las ontologías son tratadas como entidades aisladas en vez de miembros de una red compleja de relaciones y dependencias, en donde las ontologías reutilizan/extienden otras ontologías, tienen metadatos relacionados, pueden ser utilizadas para integrar bases de conocimiento heterogéneas, etc. Además, en un ambiente distribuido en el que editores de ontologías pueden estar trabajando en copias locales de la misma ontología, es necesario tener las estrategias apropiadas en su lugar, para asegurar que cambios en una copia son reflejadas en todas las demás copias.

En esta tesis se aborda la **gestión de cambios en ontologías en ambientes distribuidos para dar soporte al desarrollo colaborativo de ontologías**. La primera contribución es un **modelo de metadatos de ontologías**, el cual es un componente clave en la solución por varias razones: primero, la información acerca de cambios no es más que un tipo especial de metadato ya que provee información adicional acerca de la ontología, i.e., cuáles fueron los cambios aplicados, cuándo, por quién, etc. Segundo, el modelo de metadatos de ontologías permite identificar si una ontología ha cambiado mediante una simple inspección de sus metadatos en vez de un análisis de la ontología como tal. Tercero, éste modelo provee una visión de alto nivel de cómo ha cambiado una ontología. Por ejemplo, se han definido clases adicionales o el dominio se ha especializado. Cuarto, éste modelo es la base para los demás modelos que se proponen en esta tesis, que lo reutilizan o extienden. Por supuesto, en un contexto más general, los metadatos de ontologías juegan también un rol importante en la diseminación extendida de tecnologías basadas en ontologías ya que éstos soportan una eficiente reutilización de ontologías.

La segunda contribución consiste en **modelos, métodos y estrategias para la gestión de cambios en ontologías en ambientes distribuidos**. Extendiendo

el modelo de metadatos, se propone un modelo en capas para la representación de cambios en ontologías que provee la base para los métodos y estrategias que soportan esta actividad. En particular se tratan, por un lado, tareas complementarias para la manipulación de los cambios (e.g., captura y almacenamiento) y, por otro lado, la propagación de cambios a (i) copias distribuidas de la misma ontología, y (ii) a sus metadatos relacionados.

La tercera contribución consiste en **modelos y estrategias que soportan el desarrollo colaborativo de ontologías**. La solución está enfocada en el proceso que típicamente siguen las organizaciones para coordinar las propuestas de cambios y en el control distribuido de ontologías y cambios. Para esto, se propone un modelo para la formalización de este proceso que permite describir cómo son aplicados los cambios (e.g., los estados de los cambios y la política de las actividades de aprobación), y se proveen estrategias para controlar el proceso colaborativo durante el desarrollo de la ontología. Además, utilizando como base las estrategias para la propagación de cambios a copias distribuidas de la misma ontología, se soporta un control distribuido de ontologías y cambios.

Finalmente, se provee el **soporte tecnológico** al trabajo presentado, el cual se ha utilizado para evaluar las contribuciones.

Acknowledgements

This thesis is the result of my work as a research assistant within the Ontology Engineering Group (OEG) at the Universidad Politécnica de Madrid, Spain. I want to mention that most of this work was performed in the context of two European Projects: The Network of Excellence Knowledge Web (FP6-507482), and the NeOn Project (FP6-027595). Furthermore, this work was possible thanks to the opportunity that the scholarship program "BECAS MAE-AECI" provided me.

There are many people I want to thank since they kindly supported me in so many different ways for the successful completion of this thesis.

First of all, I would like to thank in a special way Prof. Dr. Asunción Gómez-Pérez, my supervisor, and guide during almost five years. In this period of my research, she gave me a fair amount of freedom to develop myself, but at the same time she always encouraged and supported me, providing the necessary guidance to achieve the best results. Besides of helping me in many aspects, she was always a great advisor in different facets of my life. Actually, she is the reason I decided to start my research in the semantic web field in the first place.

I would also like to express my thanks to Dr. Oscar Corcho-García, second supervisor of this thesis. I want to thank him specially for his, almost, unconditional availability and his prompt responses. He was always there when I needed to solve any doubt, and he always guided me with great ideas. He, in particular, made it possible that I could finish this thesis in the target time.

I also want to thank Dr. Peter Haase. He was my guide every time I visited the AIFB group at the University of Karlsruhe. He also provoked many of the ideas of this thesis and was always willing to help when I asked him for advice or guidance. I would like to say that I really enjoyed the time I worked with him during my research.

During the elaboration of this thesis, I found out that, in many cases, informal conversations with friends and colleagues were some of the greatest moments to clarify ideas. So, I want to thank all the members of the OEG group for the interesting discussions we had in all these years; I also want to extend my gratitude to all the friends with whom I shared many wonderful moments during my staying in Madrid. I will always remember them, specially Boris, Mauricio and Luis for the interesting debates and the good times we used to have.

I would like to express my deepest thank to my parents who always believed in me. To my dad, my role model, for being the person that encouraged and inspired me to finish this thesis. Without his support I would not have been able to make this dream come true. As he always says: "When you start something you have to finish it, and to finish it in the best possible way". To my lovely mom, for always giving me her unconditional love and time. She gave me peace and confidence, especially in those moments of weakness for being so far from home, that helped me concentrate on the things that matter. And to my wonderful sister, for giving me her love and advices every time I needed.

I owe a special debt to Gosia, whose love and understanding helped me focus

on the elaboration of this thesis.

Last, but certainly not least, I would like to thank my beloved daughter, Misia. Even though she is still too young to know, she gave me the greatest motivation to finish this work.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 Overview

The growing use and application of ontologies in the last years has led to an increasing interest of researchers and practitioners in the development of ontologies either from scratch or by reusing existing ones. In this context, ontology reuse is an important aspect that is progressively better supported by the growing availability of ontologies for different domains. Reusing existing ontologies instead of creating new ones from scratch has many benefits: it lowers the time and cost of developing new ontologies, avoids duplicating efforts, ensures interoperability, etc. In fact ontology reuse is one of the key enablers for the realization of the Semantic Web. As a consequence, complex networks of ontologies are being created in which each ontology may depend on several others and may also be related to other artifacts, such as individuals, mappings, applications and metadata.

Nevertheless, this situation also brings about new issues. Ontologies (like many other system components) are dynamic entities, that is, they evolve/change over time. An ontology, defined as a formal, explicit specification of a shared conceptualization [SBF98], may change whenever any of the elements of this definition changes. For instance, domains are not static or fixed: they may evolve when non-existing elements become part of the domain or when some elements become obsolete, among others. A similar situation occurs with shared conceptualizations, which may change, for example, when the domain experts involved in modeling acquire additional knowledge about the domain. Finally, the formal specification may change because new ontology languages or new versions of the existing ones become available, for example.

Ontology development and maintenance activities are addressed by many different methodologies (e.g., METHONTOLOGY [FLGPPSPS99], On-To-Knowledge [SSSS01] and DILIGENT [PSST04]). However, most of these methodologies only consider the development of ontologies by single users or by a small group of ontology engineers placed in the same location. It is important to note that even though they tackle the methodological aspects, in general they focus

less on the process followed by organizations to coordinate the **collaborative ontology development**. In practice ontologies may be distributed, and a whole team of ontology engineers with different roles may collaborate in their development and maintenance, usually following a well defined process. Examples of such collaborative development processes can be found in international institutions like the United Nations Food and Agriculture Organization (FAO), which is developing and maintaining large ontologies in the fishery domain [MGGPISK07]. Other similar examples are those of the Gene Ontology (GO) project[1], which addresses the need for consistent descriptions of gene products in different databases, and the caGrid project[2], which aims at providing a virtual informatics infrastructure that connects data, research tools, scientists, and organizations. Figure 1.1 illustrates an scenario where a team of three ontology editors, with different roles (editor and validator), are collaboratively developing ontology O1 following a pre-defined process.

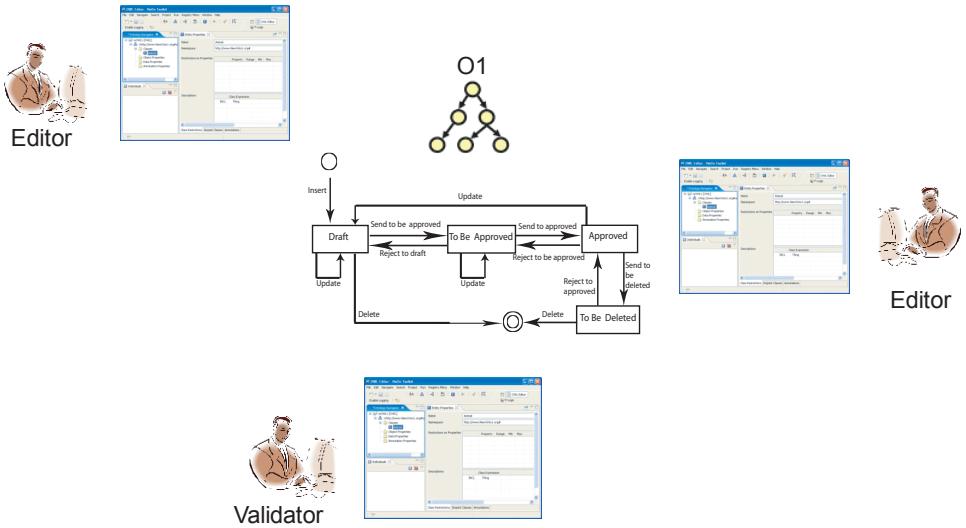


Figure 1.1: A Collaborative Ontology Development Scenario: A team of three ontology editors, with different roles, are collaboratively developing ontology O1 following a pre-defined process

In such a collaborative organizational setting, the existing approaches are not enough to support all ontology development and maintenance needs. For instance, the approaches do not consider the organizational pre-defined process, in which ontology editors can perform different actions depending on their associated role. Besides, they do not support a distributed control of the ontology and of the related changes, which is desirable in this kind of scenario, where ontology editors may be working with local copies of the same ontology. Furthermore, although recently

[1] http://www.geneontology.org/
[2] http://www.cagrid.org/

some proposals and tools have been designed specifically to support collaborative ontology development (e.g., client-server mode in Protégé[3] with the PROMPT and change-management plugins), they generally only tackle parts of the overall problem (see section 2.3.2). Most of the existing ontology tools (for example, Protégé core system[4], SWOOP[5]) support only the single-user scenario, where only one user is involved in the development and later modification of the ontologies. With such tools, a typical scenario of collaborative ontology development would look as follows: an editor changes an ontology using his ontology editor system and then sends his locally changed ontology to other users (for example, by email or by uploading the ontology to an ontology repository). Those users may add more changes to the ontology, or review the current changes. Even in the scenario where all users are editing the same ontology stored in a central server, the coordination of the actions of the editors (for example, when editors want their changes to be reviewed, or what kind of actions they can perform) is not yet fully supported, and the management of the ontology and its related changes is centralized.

Hence, in this type of scenario, we need appropriate models, strategies and infrastructure to support the process that coordinates the collaborative ontology development within an organizational setting. This process can be modeled as a collaborative workflow that, according to [GLP+07] is a special case of epistemic workflow characterized by the ultimate goal of designing networked ontologies and by the specific relations among designers, ontology elements, and collaborative tasks. The need for such workflows has also been acknowledged in the past in other related works (e.g., [TN07]). An example of such workflow is that followed by FAO (described in [MGGPISK07]), which we have taken as a use case in our work to illustrate both the analysis of the requirements and our solution to collaborative ontology development.

Following this workflow, the development process starts with proposals for ontology changes. These proposals are discussed by multiple users (with different roles) in a collaborative way. For instance, if a change is made by an ontology editor, it has to be approved by a validator. After that, the change will be considered definitive and added to the structure. Once changes are definitive, we then have a new stable version of the ontology, which requires the appropriate support to manage different ontology versions. Obviously, one could think of other kinds of workflows in different situations.

As we can see from the previous discussion, the **management of ontology changes in distributed settings** is central/crucial as the whole scenario is driven by changes in ontologies. Consequently, a first step towards the management of ontology changes is the identification of those changes. This identification can be achieved at different granularity levels:

[3] Protégé efforts to address the collaborative process during the ontology development that started last year are contemporary and similar to the work presented in this thesis to address this issue, and which we reported originally as a public technical report in 2007.

[4] `http://protege.stanford.edu/`

[5] `http://code.google.com/p/swoop/`

At a higher level, we need the means to determine *whether an ontology has changed or not*. For instance, ontology users and ontology related entities (e.g., mappings and individuals), should be aware whenever an ontology changes as this change might compromise the application behavior or raise compatibility issues. Moreover, at this granularity level we need *information of the ontologies*, which can be used whenever an ontology changes to have a high level overview of *how* the ontology changed. For example, the domain modeled was specialized, the number of classes increased, additional parties contributed to the ontology development and the content is now available in additional languages. In order to provide the previous information, we need a vocabulary for describing ontologies, i.e., **ontology metadata**. Of course, in a more general context, such a vocabulary provides additional benefits. In particular, it is a core piece to support an efficient reusing of ontologies as described above. For example, in the SEEMP project[6], whose main goal is to help job seekers to find a job in the European marketplace, there was a need to build a reference ontology in the employment domain. During the development of this ontology, engineers analyzed existing ontologies and selected the best knowledge to be reused in order to speed up the process. As an illustration, for the time domain, it was difficult first to find the best candidates for time ontologies and then to compare them. If those time ontologies had had associated and updated metadata, the process would have been easier. For example, knowing information about the ontology such as who was the creator, where the ontology is being used, key classes, provides a better grounding for the final selection.

At a lower level, we should be able to identify *which were the specific changes* that were applied to an ontology. For instance, the class added/deleted, the range of a property, the sub-classes of a class, the value of an individual property and the deleted individual. Hence, using the general ontology information we can know that one additional class was declared in the ontology, but at this level we will also be able to know which was the specific class. That is, at this level we are putting a ”magnifying glass” to the information provided by the higher level. In fact, *information on changes in the ontologies is just a specific kind of ontology metadata*. Providing the previous information is one of the goals of the ontology change management.

Furthermore, dealing with ontology changes involves the execution of many related tasks, most of them identified in the context of the ontology evolution process (e.g., [Sto04]). For example, among these tasks are the capturing and formal representation of ontology changes, the verification of the ontology consistency after the changes are performed, and the propagation of those changes to the ontology related entities. The distributed nature of a network of ontologies where complex relations can exist between ontologies and other artifacts demands the necessity to propagate ontology changes to the distributed ontology-dependent artifacts (e.g., related ontologies, ontology individuals, mappings and metadata). For instance, a

[6] `http://www.seemp.org/`

change in an ontology (e.g., add class) may require one or more updates in its related metadata (e.g., increase the number of classes by one, add an additional key class, add an additional contributor and update the date of the last modification).

Even though we can find some recent works addressing either the whole ontology evolution process or some of the individual tasks, in general, those works are considering parts of the problem, and there are still many lessons to be learned from other related fields. Even though existing approaches model the evolution process in a different way (using different names and number of steps), they identify similar activities. A fundamental activity common to all approaches is the representation of ontology changes. We can find some proposals for the formal representation of ontology changes (e.g., [Sto04], [Kle04] and [NK03]); however, none of them has become widely accepted, and even if the proposed ontologies are dependent on the underlying ontology model, they do not consider the real low-level operations allowed in a specific ontology language (e.g., add an existential restriction in an OWL ontology). Even more, most of the existing approaches consider ontologies as independent entities rather than as part of a network of ontologies, and in the few approaches where the propagation of ontology changes has been addressed (e.g., [Oli00] and [Sto04]), the main focus has been the propagation of such changes to the related ontologies or ontology individuals. In fact, the management of ontology metadata in ontology networks is still an issue that has not been even considered. Having up-to-date ontology metadata is fundamental in our context as it allows identifying efficiently whether an ontology has changed. That is, instead of having to analyze the ontology itself (which could be a time-consuming task), it could be possible to know if the ontology has changed by just analyzing its metadata (as described above). Furthermore, current metadata is of paramount importance for the successful reuse of ontologies (as aforementioned).

Thereupon, in this thesis we make contributions in three main research topics: (i) Ontology metadata vocabulary, (ii) Change management in distributed environments and (iii) Collaborative ontology development.

1.2 Thesis Structure

The remainder of this thesis proceeds as follows:

- Chapter 2 presents the **state of the art** of the topics of interest for this thesis. For each topic we analyze the limitations and open research problems, emphasizing on those to which we are provide solutions in this thesis.

- Chapter 3 provides a presentation of the **objectives and contributions** of the thesis. Based on the limitations found in the state of the art, we describe the overall objective of the thesis and the specific objectives identified. Then, we introduce the contributions of this thesis to the state of the art, followed by the presentation of the **assumptions, hypotheses and restrictions** of this work.

- Chapter 4 presents of our contribution to the topic of **ontology metadata**. First we introduce the result of the analysis conducted to identify the requirements for a community-accepted vocabulary for describing ontologies, and then we present our proposed metadata model.

- Chapter 5 describes our solution for the **management of ontology changes in distributed environments**. It includes the definition of a layered model for describing information about changes in ontologies. Based on this model, we present methods and strategies for the manipulation and propagation of changes in distributed environments. In particular, we consider the propagation of changes to (i) distributed copies of the same ontology, and (ii) to ontology related metadata.

- In Chapter 6, we present our solution to support **collaborative ontology development**. We start by introducing the requirements identified during our analysis of different scenarios, typical within an organization, for developing ontologies collaboratively. Then we present our approach for the formalization of the collaborative process that coordinates the proposal of changes. Finally, we describe the strategies for the management of the collaborative process.

- Chapter 7 is dedicated to the **evaluation** of our work. First we present the **technological support** that we provide for the models, methods and strategies proposed, including the implementation of a distributed ontology registry and a framework supporting collaborative ontology development. Then, we present the evaluations of our models followed by a complete description and analysis of an experiment conducted in a real scenario to evaluate the usability and performance of our solution.

- Chapter 8 includes our **conclusions** and the outlook for the **future work**.

1.3 Publications and Reports

Parts of the thesis have been published before:

Our contribution presented in Chapter 4 has been partially published in:

- Hartmann, J.; Bontas, E.; Palma, R.; Gómez-Pérez, A.: ”DEMO - Design Environment for Metadata Ontologies.” In: Proceedings of the 3rd European Semantic Web Conference, ESWC 2006. Volume 4011. June 2006. Budva, Montenegro. Springer Berlin. 427-441

- Palma, R.; Hartmann, J.; Gómez-Pérez, A.; Sure, Y.; Haase, P.; Suárez-Figueroa, M.; Studer, R. ”Towards an Ontology Metadata Standard”. Poster in the Proceedings of the 3rd European Semantic Web Conference 2006. ESWC’06. June, 2006. Budva Montenegro.

- Hartmann, J.; Palma, R.; Sure, Y.; Haase, P.; Suárez-Figueroa, M. "OMV-Ontology Metadata Vocabulary". In: Proceedings of the International Workshop on Ontology Patterns for the Semantic Web, located at the conference International Semantic Web Conference, ISWC2005. November, 2005. Galway, Ireland

- Hartmann, J.; Palma, R.; Sure, Y.; Suárez-Figueroa, M.; Haase, P.; Gómez-Pérez, A.; Studer, R. "Ontology Metadata Vocabulary and Applications". In: Proceedings of the International Workshop on Web Semantics (SWWS'05), located at the conference On the Move to Meaningful Internet Systems, OTM2005. October, 2005. Agia Napa, Cyprus. Springer. 906-915

- Montiel-Ponsoda, E.; Aguado de Cea, G.; Suárez-Figueroa, M.; Palma, R.; Gómez-Pérez A.; Peters, W. "LexOMV: an OMV extension to capture multilinguality". In: Proceedings of the International Workshop OntoLex07 - From Text to Knowledge. In ISWC07. November, 2007. Busan, South Korea

Additionally, this contribution is published as a Technical Report in:

- Palma, R.; Hartmann, J.; Haase, P. " OMV-Ontology Metadata Vocabulary for the Semantic Web". Version 2.4.1. March, 2009. Available at `http://omv.ontoware.org`

Some of the contributions of Chapters 5 and 6 (including the corresponding technological support presented in Chapter 7) have been published in:

- Palma, R.; Haase, P.; Corcho, O.; Gómez-Pérez, A. "Change Representation For OWL 2 Ontologies". In: Proceedings of the Fifth International Workshop OWL: Experiences and Directions. In ISWC09. October, 2009. Chantilly, Virginia, USA. To Be Published.

- Palma, R.; Haase, P.; Corcho, O.; Gómez-Pérez, A.; Ji, Q. "An Editorial Workflow Approach For Collaborative Ontology Development". In: Proceedings of the 3rd Asian Semantic Web Conference. ASWC 08. Bangkok, Thailand, December 2008. Springer.

Additional publications regarding the technological support presented in Chapter 7 are:

- Hartmann J.; Palma, R.; Gómez-Pérez, A. "Ontology Repositories". Book Chapter: Handbook of Ontologies (2nd edition). 2009. Springer. Berlin.

- Palma, R.; Haase, P. "Oyster - Sharing and Re-using Ontologies in a Peer-to-Peer Community". In: Semantic Web Challenge of Proceedings of the 4th International Semantic Web Conference, ISWC 2005. Volume 3729. November 2005. Galway, Ireland. Springer. 1059-1062.

- Palma, R.; Haase, P.; Gómez-Pérez, A. "Oyster - Sharing and Re-using Ontologies in a Peer-to-Peer Community". Demo Paper in the Proceedings of the 3rd European Semantic Web Conference 2006. ESWC'06. June, 2006. Budva Montenegro.

Additionally, parts of the contributions of Chapters 5 and 6 and the evaluation of Chapter 7 were originally published in the following technical reports of the European Project NeOn:

- Palma, R.; Wang, Y.; Haase, P.; D'Aquin, M. "D1.3.1. Propagation Models and Strategies". NeOn. Deliverable. November, 2007. Available at `http://www.neon-project.org/`.
- Palma, R.; Haase, P.; Ji, Q. "D1.3.2. Change management to support collaborative workflows ". NeOn Deliverable. December, 2008. Available at `http://www.neon-project.org/`

Chapter 2

STATE OF THE ART

In this chapter we present a summary of the existing models, methods and tools that are relevant for the management of changes in distributed and collaborative environments. In particular, the main topics of interest for this thesis include:

- ontology metadata
- ontology change management (in distributed environments)
- collaborative ontology development

To facilitate the reading of this chapter, first we briefly summarize the role of the aforementioned topics in the context of this thesis. Our overall goal is the management of ontology changes in distributed environments to support the activities of implementation and maintenance of collaborative ontology development in an organizational setting. In such a collaborative scenario, where an ontology is being modified by many different actors, change management is central. Consequently, as a first step we need to identify changes in an ontology. This can be achieved at different granularity levels:

- At a higher level, one should be able to determine *whether an ontology has changed or not*. Additionally, when an ontology has changed, it should be possible to have a *general overview* of how it has changed, for example, the domain modeled has been specialized, the number of classes has increased, additional parties have contributed to the ontology development, or the content has been made available in additional languages. In order to provide the previous information we need a vocabulary for describing ontologies, i.e., **ontology metadata**.
- At a lower level, one should be able to determine *which were the specific changes* that were applied to an ontology, for example, the classes added/deleted, the range of a property, the sub-classes of a class, the values of an individual properties and the deleted individuals. Depending on the level of abstraction, the types of changes may be different, ranging from

atomic changes (e.g., add/remove RDF triple) to composite ones (e.g., move subtree). Providing the previous information is one of the goals of the **ontology change management** process.

Moreover, the management of changes involves several other activities, some of them particularly important to support collaborative ontology development in distributed environments. For instance, methods and strategies for *capturing, maintaining and storing changes* should be put in place to support collaborative scenarios. Similarly, strategies for the *propagation of changes* are crucial to support a distributed control of changes.

Finally, different methods and strategies should be taken into account during the **collaborative ontology development**. For instance, typically within an organization, a team of ontology editors follow a well-defined *process for the coordination of change proposals*. In this scenario, ontology editors can play different roles and are allowed to perform different activities, such as proposing changes or accepting/rejecting changes. Additionally, this process specifies *how changes should be applied*, for example, the policy used for the proposal of changes and curation activities, and the states that changes can have during this process. Similarly, as there can be different collaborative scenarios (e.g., distributed editors working with central copy of ontology or with local copies of the same ontology), it is necessary to have strategies supporting those scenarios.

Hence, in the following sections we provide an overview of the existing works on all these areas, focused on the main aspects of interest for this thesis.

2.1 Ontology Metadata

Metadata and metadata standards have a long-tradition in a variety of areas of computer science, such as digital libraries (e.g., Dublin Core[1] -DC- and Metadata Object Description Schema[2] -MODS-) or data management and maintenance systems (e.g., Metadata for Images in XML[3] -MIX- and PREservation Metadata Implementation Strategies[4] -PREMIS-). However, only a few proposals can be found for the domain of ontologies. We briefly mention some metadata models, focusing specially on those models relevant for the Semantic Web; we also review how those models are used in ontology repositories and registries.

2.1.1 Ontology Metadata Models

Previous efforts in the definition of metadata models relevant for the Semantic Web field range from general purpose ones to ontology specific ones. They can be summarized as follows:

[1] http://dublincore.org/
[2] http://www.loc.gov/standards/mods/
[3] http://www.loc.gov/standards/mix/
[4] http://www.oclc.org/research/projects/pmwg/

- The **Dublin Core (DC)** metadata standard is a simple element set for describing a wide range of networked resources[5]. It includes two levels: Simple (with fifteen elements) and Qualified, including an additional element as well as a group of element refinements (or qualifiers) that adapt the semantics of the elements for resource discovery purposes.

- The **Reference Ontology**[AGPLTP00] is a domain ontology that gathers, describes and has links to existing ontologies. However, its focus is to characterize ontologies from the user point of view, and it provides only a list of property-value pairs for describing ontologies.

- In [MMS+03a] the authors introduce the **ontology metaontology (OMO)** for describing domain ontologies. It is an adaptation of the meta-ontology proposed in [AGPLTP00], extended with specific aspects that are required for managing multiple ontologies. OMO includes simple properties such as creation date, ontology name, location, etc., and also relations to additional concepts that model related entities such as the creator of the ontology, the project and application field associated to the ontology and the terms describing what the ontology is actually about.

- Similarly, in [Jar05] the author presents the ontology metadata used in the **DogmaModeler** ontology engineering tool, which provides a specialization and an extension of the Dublin-Core metadata elements. Similar to the previous approaches, this model consists mainly of a list of property-value pairs for describing ontological resources.

- In [HPS+05a], [HPS+05b], we presented the **Ontology Metadata Vocabulary (OMV)**, which is a modular model intended to capture reuse-relevant information about ontologies in a machine-understandable form. OMV distinguishes between the OMV Core and various OMV Extensions, allowing representing the needs of the majority of ontology users, but at the same time supporting extensions and refinements for particular application scenarios. The OMV core consists of several classes, properties and relationships for describing various aspects related to the creation, management and usage of an ontology. OMV is described in detail in Chapter 4.

Table 2.1 summarizes the temporal evolution of the aforementioned metadata models.

The issue of creating a metadata standard for ontologies is also addressed by various ontology repositories/registries/search engines initiatives. However, the majority of these systems rely on a restricted, implicitly declared vocabulary, whose meaning is, in many cases, not machine-understandable. For instance, the Semantic Web search engine **SWOOGLE**[6] [D+04] makes use of an implicitly

[5] c.f. `http://dublincore.org/`

[6] c.f. `http://swoogle.umbc.edu/`

Table 2.1: Temporal Evolution of Ontology Metadata Models

Metadata Model	Year
Dublin Core	1998
Reference Ontology	2000
Ontology Metaontology (OMO)	2003
DogmaModeler Ontology	2005
Ontology Metadata Vocabulary (OMV)	2005

defined metadata schema, which covers information that can be extracted automatically from ontology implementations (e.g., ontology syntax, file length, number of triples and instances). The **DAML ontology library**[7] provides a catalog of DAML ontologies that can be browsed by different properties (e.g., keywords, classes and properties). The **FIPA ontology service** [S$^+$01] defines an agent wrapper of open knowledge base connectivity, and the **SchemaWeb Directory**[8] is a repository for RDF schemas expressed in RDFS, OWL and DAML+OIL that allows browsing ontologies, querying RDF triples and displaying some details of the ontology (e.g., name, description, contact name and local version). The **KnowledgeZone** ontology repository[9] uses an ontology of features (metadata) that characterizes an ontology (including the ontology domain, version, author, contact, representation language, etc.). Also, the biomedical ontology repository **BioPortal**[10] maintains a small set of ontology metadata, such as the name, format, category, and description. Similarly, the Semantic Web Gateway, **Watson**[11] uses also an implicitly defined metadata schema, similar to SWOOGLE, including the size of the file, the number of statements, the representation language, etc.

Table 2.2: Temporal Evolution of Ontology Metadata-Aware Systems

System	Year
DAML Ontology Library	2000
FIPA ontology service	2001
SchemaWeb Directory	2003
SWOOGLE	2004
Oyster	2005
KnowledgeZone	2007
Watson	2007
BioPortal	2008

[7] c.f. `http://www.daml.org/ontologies/`
[8] c.f. `http://www.schemaweb.info`
[9] `http://smi-protege.stanford.edu:8080/KnowledgeZone/`
[10] c.f. `http://bioportal.bioontology.org/`
[11] `http://watson.kmi.open.ac.uk/WatsonWUI/`

Finally, **Oyster**[12] (described in section 7.1.1) is an ontology metadata registry that implements OMV as the way to describe ontologies and related entities, supporting their exchange and re-use.

Table 2.2 summarizes the temporal evolution of the aforementioned systems.

2.1.2 Ontology Repositories and Registries

Repositories have evolved throughout time from general purpose data repositories to specialized ontology repositories. In the literature there are many different meanings and definitions about what a data repository is, and in general, what a repository is. Hence we will first discuss what we understand by a data repository, instead of giving another definition. We consider a data repository as a collection of digital data that is available to one or more entities (e.g., users and systems) for a variety of purposes (e.g., learning, administrative processes and research) and that has the characteristics proposed by Heery R. and Anderson, S. [HA05]:

- Content is deposited in a repository, whether by the content creator, the owner or a third party.
- The repository architecture manages content as well as metadata.
- The repository offers a minimum set of basic services, e.g., put, get, search and access control.
- The repository must be sustainable and trusted, well-supported and well-managed.

The term data library is usually used in the literature to refer to subject specific datasets (e.g., climate data library, time series data library and geospatial data library). Moreover, a data library tends to house local data collections and provides access to them through various means. Thus, in general a data library usually provides access to the complete dataset instead of providing the basic services (e.g., search, put and get) that a data repository offers.

Around the middle of 1990s, the term digital library (previously also known as electronic library or virtual library) was first made popular by the NSF/DARPA/NASA Digital Libraries Initiative. According to [Arm01] a digital library is a managed collection of information, with associated services, where the information is stored in digital formats and accessible over a network. The information stored can be very diverse and used by many different users. In general, a digital library is considered similar to a traditional library, i.e., it is used by users to find information that others have created; this end-users may use this information for study, reference, or entertainment. However, digital libraries take advantage of the new technologies to deliver the information to users.

[12] `http://oyster.ontoware.org`

Data warehouses [Inm02] became popular during the late 1980s and early 1990s. The purpose of a data warehouse is to perform analysis of the stored data for the management's decision making. Data is entered into this repository periodically, usually in an append-only manner. A data repository however, does not necessarily have the analysis functionality that a data warehouse provides.

Similarly to data repository, it is also possible to find many different meanings and definitions of what a knowledge base is. Yet, in general, a knowledge base is a central repository of knowledge artifacts. Usually a knowledge base may use an ontology to formally represent its content and its classification scheme, but it may also include unstructured or unformalized information expressed in natural language or procedural code. Also, in contrast to a data repository, usually the purpose of the knowledge base is to allow automated deductive reasoning over the stored knowledge, i.e., to decide how to act by running formal reasoning procedures over its knowledge.

It is not surprising that some years ago, the ontology and semantic web community became interested in using repositories to hold semantic content (e.g., ontologies). In recent years, ontologies have been widely developed and applied in many domains, especially in the context of the semantic web. The academic and industrial world are developing and using ontologies to provide new technologies and to support daily operations. Therefore, currently there exists a large amount of ontologies developed by many different parties, which makes necessary to have the means for sharing and reusing them.

Initial efforts to collect the base of existing ontologies proposed the creation of library systems (known as Ontology library systems) that offered various functions for managing, adapting and standardizing groups of ontologies [DF01]. These systems defined an environment for grouping and reorganizing ontologies for further re-use, integration, maintenance, mapping and versioning. They defined an evaluation model based on the functionality that the library system provided. Some examples of library systems are: WebOnto[13], Ontolingua[14] DAML Ontology Library System[15] and SchemaWeb[16].

Currently, efforts are put in the creation of ontology repositories. An **ontology repository** is similar to what Ding et. al. defined as an ontology library system [DF01], though there are some differences between them. In our work in [HPGP09] we defined an Ontology Repository (OR) as a structured collection of ontologies (schema and instances), modules, and additional meta knowledge that uses an ontology metadata vocabulary. The references and relations between ontologies and their modules build the semantic model of an ontology repository. Access to resources is realized through semantically-enabled interfaces applicable to humans and machines. Therefore, a repository provides a formal query language. Unlike the previous notion of ontology library systems ([DF01]), we

[13] `http://projects.kmi.open.ac.uk/webonto/`

[14] `http://www.ksl.stanford.edu/software/ontolingua/`

[15] `http://www.daml.org/ontologies/`

[16] `http://www.schemaweb.info`

require a more complete set of functionalities for an ontology repository, including personalized visualizations, rating services, mapping mechanisms, trust management, access control and right management functionalities, ontology modules support, evolution mechanisms, metadata support, registry services, etc. We also defined the software to manage an ontology repository as Ontology Repository Management System (ORMS). An ORMS is a system to store, organize, modify and extract knowledge from an Ontology Repository. Both the OR and the ORMS constitute what we called the Generic Ontology Repository Framework (GORF).

Some examples of ontology repositories are ONTHOLOGY [Har06], KnowledgeZone [SRNM07] and the Semantic Web Gateway, Watson[dBG+07]; the last one provides ontology repository functionalities, among others.

Recently, several initiatives have been started in order to build public ontology repositories. The Ontology PSIG[17] of the OMG together with other OMG subgroups is developing a repository of Ontologies and Vocabularies. In a similar attempt, the Ontology Summit 2008 has started an initiative to build an OOR (Open Ontology Repository)[18] to provide an architecture and an infrastructure that support the creation, sharing, searching, and management of ontologies.

Directly connected to the concept of ontology repositories, is the concept of **Ontology Metadata Registries** (OMRs); OMRs provide services for storing, cataloging, discovering, managing, and retrieving ontology metadata definitions. OMRs provide the means to support advanced semantic searches of ontologies based on their characteristics. In general, the OMR can be a component of the ontology repository or an independent system.

In analogy to the Dublin Core Metadata Initiative's (DCMI) Metadata Registry[19], OMRs are designed to promote the discovery and reuse of existing ontologies. They provide users and applications with an authoritative source of information about the characteristics of ontologies, thus simplifying the discovery process.

Some examples of existing ontology metadata registries are Oyster [PH05] (see section 7.1.1), the DCMI Registry[HW02], or the (Onto)2Agent [AGPLTP00].

To summarize and in order to avoid confusions, it is important to make clear the difference between an ontology metadata registry and an ontology repository. Simply put, the former stores descriptions of ontologies (e.g., name, type of ontology, domain of ontology, creator and number of classes), while the latter stores the ontologies themselves.

[17] http://www.omg.org/ontology/

[18] http://ontolog.cim3.net/cgi-bin/wiki.pl?OntologySummit2008_Communique

[19] http://dublincore.org/dcregistry/

2.1.3 Limitations on Ontology Metadata Models, Repositories and Registries

The need for a metadata vocabulary for describing ontologies has been acknowledged in the past by previous efforts. However, at the moment, most of the current ontologies exist in pure form without any additional information, e.g., domain of interest, authorship information and statistic information. This is due in part to the **lack of standards or community-accepted vocabularies** for documenting and annotating ontologies with metadata information. Moreover, most of the previous efforts carried out on this issue provide only a **list of property-value pairs** for describing ontologies, limiting the processing capabilities and the related relevant information that can be described. Similarly, ontology metadata is, in most cases, **implicitly defined** within existing systems and repositories (e.g., SWOOGLE, Watson, KnowledgeZone and bioPortal), which makes it difficult to integrate or use. Finally, **general purpose standards**, such as Dublin Core, **are not appropriate** for capturing information about ontologies because of the differences between arbitrary information sources and ontologies. For instance, aspects related to the application scenario, scope, purpose, or evaluation results are essential when describing ontologies. Additionally, besides structural and technical information, ontologies have to be described in terms of descriptive metadata, such as provenance information, ontology categorizations, underlying methodologies or knowledge representation paradigms that are specific for ontologies.

Regarding the repositories and registries, similar conclusions can be drawn. One of the main drawbacks is that, due to the lack of standards for ontology metadata, generally the existing tools **make use of a implicit defined metadata schema**. Similarly, current repositories provide only a **limited set of search and navigation services** (e.g., keyword-based queries), usually offering a simple Web interface to the ontological resources. Consequently, **integrating existing repositories** with other systems (if possible at all) to find/use ontological resources is usually limited by simple API's/Web Services that **do not exploit ontology metadata**. Finally, most of the existing tools are **centralized**. While a centralized model offers many advantages, it is also possible to find interesting scenarios where a distributed approach can be useful (e.g., users working with local copies of the resources).

2.2 Ontology Change Management

According to [Yil06] and [Sto04], change management in ontologies can take several forms:

- Ontology Modification - it allows changes to the ontology that is in use, without considering the consistency.
- Ontology Evolution - it facilitates the modification of an ontology by preserving its consistency.

- Ontology Versioning - it allows handling ontology changes by creating and managing different versions of the ontology ([KF01]).

In this work we have focused on the last two aspects, since ontology modification refers to changing the ontology without bothering about its consistency, and hence it is not the case we have considered for our collaborative ontology development scenarios.

Ontology Evolution has been defined in more detail by [Sto04] as the timely adaptation of an ontology to the arisen changes and the consistent propagation of these changes to dependent artifacts. Similarly, in [FP05], ontology evolution is defined as the process of modifying an ontology in response to a certain change in the domain or its conceptualization. And in [HvHH+05], the authors define the consistent ontology evolution as the process of managing ontology changes by preserving the consistency of the ontology with respect to a given notion of consistency defined in terms of consistency conditions, or invariants that must be satisfied by the ontology.

On the other hand, ontology versioning has been defined also in [LM07] as a mechanism that allows users to keep track of all changes in a given system (e.g., history), and to undo changes by rolling back to any previous version.

In most proposals, ontology evolution and versioning are clearly separated, although there are some cases in which they are defined as a single concept. For instance, in [Kle04] and [NK04], the authors define this single concept as the ability to manage ontology changes and their effects by creating and maintaining different variants of the ontology. Similarly, in a previous work presented in [Hef01], the author proposes that one possible solution to the ontology evolution problem is to require that revisions (variants) of ontologies be distinct ontologies in themselves (different versions), so each resource can commit to a particular version of an ontology.

2.2.1 Ontology Evolution and Versioning Approaches

2.2.1.1 Ontology Evolution

Although, the evolution of conceptual models such as schemas in databases or XML schemas, has been thoroughly investigated in the computer science field, the evolution of ontologies is still under continuous research. Next, we provide the most relevant works in this area.

In [Sto04], the author proposed a six-phase process that addresses single and distributed ontology evolution (depicted in Figure 2.1):

- The *change capturing* phase carries out the continual improvement of an ontology. It distinguishes two types of changes: top-down changes and bottom-up changes. The top down (deductive/explicit) changes are the result of the

knowledge elicitation techniques that are used to acquire knowledge direct from human experts. Bottom-up (inductive/implicit) changes are those derived from machine learning techniques, which use different methods to infer patterns from a sets of examples. The implicit changes are reflected in the behavior of the system and can be discovered only through the analysis of this behavior (e.g., Structure-driven change discovery, Data-driven change discovery and Usage-driven change discovery).

- The *change representation* phase is in charge of representing formally and explicitly a request for a change as one or more ontology changes. This formal representation of ontology changes makes them machine-understandable, usable by other ontology evolution systems and exploitable for supplementary functionality of an ontology evolution system such as learnability.

- The *semantics of change* phase prevents inconsistencies by computing additional changes that guarantee the transition of the ontology into another consistent state. It enable the resolution of induced changes in a systematic manner, ensuring consistency of the whole ontology. In particular, the author focuses on the structural inconsistencies that arise when the ontology model constraints are invalidated after a change request.

- The *change propagation* phase is in charge of updating all of the ontology dependent artifacts. The related artifacts considered by the author include distributed ontology instances, dependent ontologies and applications using the ontology that has to be changed.

- The *change implementation* phase has as a role to inform the ontology engineer about all the consequences of a change request, to apply all the (required and derived) changes to the ontology in a transactional manner, and to keep track of the changes performed.

- Finally, the *change validation* phase enables to justify the changes performed (e.g., generation of explanation) and to undo them at the user's request (also known as reversibility). This phase helps ontology engineers to find out whether they have built the right ontology, i.e., whether the ontology changed represents a piece of reality, the users' requirements and/or the application requirements correctly.

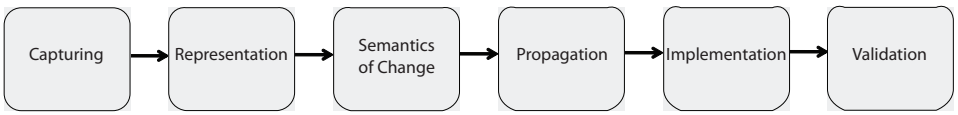


Figure 2.1: Ontology Evolution Process ([Sto04])

Another approach is given by [KN03], where the authors introduce a component-based framework for ontology evolution based on the different representations of ontology changes. The framework integrates these representations and covers the following tasks:

- *Data Transformation.* In this task, when an ontology version V_1 is changed to V_2, the data described by V_1 is translated so as to bring it in line with V_2, if necessary.
- *Data Access.* This task ensures that all data accessible via queries in terms of V_1 is retrieved with queries in terms of V_2.
- *Ontology Update.* In this task, when a remote ontology is adapted to specific local needs, and this remote ontology changes, those changes must be propagated to the local ontology adapted.
- *Consistent Reasoning.* This task analyzes the changes occurred in an ontology to determine whether specific axioms that were valid in V_1 are still valid in V_2.
- *Verification and Approval.* When necessary (e.g., developing an ontology collaboratively), developers should be able to verify and accept or reject specific changes, enabling the execution of some changes and rolling back others.

An approach for modeling the ontology evolution process from the single user ontology evolution point of view is presented in [LM07]. This approach is based on the work proposed in [BR00], which distinguishes four generic activities in the process of making any change to any type of artifact that is subject to changes. Therefore, interpreting this process in the context of ontology evolution, they propose the following activities:

- *Requesting a change* has to do with initiating the change process and it includes activities related with the representation of changes, prioritization of multiple changes, and the discovery of changes.
- *Planning the change* has to do with understanding *why* the change needs to be made, and *where* the change needs to be made. Therefore, a crucial part of this activity has to do with the change impact analysis where all the potential consequences (side effects) of a change are identified along with an estimation of what needs to be modified to accomplish a change. Finally, an estimated cost of evolution is provided to allow the engineer to decide whether to implement the change or not.
- *Implementing the change* involves the activities of change propagation, restructuring the ontology before the implementation of the change, and the inconsistency management.

- Finally, *verifying and validating* the change deals with the issues of building the right ontology and building it in a right way. Then, the activity of quality assurance will ensure that the developed ontology satisfies all the desired qualities.

According to [KFAC07], the most critical part of an ontology evolution algorithm is to determine *what* can be changed and *how* each change should be implemented. The authors claim that this determination can be split into the following five steps, which, although unrecognized, are shared by many evolution frameworks:

- *Model Selection.* Since the selection of the model may have critical effects on what can be changed, i.e., the changes allowed are constrained by the expressive power of the ontology representation model, this is an important parameter of the evolution algorithm.
- *Supported Operations.* Here, the change operations allowed upon the ontology are specified.
- *Consistency Model.* The consistency model assumed constraints the possible consistency problems of the resulting ontology that may arise whenever a change operation is executed.
- *Inconsistency Resolution.* In this step, the different actions that can be performed to restore the consistency of the ontology are determined for each operation and inconsistency problem supported.
- *Action Selection.* Finally, a selection process determines the most preferable action among the various potential actions identified in the previous step.

The first two steps determine *what* can be changed and roughly correspond to the change capturing phase introduced in [Sto04], while the last three steps indicate *how* changes should be implemented and roughly correspond to the semantics of change phase of [Sto04].

As a summary, we can say that the works mentioned focus on the steps to be followed in order to support the ontology evolution process, and that although they model this process in different ways (using different names and number of steps), in general, they identify very similar tasks (e.g., change representation, inconsistency management and ontology validation). Nevertheless, they differ in their focus. For instance, [Sto04] focuses on the consistent propagation of changes to ontology related artifacts, particularly to dependent ontologies and ontology instances, while the focus of [KN03] is on the representation of ontology changes and on the management of ontology versions. The work described in [LM07] provides only high-level practices for changing artifacts and is centered on the user-point of view. Finally, [KFAC07] focuses only on the task of identifying what can be changed and how it should be implemented.

2.2.1.2 Ontology Versioning

Although there is a clear distinction between schema evolution and schema versioning in the database community, this distinction is not so clear when applied to ontologies (ontology evolution and ontology versioning) and in fact they can easily lead to confusion. For instance, with respect to databases, according to [Rod95], schema evolution is the ability to change a schema of a populated database without loss of data, i.e., providing access to both old and new data through the new schema, whereas schema versioning is the ability to access all the data (both old and new) through different version interfaces. However, as aforementioned, with respect to ontologies, some authors claim that one cannot distinguish between evolution and versioning and they combine them into a single concept. For instance, in [Kle04] and [NK04] the authors propose that developers should maintain not only the different versions of an ontology, but also some information on how the versions differ and whether they are compatible with one another or not.

In other works (e.g., [HSV04]), ontology versioning is considered a stronger variant of handling changes to ontologies. The authors claim that whereas ontology evolution is concerned with the ability to change an ontology without losing data and maintaining consistency, ontology versioning allows accessing the data through different instances of the ontology that are valid at a certain point in time. Hence in ontology versioning, it is important to manage individual versions of the ontology as well as the derivation relations between the versions (e.g., compatibility between versions, mapping relations between versions, or transformations of data corresponding to the various versions).

General Frameworks for Ontology Versioning

In [KF01] the authors argue that they need a methodology with methods to distinguish and recognize versions and with procedures to update and change ontologies, keeping track of the relationships between versions. The authors impose three requirements for a versioning framework: (i) ontology identification; (ii) change specification, i.e., to make explicit the relations between versions; and (iii) transparent evolution, i.e., to translate and relate the versions and data sources. In [KFK$^+$02], the authors extended this list with two additional requirements to distinguish between the effect of changes on different tasks and to determine the compatibility of different versions for which there is no trace of the changes that led from one version to another, namely, (iv) task awareness and (v) support for untraced changes.

The authors in [NKKM04] introduce some design principles for an ontology-versioning environment: automatic comparison of versions (at the conceptual level), identification of complex changes, contextual presentation of changes (to see the change itself and to see its context, too), navigation among changes, access to old and new values, mechanism for accepting and rejecting changes, and different levels of granularity for accepting and rejecting changes.

Ontology versioning has been classified in different types:

- [Lia06] classify existing ontology versioning methods into two groups: passive ontology versioning and active ontology versioning (also defined in [NK04] as untraced and traced ontology evolution modes). Passive ontology versioning methods intend to analyze ontology changes based on the comparison of existing versions of the same ontology (e.g., [Kle02] and [NKKM04]). Active ontology versioning intends to keep every record of the changes happened on the ontologies from the beginning of updating process (e.g., [NCLM06] and [Sto04]).

- In [LM07], the authors also identify two variants of versioning: state-based versioning (at any given moment in time, the system under consideration is in a certain state, and any change made to the system will cause the system to go to a new state) and change-based versioning, where changes are treated as first-class entities (the information about the precise changes that were performed is stored).

Versioning Support in Different Ontology Languages

Early efforts in ontology versioning can be found in [HH00] where the authors present SHOE, a web-based knowledge representation language that supports multiple versions of ontologies, giving support to the management of the effects of ontology revision on SHOE web pages and providing methods for implementing ontology integration using SHOE's extension and version mechanisms. SHOE also enables ontology developers to state whether a version is backward-compatible with an older version or not.

In [OK02], the authors propose a model for the tracking of changes, versioning, and meta-information for RDF(S) repositories. The approach is based on the fact that the RDF statement, which is the smallest directly manageable piece of knowledge, cannot be changed, i.e., it can only be added and removed. Hence, two basic types of updates in a repository, namely, addition and removal of a statement, are tackled here. Each update changes the repository into a new state, which is defined as the set of statements that are explicitly asserted. Some of the states can be in turn considered as versions (depending on the needs and desires of the users and/or applications) for which additional knowledge could be supported as a meta-information. The meta-information is supported for resources, statements, and versions. The authors only consider a small set of meta-information, but offer the possibility to extend it on demand.

The structural specification and functional-style syntax of the emerging OWL 2 ontology language addresses the issue of versioning in the current working draft[20]. It proposes a simple mechanism for versioning OWL 2 ontologies. In particular, ontologies may have an ontology IRI (International Resource Identifiers[21]), which

[20] http://www.w3.org/TR/2008/WD-owl2-syntax-20081202/

[21] http://www.ietf.org/rfc/rfc3987.txt

is used to identify an ontology. If an ontology has an ontology IRI, the ontology may additionally have a version IRI. The set of all the versions of a particular ontology (known as ontology series) is identified using the ontology IRI and each version in the series is assigned a different version IRI. Furthermore, the document[22] of the ontology representing the current version of the series should be accessible from the ontology IRI and, if present, also from its version IRI. The ontology documents of the previous versions should be accessible only from their respective version IRIs.

Finally, one interesting contribution is the discussion about the *identification of ontologies* presented in [KF01]. If an ontology is seen as a specification of a conceptualization, then every modification to that specification can be considered a new conceptualization of the domain. In that case, even syntactic changes and updates of natural language descriptions specify different concepts, which are per definition not equal (although they are fully compatible revisions). The authors take the following (practical) position: they assume that an ontology is represented in a file on the web. Every change that results in a different character representation of the ontology constitutes a revision. In case the logical definitions are not changed, it is the responsibility of the author of the revision to decide whether this revision is a semantic change and thus forms a new conceptualization with its own identity, or just a change in the representation of the same conceptualization.

In any case, there has to be a way to deal with the identification of ontologies on the web. Hence, the authors propose a method to separate the identity of ontologies (ontology resources) completely from the identity of files (file resources) on the web that specify the ontology. The idea is that every revision is a new file resource and gets a new file identifier, but it does not automatically get a new ontology identifier. The authors rely on the assumptions that an ontology can be regarded as a resource that can be identified by a Uniform Resource Identifier (URI) [BLFM05], and that URIs provide a general identification mechanisms, as opposed to Uniform Resource Locators (URLs), which are bound to the location of a resource.

The proposed method, however, is not compliant with the RDF Schema specification[BG04] which states that a new namespace URI should be declared whenever an RDF schema is changed. Yet, this recommendation seems too strong and has already showed problems in practice (e.g., Dublin Core). Hence the identification method proposed is based on the following points:

- a distinction between three classes of resources: (i) files; (ii) ontologies; and (iii) lines of backward compatible ontologies;
- a change in a file results in a new file identifier;
- the use of a URL for the file identification;
- only a change in the conceptualization results in a new ontology identifier;

[22] Each ontology is associated with an ontology document, which physically contains the ontology stored in a particular way

- a new type of URI for ontology identification with a two level numbering scheme (mayor.minor[23]):
  - minor numbers for backward compatible modifications (an ontology-URI ending with a minor number identifies a specific ontology);
  - major numbers for incompatible changes (an ontology-URI ending with a major number identifies a line of backward compatible ontologies);
- individual concepts or relations, whose identifier only differs in minor numbers, are assumed to be equivalent;
- ontologies are referred to by an ontology URI with the according major revision number and the minimal extra commitment, i.e., the lowest necessary minor revision number.

To summarize, ontology versioning has been addressed in the past in many different ways. General works focus on clarifying the relationship between ontology versioning and evolution (e.g., [Kle04] and [HSV04]). Some other works focus more on the definition of a general framework that characterizes what ontology versioning should deal with (e.g., [KF01], [KFK+02] and [NKKM04]), or on the characterization of different types of ontology versioning (e.g., [Lia06], [NK04] and [LM07]). Finally, in other works, the authors focus on supporting ontology versioning in different ontology languages, such as SHOE (e.g., [HH00]), RDF (e.g., [OK02]) and OWL2 (e.g., OWL2 specification).

2.2.2 Change Management Models, Methods and Strategies

As can be seen from the previous section, both ontology evolution and ontology versioning address the problem of change management. In this thesis, we are mainly interested in ontology evolution activities (also known as traced evolution mode or active ontology versioning) because we intend to keep record of the changes in ontologies and the scenarios considered are those where ontologies change without losing data and maintaining consistency. Nevertheless, we have also considered the management of different ontology versions.

Even though the previous ontology evolution approaches model this process in a different way, they identify very similar activities, as described in section 2.2.1.1. In all cases, change representation is fundamental and so are the activities related to the discovery of ontology changes, the potential side-effects of those changes on related artifacts, and the verification of the ontology consistency after applying

[23] In software versioning schemes, usually the major number is increased when there are significant jumps in functionality, while the minor number is incremented when only minor features or significant fixes have been added

them. Finally, besides the activity of the actual implementation of the changes, some of the approaches introduce the activity of the propagation of changes. In this section we focus on the change representation and change propagation activities since they are of great interest for this thesis.

2.2.2.1 Change Representation

There are some approaches in the literature for the formal and explicit representation of ontology changes. Much of the current work is focused on devising taxonomies of elementary change operators that are sound[24] and complete[25], and, typically, the outcome of this activity is an ontology for representing ontology changes, i.e., change ontology.

In [Sto04], the author describes a change ontology that is used as a backbone for creating evolution logs. This ontology models what changes, why, when, by whom and how changes are performed in an ontology. The changes are classified as elementary, composite and complex. An elementary change is an ontology change that modifies (adds or removes) only one entity of the ontology model. A composite change represents a group of elementary changes applied together. A complex change is an ontology change that can be decomposed into any combination of at least two elementary and composite ontology changes (e.g., a list of 12 complex changes is presented in [SMMS02]). In addition to the hierarchy of ontology changes, which is based on the KAON ontology model, the evolution ontology includes information such as cost, relevance, priority, textual description of the reason for a change and others. Furthermore, there are properties establishing the relationships between changes (e.g., previousChange) along with their corresponding inverses. Finally, there are additional properties that depend on the change type (e.g., hasReferenceEntity). These properties are used to represent the peculiarities of a particular type of a change, for example, its arguments.

Another similar change ontology, focussed on the OWL language, is described in [Kle04]. This ontology models the relations between the most important concepts around ontology changes. Changes are classified as atomic or composite. Atomic operations are operations that cannot be subdivided into smaller operations and constitute the top-level concept of the hierarchy of change operations for the OWL language. Composite operations provide a mechanism for grouping a number of basic operations that together constitute a logical entity. Each change is related to the old and new version of a definition. Also, the change ontology represents a set of change operations as an entity that has a source and target ontology. Finally, the ontology includes properties that specify arguments for a change operation as well as a property effect to annotate the class of operations with the effect of the change.

This change ontology is based on previous efforts by the same authors and other colleagues ([KFK+02], [KN03], [NK03], [SK03]) and it has been extended

[24] The manipulation operators should only generate valid ontologies.

[25] The set should subsume every possible type of ontology access and manipulation.

in later works ([NCLM06]):

- In [KFK+02], a change ontology for the OWL-light knowledge model is presented. At the highest level, the ontology makes a distinction between atomic changes and composite changes. Some of the composite changes are named, others are just unnamed aggregations of atomic changes. The named composite changes make it possible to specify an effect that is different than the combination of the effects of composite changes. Nevertheless, the effect and semantics of the composite change is not clearly specified, since the ontology is modeled in RDFS.

- [KN03] describes an ontology of change operations for the OWL knowledge model that consists of two parts: the basis is an ontology of basic change operations (which is essentially the same as the one in [Kle04]) and there is an extension that defines complex change operations (in [SK03] the author claims that their ontology of changes include 120 change operations (basic and complex changes) and that the list is still growing as new complex operations are defined). Complex operations are defined as operations composed of multiple basic operations or that incorporate some additional knowledge about the change. However, the extension is just a (non-hierarchical) list of some complex change operations that model specific variants of complex changes as subclasses of the root change class. The authors propose two methods for finding such complex changes: on the one hand, using a set of rules to generate a complex change from a set of basic changes. On the other hand, using heuristics to determine if a complex change occurred.

- The work in [NK03] focuses on the identification of a set of common complex changes combining the information of the structural differences between two versions of an ontology[NM02] with the hierarchical information. The complex changes identified include class moves and tree-level operations. However, those changes are not formally and explicitly represented and, therefore, they are not clearly defined.

- [NCLM06]) presents the Change and Annotation Ontology (CHAO) to represent changes between two versions of an ontology and user annotations related to these changes. The ontology contains two main classes: the class Change to represent changes in the ontology (reusing the previous works) and the class Annotation to store related annotations on changes.

Some additional proposals that focus on more specific aspects can also be found in the literature. In [Lia06] the authors propose a Log Ontology to represent ontology changes. Similar to the ontology of changes in [KFK+02], it models a hierarchy of the possible change operations during the ontology evolution process but it differs also in some aspects: the Log Ontology only considers changes

on classes and properties, it does not consider changes on the individuals. Also, the Log Ontology only considers elementary changes even though the ontology hierarchy includes modifying and renaming operations. Finally, the perspective application makes the Log Ontology different to Klein's Ontology of Change. The Log Ontology is used as a concept structure to organize the change information among the various versions of the same ontology in order to make the change process traceable.

Another work for OWL DL ontologies was introduced in [HSV04] where the authors define the atomic change operations of adding and removing axioms. As in previous approaches, composite ontology change operations can be expressed as a sequence of atomic ontology change operations. The work, however, does not elaborate further the idea proposed and does not refer to any concrete implementation. The representation of atomic changes consisting of adding/removing axioms, was also treated in [HvHH$^+$05], though the authors focused only on this possible representation.

Finally, a different approach is presented in [FP05]. The authors here propose instead of creating a list of all possible change operations, a logic-based approach, heavily influenced by the related field of belief revision (or belief change) to automatize (as much as possible) the way of dealing with the representation of changes (and the semantics of the change). They claim that we need a method to represent all possible changes in a uniform way. However, they do not provide the concrete solution though they propose the introduction of four different ontology evolution operations, namely, ontology revision, ontology contraction, ontology update and ontology erasure. In all those cases, the change is represented with a set of DL axioms.

To summarize, we can identify two main works for the representation of changes: [Sto04] and [Kle04]. They classify changes in a similar manner (atomic or elementary, composite and complex), where the atomic changes refers to the operations at the entity level. However, besides the different underlying ontology model used by the previous approaches (KAON and OWL), there are other differences between these approaches, such as the representation of *modify* operations at the lower level, or the way changes are related to the associated ontology elements. Some other works in the literature resemble (e.g., [KFK$^+$02] and [Lia06]) or extend (e.g., [KN03] and [NCLM06]) the previous models or provide partial solutions (e.g., [NK03], [HSV04] and [FP05]).

Change Representation in related fields

The problem of representing changes has also been addressed in the past in other related fields. In [BKKK87] we can find a taxonomy and semantics of schema changes in object-oriented databases. The authors propose a classification of elementary changes including add, drop and change operations, based on the schema

changes supported by the object-oriented database system called ORION. Similar sets of elementary changes for object-oriented databases can be found in the literature (e.g., [Zic92]).

Another proposal for schema evolution in object databases is [PK97]. The authors provide a complete classification of modifications and propose three levels of modifications: primitive, composite and complex. They discuss the need for the complex changes, but they provide neither a catalogue of these changes nor a means to form and resolve them. The same three levels of modifications were reused by [Sto04] but tailored for KAON ontology model.

Furthermore, in [OSSM99] the authors introduce a set of change operations, their semantics and a change-documentation model as part of their proposed CONCORDIA model for controlled medical vocabularies. The set of changes consists of a list of possible operations classified as changes that ”do” and ”do not” affect the hierarchy. The CONCORDIA model, further elaborated in [Oli00], comprises (i) a structural model that defines a set of concept data elements and constraints (information required by concepts/attributes such as name, unique id, etc.); (ii) a change model that defines a set of change operations and their semantics (e.g., add concept, retire concept and merge); and (iii) a log model that defines a set of log-file data elements and constraints (information required of a completed change, such as timestamp, author, explanation, etc.).

In [Ler00] the author introduces models for different levels of granularity in change operators for schema evolution, namely, non-elementary or compound change operators. In particular, the author presents a model of change types incorporating changes local to individual types (the object changes can be performed by modifying each object in isolation) as well as compound changes involving multiple types. The model describes both change types and their impact on data by defining derivation rules; this rules initialize new data based on the existing data. They can also describe local and nonlocal changes to types to capture the intent of a large class of change types operations.

The representation of changes has also been tackled for XML data. For instance, in [Fon01] the authors propose a way to represent changes to XML data using XML itself. In particular, they propose an XML delta format for identifying and representing XML changes. The delta should be a well-formed XML file where only the ”minimum” changes should be represented. Hence, they propose to use a few attributes (e.g., d:delta, d:new-attribute and d:old-attribute) and elements (e.g., d:PCDATAmodify and d:exchange) in the delta file to provide all the information needed to represent changes.

Finally, in [CAW98] the authors discuss how to represent and query changes in semi-structured data. According to these authors, semi-structured data is any data that has some structure, but that may be irregular or incomplete and that does not necessarily conform to a fixed schema (e.g., HTML documents). They present a general model, DOEM, for the representation of changes. DOEM (Delta-OEM) is based on OEM (Object Exchange Model) [PGmW95], a simple graph-based data model with objects as nodes and object-subobject relationships represented with

labeled arcs. DOEM uses annotations on the nodes and arcs of the OEM graph to represent changes. These authors consider four basic change operations (create node, update node, add arc and remove arc) and propose a one-to-one correspondence between them and the annotations.

2.2.2.2 Ontology Change Propagation

The issue of propagating changes has already been addressed in related areas (e.g., databases and distributed systems). However, for the ontology domain, only a few works tackle the propagation of ontology changes and in most of them, the issue is just partially addressed.

The propagation of changes to related ontologies has been elaborated in some works by different means. In [Oli00] the authors present an approach for the management of changes that comprises the ability to update copied (and possibly changed) versions of controlled vocabularies (e.g., ontologies) with a remotely changed controlled vocabulary (for the medical domain). For a local site, there is a tradeoff between having autonomy over a local vocabulary and conforming to a shared vocabulary to obtain the benefits of interoperation. If the local site is motivated to conform, then the burden lies with the local site to manage its own changes and to incorporate the changes of the shared version at periodic intervals. The author calls this process *synchronization* and is defined as the application of shared-vocabulary changes to the modified local version of a shared vocabulary in order to reach a target state. The process is based on the CONCORDIA model, which comprises a structural model, a change model, and a log model to which the shared and local vocabularies conform (see previous section). Since the goal of this process is to update the local vocabulary to obtain the benefits of shared-vocabulary updates while maintaining local changes that serve local needs, it can be considered as a distributed ontology evolution process. However, the underlying model is very simple, which results in few types of changes.

Using the previous approach, [Kle04] describes how the minimally required elements of the synchronization process based on the CONCORDIA model (structural model, change model and log model) that is specific for the medical scenario can be applicable for a subset of all possible evolution scenarios and ontologies in OWL. Also, there is a discussion on the extensions that should be made to make it applicable to a broader set of ontologies. However, according to the author, generally, the synchronization approach proposed is not applicable to an unrestricted setting.

In [SKKM02] and [SKKM03], the authors describe the management of the dependency between component ontologies in an ontology as a whole, i.e., an ontology is divided into several component ontologies. They consider two kind of dependencies between component ontologies (is-a relation and referring-to relation) and also the influence of a change of one ontology to others through its dependencies; finally, they design a function to suggest a few candidate modifications of the influenced ontology for keeping the consistency, i.e., propagate changes. This

approach considers that the components ontologies can be distributed and changed locally, but the management of the dependencies is done in a centralized manner.

A more complete approach is described in [Sto04]. The author here discusses how changes can affect related artifacts including dependent ontologies, instances, and applications and elaborates on the propagation to dependent ontologies (also in [MMS+03a] and [MMS03b]) by introducing first two basic building blocks for carrying out ontology reuse: ontology inclusion and ontology replication. The former allows reusing ontologies available within the same node, while the latter enables inclusion when ontologies are distributed on different ontology servers (nodes). For this task, the author proposes to replicate distributed ontologies locally and to include them in other ontologies. Replicated ontologies should never be modified directly. Instead, the ontology should be modified at the source and the changes should be propagated to replicas. The propagation of changes is handled at two different levels: propagation of changes for multiple dependent ontologies within a single node (called dependent ontology evolution) and the propagation of changes for multiples ontologies at multiples nodes (called distributed ontology evolution). For the dependent ontology evolution problem, the author proposes the synchronization between dependent ontologies using a push-based approach, i.e., changes from the changed ontology are propagated to dependent ontologies as they occur. Even more, since permanent consistency of ontologies within one node is necessary, changes are propagated immediately, that is, when they occur. With respect to the distributed ontology evolution, the author introduces two problems to solve: the propagation of changes from an original to its replica (called the replication ontology consistency), and the propagation of changes from a replica to an ontology that includes it (called the dependent ontology consistency). The solution proposed employs a hybrid synchronization strategy in which the first task is addressed using a pull approach for synchronizing originals and replicas and the second task is addressed using a push approach for maintaining the consistency of the ontologies within one node by applying the dependent evolution process to deltas, i.e., changes that have been applied to the original since the last synchronization of the replica.

The propagation of changes to ontology instances has also been elaborated on some works. In [SSH02a] and [SSH02b], the authors propose three steps for this process (called metadata evolution): (i) the *metadata capturing* step, which collects the instances (e.g., from the web) in the knowledge base. (ii) The *metadata analysis* step, in which the automatic translation of the instances is performed according to the changes in the ontology; this step provides an output in the form of list of modified instances with the reference to the corresponding resource (knowledge source). And (iii), the *generation of a proposal for modifications*, in which out-of-date instances on the Web are replaced with the corresponding up-to-date instances.

In [MPSd06] the authors propose an integrated approach to the evolution process of ontologies and related metadata, i.e., propagate ontology changes to ontology metadata. In particular, they discuss a methodology to capture the

evolution of metadata induced by changes to the ontologies, and viceversa. In this context, the term metadata is defined as concept instantiation in the form of the annotation of textual (or other forms of) data, and, therefore, it contains information about the linguistic content of the ontology. This kind of metadata is often called semantic metadata or annotations. The authors propose to classify the changes according to those derived from the concept, the instance or the property level. They study the effect on the instances caused by changes, describe the changes and the effect on the data (similar as in [KN03]) and propose the actions that should be taken: (i) to do nothing, (ii) manual action or (iii) automatic action. The second action requires some kind of re-annotation of the corpus, while the third action requires a set of procedures to be followed for automating the reclassification of instances in the ontology (the authors provide a few examples based on the GATE ontology API).

Finally, an approach to controlling the impact of ontology changes to dependent applications/services is presented in [LADS06], where the authors propose four stages to tackle the problem: first (i) changes between two versions of the ontology are *captured* manually; (ii) during the *management* stage a representation of ontology changes, called Log Ontology, is produced based on the changes identified in the first stage; (iii) the queries submitted from the applications/services are *analyzed* to find out their effectiveness and usability by checking each entity within the queries coordinating with the Log Ontology; (iv) during the *access* stage, if certain entities within the users' queries are found updated in the new versions, these entities would be replaced accordingly to form the new queries with updated entities, and then submitted to applications/service for execution; finally, (v) after the new-formed queries are submitted to the application or service for execution, the results are returned back during the *response* stage. Additionally, change/update information, including the new entity name, domain or range information, and change operations performed during the update process are also returned back to the users with the query results to inform them of the updated domain knowledge.

To summarize, ontology propagation has mainly been considered in the past for two different types of ontology related artifacts: related ontologies ([SKKM03], [Oli00], [Sto04]) and instances ([SSH02b], [MPSd06]). For the former, the problem has been addressed for different cases, i.e., the propagation of changes to local related ontologies and distributed ontology replicas ([Sto04]), the propagation of changes from a shared central ontology to distributed (modified) copies ([Oli00]), or the propagation of changes from distributed ontology components to a central ontology copy ([SKKM03]). Finally, the propagation of changes to related applications is discussed in [LADS06].

Change Propagation in related fields

The propagation of changes allows keeping the coherence between related objects (changes in a dependent object have to be propagated to depending objects in order to keep them coherent). For example, an important issue in the dissemination of time-varying web data such as sports scores and stock prices is the maintenance of temporal coherence. In [DKP+01] there is an evaluation of push and pull algorithms to maintain such coherency. In the case of servers adhering to the HTTP protocol, clients need to frequently pull the data based on the dynamics of the data and on the user's coherency requirements. In contrast, servers that possess push capability maintain state information pertaining to clients and push only those changes that are of interest to a user. These two canonical techniques have complementary properties with respect to the level of temporal coherency maintained, to communication overheads, to state space overheads, and to loss of coherence due to (server) failures. The paper shows how to combine push- and pull-based techniques to achieve the best features of both approaches. The combined technique tailors the dissemination of data from servers to clients based on (i) the capabilities and load at servers and proxies, and (ii) clients' coherency requirements.

The problem of change propagation has been thoroughly investigated for schema evolution ([PÖ97], [NR89], [PS87], [LH90], [RR97], [BKKK87], etc.). Although in data schema evolution the principal dependent artifacts are the instances representing the database population, the work can be reconsidered for the ontology domain (e.g., for updating ontology instances). In general, either objects are explicitly coerced to coincide with the new definition of the schema (updating the affected objects, changing their representation as dictated by the new schema), or a new version of the schema must be created leaving the old version intact. The approaches include the techniques of screening (also known as deferred propagation), conversion (also known as immediate propagation), and filtering to define when and how coercion takes place. There are also hybrid approaches that combine two or more of the above methods.

Another relevant work is presented in [Raj97]. The paper presents a model of change propagation during software maintenance and evolution. Change propagation is modeled as a sequence of snapshots, where each snapshot represents one particular moment in the process, with some software dependencies being consistent and others being inconsistent. A snapshot is changed into the next one by a change in one software entity and in the dependencies related to it. The formalism for this process is based on graph rewriting. The paper discusses two basic processes of change propagation: change-and-fix, and top-down propagation. The change-and-fix process is the most common process of software maintenance. It allows changes to propagate in all directions, and the first change can begin anywhere. However, many of its properties are unclear. The top-down approach is a more predictable process, where changes propagate only from top-down and the first change can begin just in the topmost class.

Another relevant work in the Grid field is presented in [CAM+07]. Here the authors discuss how to manage the evolution and change of metadata and knowledge

models in distributed contexts, and how to synchronize adequately the evolution of all these related entities by means of notification mechanisms. In particular, they propose to use a WS-Notification[26] mechanism within the context of the S-OGSA architecture[CAK+06] (Semantic-Open Grid Service Architecture), where a set of pre-defined topics are associated to the changes.

Finally, we would like to introduce another related technology that can be useful for our issue of propagating changes. RSS[27] (which, in its latest format, stands for "Really Simple Syndication") is a family of web feed formats used for publishing frequently updated content such as blog entries, news headlines or podcasts. An RSS document (also known as a "feed", "web feed" or "channel") contains either a summary of content from an associated web site or the full text. RSS content can be read using software called a "feed reader" or an "aggregator." The user subscribes to a feed by entering the feed's link into the reader or by clicking an RSS icon in a browser that initiates the subscription process. The reader checks the user's subscribed feeds regularly for new content, downloading any updates found.

2.2.3 Tools for Change Management

Most of the existing tools supporting the management of changes provide this functionality as part of the collaborative ontology development support, such as Hozo and Collaborative Protégé, and they are described in section 2.3.2. Hence, in this section we only focus on those tools/systems that do not tackle additional collaborative aspects.

A number of tools supporting different aspects of ontology versioning have been reported in the literature. OntoView ([KFK+02], [Kle02], [Kle04]) is a web-based change management system for ontologies, inspired by the Concurrent Versioning System, CVS[Ber90]. It provides an interface to different versions of ontologies, maintaining not only the transformations between them, but also the conceptual relation between concepts in different versions. It uses several rules to find changes in ontologies (called transformation set) and visualizes them (and some of their possible consequences) in file representations. The user is able to specify the conceptual implication of the differences, which allows the interoperability of data described by the ontologies. The main functions of the system include comparing ontologies, reading changes and ontologies, identifying ontologies, making a basic analysis of the effects of changes, and exporting changes.

PROMPTdiff ([NKKM04]) is a Protégé plugin based on Ontoview. It compares two different versions of an ontology (similar also to the SemVersion system presented in [Völ06]) producing a structural *diff* between ontologies, i.e., mappings. Users can visualize the resulting *diff* and analyze the changes, that is, they can view concepts and groups of concepts that were added, deleted, and moved; these changes are distinguished by their appearance and have direct access to the

[26] www.oasis-open.org/committees/wsn/

[27] http://cyber.law.harvard.edu/rss/rss.html

additional information that characterizes such changes. The users can then act on the changes, accepting or rejecting them at three levels of granularity: (i) individual changes to property values and definitions; (ii) all changes in a definition of a class; and (iii) all changes in class definitions and class-hierarchy positions in a subtree rooted at a particular concept. In [Kle04] the author describes two extensions to the PROMPTdiff tool. The first extension uses the mappings produced by PROMPTdiff as a basis for producing a transformation set. The second extension is able to detect some composite changes and presents these to the user in a conveniently arranged way.

In [OK02] the authors propose an ontology middleware system for the management of knowledge bases (RDF repositories). In particular, they propose that such a middleware should provide (i) a repository providing the basic storage services in a scalable and reliable fashion (Sesame); (ii) a multi-protocol client access; (iii) support for plugable reasoning modules suitable for various domains and applications; and (iv) a Knowledge Control System (KCS). The KCS provides features for the versioning and tracking of changes of knowledge bases (such as CVS for software), an access control system and meta-information for knowledge bases (as described in 2.2.1.2). The design and implementation of the ontology middleware proposed is just an extension of the Sesame architecture [BKvH02] that already covers many of the desired features. For the tracking of changes, the implementation considers that for each repository, there is an update counter (UC), i.e., an integer variable that increases each time when the repository is updated; in the basic case this means that when a statement is added to or deleted from the repository. Each separate value of the UC is called UID (UC update identifier); then, for each statement in the repository the UID when it was added and the UID when it was removed will be known and these two values determine the ”lifetime” of the statement. Additionally, each state of the repository is identified by the corresponding UID. Since the authors propose that versions be just labeled states, each version can be uniquely identified by the UID.

OntoAnalyzer ([RP05]) is another tool that consists of three components: a meta-model for change descriptions, a change tracer and a change history manager. The change tracer is able to track complex changes in addition to elementary ones. In particular, it collects changes from logs in a standardized manner and formalizes them using a proposed model. To standardize the collection of changes (from different ontology editors), a set of metadata is used to describe changes according to the format of the change tracer. Then to formalize those descriptions, they introduce the OntologyChange (OC) language, which consists of a set of constructs that are used to declare specific changes in OWL ontologies, such as oc:Add (Class, Property), oc:Delete, oc:Modify, oc:Move, oc:Merge, oc:Split, etc. After all changes are formalized, the change tracer appends them directly to the next ontology version (Vn+1) by means of the *owl:versionInfo* statement. The new ontology Vn+1 contains, besides the underlying domain conceptualization, all the information regarding ontology evolution. Finally, the change history manager allows the identification and the analysis of the changes that lead to a

specific ontology version. This module reads the OC formalized changes declared within the ontology version and generates both an interface for users and a file for software agents.

In [Sto04] the author presents an ontology evolution system developed within the Karlsruhe Ontology and Semantic Web framework (KAON[28]). The KAON ontology evolution support is implemented at the API level and includes the following functionalities:

- The *evolution logging* is responsible for keeping track of the ontology changes in an evolution log.
- *Change reversibility* enables undoing and redoing changes made in an ontology.
- The *evolution strategy* is responsible for ensuring that all changes applied to the ontology leave the ontology in a consistent state and for preventing illegal changes. Also, the evolution strategy allows the user to customize the evolution process.
- The *evolution graph* enables ontology engineers to enhance a set of changes with their own changes and to resolve them.
- *Ontology inclusion* facilities, together with the *dependent evolution*, are responsible for managing multiple ontologies within one node, i.e., change propagation.
- *Ontology replication* facilities, together with the *distributed evolution*, are responsible for enabling the reuse and management of distributed ontologies, i.e., change propagation.
- *Change discovery* includes the means for discovering problems in an ontology and for making recommendation for their resolution.
- *Usage logging* is responsible for keeping track of the end-users' interactions with ontology-based applications in order to adapt ontologies to the users' needs.

The evolution support also includes a *change model* that is based on events rather than change methods, and *virtual ontologies* for (i) preventing illegal changes and (ii) keeping consistency.

Finally, the authors in [LADS06] report the implementation of a prototype system that implements the approach described in 2.2.2.2 to control the impact of ontology changes on dependent applications/services. They define a Log Ontology

[28] `http://kaon.semanticweb.org`

that captures and manages the change information between two versions of ontologies and develop a Middle Analyzing Layer that analyzes the end-users' queries, amends the entities within the queries accordingly related to the change information captured in the Log Ontology, and informs the end-user of the changes and actions taken.

To sum up, we can say that most of the existing tools for the management of ontology changes, which do not consider collaborative aspects, focus on the tracking of changes and on the management of different ontology versions (e.g., OntoView ([KFK+02]), PROMPTdiff ([NKKM04]), [OK02] and OntoAnalyzer ([RP05])). On the other hand, KAON ([Sto04]) implements a set of functionalities to support many of the tasks of the ontology evolution process. Finally, [LADS06] is a prototype that supports the propagation of changes to related applications/services.

2.2.4 Limitations on Change Management

In the previous sections we have presented a comprehensive state of the art of all the activities related to the management of changes (metadata). As we have noted, a large number of those activities (e.g., change propagation, change representation and versioning approaches) have been thoroughly investigated in many other computer science fields; however, for our domain (ontologies), they are still under active research. In the rest of this section we will briefly discuss the main limitations and challenges encountered in this area for the core topics of this thesis, namely, change representation, change propagation and tools support.

Regarding the *representation of changes* (section 2.2.2.1), we can summarize that much of the current work focuses on devising taxonomies of change operations that classify them as elementary (atomic) or composite, and some approaches also consider complex type of changes. Although the existing approaches introduce the elementary (atomic) changes proposed as operations that cannot be subdivided into smaller operations, **they consider changes at the entity level**, that is, concepts, properties and individuals (e.g., [Sto04], [Kle04], [KFK+02], [NK03], [Lia06] and [NCLM06]), and in some cases these elementary changes are not even minimal, for example, they include *modify* operations (e.g., [Kle04]). Furthermore, **existing approaches are dependent on the underlying ontology model**, that is, they are based on proprietary models (e.g., KAON [Sto04]) or they are developed for specific languages (e.g., OWL [Kle04]), consequently, they have different sets of elementary (atomic) changes. However, even though the different ontology languages have their own semantics and support a particular set of ontology elements and operations, all of them are expected to support some fundamental operations, such as adding a class (or a concept depending on the underlying knowledge representation paradigm). Hence, **one key challenge would be to provide a language-independent approach for the representation of changes** that could be reused and specialized for the different ontology languages. Furthermore, **another challenge would be to consider a further lower**

level for the type of ontology change that is the actual "minimal" operation to be performed in an ontology model (e.g., add/remove OWL axioms in an OWL ontology). This lower level would enable to establish the link between the level at which users typically work – ontology elements – (e.g., add a class with properties) and the level at which the systems typically work – axioms, triples, etc. – (e.g., add class axiom, add properties axioms and add range constraints axioms).

With respect to the *propagation of ontology changes*, still much can be learned from other related fields. Although this issue has been addressed in fields such as databases, the state of the art shows that only a few works tackle this issue in the ontology field and in most of them the issue is just partially addressed. In particular, it has been investigated the propagation of changes to dependent ontologies (e.g., [Sto04], [Oli00]) and [SKKM03]), to ontology individuals (e.g., [SSH02a] and [MPSd06]), and, in one specific work, to dependent applications ([LADS06]). For the case in which the dependent ontologies are distributed, the existing approaches only address one of the following cases:

- An ontology is locally edited by a single user; changes to that ontology are propagated to physically distributed replicas of the ontology (e.g., [Sto04]).
- A shared-central copy of an ontology is shared by several users. Each user has a local copy of this ontology (which may even be modified), that may (or may not) be updated (synchronized) whenever the shared-central copy is changed in a centralized manner by a single user (e.g., [Oli00]).
- An ontology is divided into several (distributed) component ontologies related to each other. Each component ontology is then treated as a normal ontology, this ontology is locally edited by a single user and its changes are propagated to related ontology components; such a process is carried out in a centralized manner when the user publishes the component ontology in a central server (e.g., [SKKM03]).

Hence, in all cases, **the existing approaches consider a central (main) copy of the ontology** that is either replicated or divided into several component ontologies. Moreover, in most cases, **changes are propagated only in one direction: from the main copy to its replicas**. The only exception occurs when the ontology is divided into several component ontologies, but in this case **the management of changes is also centralized**.

Finally, existing *tools* support different aspects of the change management. Many of them are focused on the management of ontology versions (e.g., [Kle02], [NKKM04] and [RP05]) rather than on supporting the tasks of the ontology evolution process that are our main interest. Nevertheless, some tools address evolution aspects, such as the effect of changes on dependent applications (e.g., [LADS06]) or, in particular for this thesis, the tracking of changes and their formal representation and/or the propagation task (e.g., [Sto04], [OK02] and [RP05]). However,

in those cases, **the management of ontologies and their changes is either centralized or the changes are propagated from a main copy of the ontology to its distributed non-editable replicas**.

Furthermore, existing technologies, like CVS[29], were found in the past to be inappropriate or insufficient when dealing with ontologies (e.g., [HSV04] and [KVH05a]) as they were not designed to support complex structures. For example, CVS works on the syntactical not on the conceptual level, that is, it is not capable of versioning objects; therefore, CVS (with its underlying diff operation) can only compare versions at line-level or at character-level, highlighting the lines that textually differ in two versions. What we actually need is a comparison of ontologies at a structural level that shows which definitions of ontological concepts or relationships have changed. Nevertheless, CVS has served as inspiration in the past for ontology versioning systems, such as [Kle04].

2.3 Collaborative Ontology Development

A range of ontology engineering methods and methodologies have been proposed in the literature (e.g., Gruninger [GF95], METHONTOLOGY [FLGPPSPS99] and On-To-Knowledge [SSSS01]). All of them have taken a step forward in the process of transforming the art of building single ontologies into an engineering activity. However, as has already been analyzed in [CFLGP03], none of those methods and methodologies have taken into account the collaborative and distributed construction of ontologies. A comparative and detailed study of the earlier methods and methodologies can be found at [FL99].

Nevertheless, the problem of collaborative ontology development has been partially addressed in the literature with methodological and technological results, which are not necessarily aligned. Some of those works are inspired by or based on similar works in related areas (e.g., Co4 [Euz95]).

In the following sections, we present the most relevant approaches for the collaborative ontology development, followed by an overview of the tool support in this area.

2.3.1 Collaborative Ontology Development Approaches

In this section we focus on the existing works of ontology development, where some kind of reviewing process has been acknowledged.

An early work in a similar field described in [Euz95] is Co4[30]. This is a protocol for the collaborative and distributed construction of a repository (e.g., knowledge base, document, design description). The protocol manages the communication between collaborators in order to maintain the consistency of the repository and the agreement of all the collaborators on the content of the repository. It also

[29] `http://www.cvshome.org/`

[30] `http://co4.inrialpes.fr/#co4`

mimics the submission of articles to peer-reviewed journals (with the exception that each change must be accepted by all the participants). The protocol is independent from the nature of the repository and is based on a restricted set of message types expressed as a collection of speech acts, such as *achieve* (submit a proposal for inclusion into the consensual base), *ask-all* (ask for the acceptance of subscriber bases, call for comments), *accept* (accept the insertion of the piece of knowledge), etc. The communication between collaborators is described through a set of rules, but it is very restrictive, that is, it is unable to account gracefully for the concurrent submission of mutually contradictory proposals.

General approaches in this area focus on the characterization of the ontology development process in collaborative environments. DILIGENT ([PSST04], [Tem06]) is a robust approach that identifies several key roles (e.g., experts, with different and complementary skills) involved in building collaboratively the same ontology. It comprises the following five main steps:

- *Build.* The process starts by having domain experts, users, knowledge engineers and ontology engineers build an initial ontology.

- *Local adaptation.* Once the core ontology is available, users work with it and, in particular, adapt it to their local needs. By logging local adaptations, the control board collects the changes requested to the shared ontology.

- *Analysis.* The board analyzes the local ontologies and the requests and tries to identify similarities in the users' ontologies. Since not all of the changes introduced or requested by the users will be introduced to the shared core ontology, a crucial activity of the board is to decide which changes are going to be introduced in the next version of the shared ontology.

- *Revise.* The board should regularly revise the shared ontology, so that local ontologies do not diverge considerably from the shared ontology.

- *Local update.* Once a new version of the shared ontology is released, users can update their own local ontologies to use better the knowledge represented in the new version.

After the local update takes place, the iteration continues with local adaptation. During the next analysis step, the board will review which changes were actually accepted by the users. DILIGENT also considers the provision of arguments for the changes requested and design decisions in a semi-formal way (similar to [Sto04]).

DOGMA-MESS ([dMLM06], [LM07]) proposes a generic model for understanding the interorganizational ontology engineering process. Through this process the knowledge moves in an upward spiral starting at the individual level, moving up to the organizational level, and finally up to the interorganizational level. According to the authors, an interorganizational ontology (IOO) consists of various, related sub-ontologies. The engineering process starts with the creation of

an upper common ontology (UCO), which contains the conceptualizations and semantic constraints common to and accepted by a domain. Each participating organization contextualizes (through specialization, for instance) this ontology into its own Organizational Ontology (OO), thus resulting in a local interpretation of the commonly accepted knowledge. In a so called Lower Common Ontology (LCO), a new proposal for the next version of the IOO is produced, selecting and merging relevant material from the UCO and various OOs. The part of the LCO that is accepted by the community then forms the legitimate UCO for the next version of the IOO. Based on this approach, each version of the IOO construction consists of three stages: (i) creation of the templates (describing a common knowledge definition that is the most relevant to the common interest); (ii) definition of the organizational (template) specializations (divergence of definitions); and (iii) definition of the common specializations (convergence of definitions).

In [NCLM06] the authors identified several dimensions that we can use to classify scenarios for collaborative ontology evolution:

- *Synchronous vs asynchronous editing*. The synchronous mode allows collaborators to work on the same version of an ontology that resides on a server accessible to all members of the team, while in asynchronous mode, collaborators check out an ontology or a part of it and edit it off-line. Then they merge their changes into the common version.

- *Continuous editing vs periodic archiving*. In continuous editing, the only version of an ontology that exists is the latest one, and any rollback is performed using an undo operation. In contrast, versions can be archived periodically, and one can roll back to any of the archived versions.

- *Curation vs no curation*. This dimension differentiates scenarios where after editors perform a set of edits a separate curation step takes place or not. In the curation step, one or more curators examine all changes and accept/reject them, resolving any conflicts.

- *Monitored vs non-monitored*. This dimension differentiates whether the tools record the changes and, possibly, the metadata about these changes (monitored edition) or not (non-monitored edition).

Further, this work introduces some functional requirements for collaborative ontology development, including change annotations, change history for a concept, representation of changes, access privileges, query old versions using the vocabulary of the new version, identification of conflicts, negotiation mechanism to resolve conflicts, roll-back feature, accept and reject changes, specialized views of changes, and comparison and description of versions.

Some other works in this area focus on specific aspects or address specific scenarios. In [SKKM02] and [SKKM03], the authors describe an approach where an ontology is divided into several component ontologies, developed by different

developers in a distributed environment where the target ontology is obtained by compiling the component ontologies. So, as was described in section 2.2.2.2, the authors consider two kind of dependencies between component ontologies (is-a relation and referring-to relation), the influence of a change of one ontology to others through its dependencies and they design a function to suggest a few candidate modifications of the influenced ontology for keeping the consistency. In particular, they propose two possible countermeasures to accept the change and three to reject it. Each component ontology can be developed locally only by a single user; the management of changes is done in a centralized manner, that is, users have to publish their ontologies on a shared space of a server computer.

The Finland national ontology library development framework ONKI ([KVH05a] [KVH05b]). features change management and versioning of ontologies as well as a browser component that provides services for ontology search and utilization as web services. The approach includes the description of ontology changes as instances of an RDF change ontology that are stored as metadata of the ontology development version when editing an ontology. They also introduce the notion of a proxy that is used to import concepts from related ontologies. It creates a copy of the borrowed concepts and keeps track of their origin.

Wiki-based approaches, where the main paradigm is simplicity, have also been reported in the literature. In [BH04] the authors propose using the wiki technology to support collaborative ontology development. The strength of this approach is its logical foundation. It is based on OSHOQP(D), a modular ontology representation language with preference partial order on axioms. The authors here discuss how to integrate and reconcile multiple independently developed, semantically heterogeneous, and very likely inconsistent ontology modules, where an ontology module is composed of one or more wiki pages. A similar informal approach is introduced in [HBS06]. The authors propose to use standard wiki technology as an ontology development environment. In particular, they propose to use a wiki as a mechanism so that a community can create a URI for any concept needed, describe the concept using natural language, refine and modify the definition, and link this approach with the wealth of concepts already defined in Wikipedia.

Recently, the Protégé team have started to get interested in workflows for collaborative ontology development. In [SNTM08] the authors analyze for some projects the way the members contribute, the different roles they play and the mechanisms they use to carry out discussions and to achieve consensus. They introduce an ontology to represent different aspects of workflows for the collaborative ontology development and they show how to use the ontology to represent formally two different collaborative workflows. Furthermore, the authors present in [TNTM08] an updated list of requirements for supporting collaborative ontology development, as they did in [NCLM06] (see above). However, this time they focus more on real collaborative aspects including (i) integration of discussions and annotations in ontology development, (ii) support for various

levels of expressiveness, (iii) user management and provenance of information, (iv) scalability, reliability, and robustness, (v) access control, (vi) workflow support, and (vii) synchronous and asynchronous access to shared ontologies.

A summary of the features of the works presented appears at the end of this section in Table 2.4.

2.3.2 Tools for Collaborative Ontology Development

The collaborative creation of knowledge is supported by a number of tools for different areas and in different degrees. As was analyzed in [NCA08], constructing knowledge collaboratively is an issue often tackled in the past for unstructured knowledge, that is, no semantic links between artifacts exist and usually the artifacts are not related to one another in any structured form (e.g., wikis and folksonomies). Nevertheless, in the last years, a new generation of tools are starting to support the collaborative construction of structured knowledge, that is, artifacts usually contain explicit definitions of, and links between their components, often with well-defined semantics (e.g., ontologies and database schemas). These new generation of tools (many of them still in the early stages of development) range from extensions of wikis that support semantic links between wiki pages (for example, Semantic MediaWiki[31], BOWiki[32] and Platypus Wiki[33]), to tools that enable the tagging of bookmarks collaboratively and the organization of tags in some semantic structure (BibSonomy[34] and SOBOLEO[35], for example), to ontology editors that support at least some of the aspects of distributed and collaborative development of ontologies (Collaborative Protégé[36] and OntoWiki[37], among others).

For instance BibSonomy[HJSS06] is a web-based system for sharing bookmarks and lists of literature. Users discover a bookmark or a publication on the web and store it on their server. The post can be associated with tags describing topics, names, places and events in order to retrieve it more easily. This system also allows users to create relations between tags. The main benefit is that users share their tags and tagged resources with other Isonomous users. Another example is SOBOLEO[ZB07] that is a web-based tool for the collaborative engineering of SKOS ontologies[38] and for the annotation of web resources. This tool enables users to create collaboratively a taxonomy, to annotate web resources and to use the tool as background knowledge during the search. It consists of four main parts: (i) a collaborative real time editor for changing the taxonomy, (ii) a tool for annotating web

[31] http://meta.wikimedia.org/wiki/SemanticMediaWiki

[32] http://onto.eva.mpg.de/bowiki/

[33] http://platypuswiki.sourceforge.net/

[34] http://www.bibsonomy.org/

[35] http://soboleo.fzi.de:8080/webPortal/

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

[37] http://3ba.se

[38] SKOS provides a model for expressing the basic structure and content of concept schemes such as taxonomies, thesauri, classification schemes, etc.

resources, (iii) a semantic search engine for the annotated web resources, and (iv) a taxonomy browser for navigating through the taxonomy and the content of the web resources index. A related tool is DBin [TMN06], which is a peer-to-peer application that enables user groups to create knowledge bases collaboratively. Similar to a filesharing client, DBin allows sharing and receiving semantically-structured information. Also, as happens in a newsgroup client, bits of information (called "annotations") are collected and inserted by entering topic rooms. To enjoy the topic rooms better, DBin allows domain experts to create domain- or task-specific user interfaces (known as brainlets) from a collection of components, such as ontology or instance editors, discussion forums, specific views, and other plugins. Users edit ontologies – to be used for annotations in the domain – directly on the server and they can see changes made by other users. The system keeps and makes readily available the authorship of any piece of information, such as annotations.

Some additional tools can be found in the literature (e.g., the implementation[39] of the Co4 protocol (see above)), but since our main concern is the development of ontologies, in the rest of this section we focus only on existing tools and prototypes that provide at least some support for the development of ontologies collaboratively. After analyzing those tools, we can differentiate three different generations: (i) web-based tools which provide different users with the possibility of working on the same ontology remotely, for example, Ontolingua and WebODE; (ii) tools with some initial/isolated collaborative functionality such as chats, discussions, etc., like WebOnto; and (iii) tools with a set of integrated collaborative functionalities (e.g., Hozo and Collaborative Protégé). Next, we briefly introduce some of the tools of the first two generations, and then we provide a more detailed overview of the more complete tools.

The Ontolingua Server [FFR96] was the first ontology tool created. It appeared at the beginning of the 1990s, and was built to address the development of Ontolingua ontologies with a form-based web application. Ontolingua Server consists of several modules such as the ontology editor (main module), Webster, an equation solver, an OKBC (Open Knowledge Based Connectivity) server, etc. Additionally, the ontology editor provides translators to several languages, for example, Loom, Prolog and CLIPS. Remote editors can browse and edit ontologies, and remote or local applications can access any of the ontologies in the ontology library with the OKBC protocol [CFF+98].

WebODE [ACFLGP01], the successor of ODE (Ontology Design Environment) [BFLGPGP98], is an ontology-engineering suite created with an extensible architecture. WebODE is used as a Web server with a Web interface; it employs a relational database to store ontologies. The core of this environment is the ontology access service, which is used by all the services and applications plugged into the server, especially by the WebODE's Ontology Editor. It also includes

[39] Seehttp://co4.inrialpes.fr/docs/co4-1.0a/co4-Appendix.htmltoseehowtogetCo4

several services such as importation/exportation for ontology language (e.g., XML, RDF(S), OIL, DAML+OIL and FLogic), axiom edition with WebODE Axiom Builder (WAB) [CFLGPV02], ontology documentation, ontology evaluation and ontology merge. Finally, WebODE gives support to most of the activities involved in the ontology development process proposed by METHONTOLOGY, although this support does not prevent it from being used with other methodologies or without following any methodology.

Tadzebao and WebOnto [Dom98] are two complementary systems enabling the edition of ontologies with support for discussions during the collaborative development. Tadzebao permits knowledge engineers to hold synchronous and asynchronous discussions about ontologies, whereas WebOnto supports the collaborative browsing, creation and editing of OCML[40] ontologies. The architecture is composed of a central server (Web server) and clients written in Java.

The new generation tools for the collaborative ontology development include a more integrated set of functionalities that address in different degree or different scope the collaborative aspects. In [PSST04] the authors introduce a tool support for the DILIGENT process. It is an implementation of the Edit component of the SWAP environment [EHS+03], thus it works on the information stored in the local node repository, and it is realized as a plugin of the ontology engineering environment OntoEdit [SAS03]. The tool supports users to (i) *build* an ontology (using OntoEdit) and (ii) *adapt locally* the ontology by creating concepts, relations and instances from folder structures or by using information from other peers. Additionally, the tool supports the board to (iii) *analyze* the activities which have taken place by gathering the ontologies from all participating peers on one central peer. The main task of the board is to incorporate the change requests into the core ontology and to identify common usage patterns. Then the board (iv) *revises* the ontology by integrating the change requests and the common usage patterns recognized. Finally, for the (v) *Local update*, the changes to the core ontology are propagated to all peers afterwards.

DOGMA-MESS is supported by a web server ([dMLM06]). The core of the server is a Java server that interacts with the DOGMA Studio server, which in turn implements the standard DOGMA ontology engineering and analysis engine. Special converters translate standard DOGMA representations (templates) to and from a simple native DOGMA-CG format. Concept type hierarchies can be imported as indented text files. Type hierarchies and templates, like organizational specializations, can also be edited through the DOGMA-MESS web interface. The main operation used is the projection operation for checking whether organizational specializations conform to their templates.

Hozo ([SKKM02], [SKKM03]) proposes to divide an ontology into several component ontologies, which are developed by different developers in a distributed

[40] Operational Conceptual Modeling Language[DMC99]

environment where the target ontology is obtained by compiling the component ontologies (each developed locally by a single user). Hozo is composed of four components: (i) the ontology manager, which provides a global view of the target ontology and gives information of dependencies between component ontologies; (ii) the ontology editor, which provides users with a graphical interface for browsing and modifying ontologies; (iii) Onto-Studio, which helps users to design ontologies from technical documents; and (iv) the ontology server, which manages the ontologies and models built (instances of the ontology) keeping track of the developer. In this environment, the development of the ontology consists of the following steps: (i) a developer logins Hozo and runs Ontology Manager; (ii) the developer selects the ontology to edit and open it with the ontology editor (checkout); (iii) the developer edits the ontology; and finally (iv) the developer updates his ontology on the ontology server and publishes it (commit). During the checkout and edition of the ontology, the dependencies can be checked to determine if there exist any inconsistency, in which case user can select countermeasures to cope with the change. To manage the dependencies, the system keeps the information about the name and version of the ontology being edited and about all the ontologies it depends on, and it also keeps a copy of the definition of concepts in other ontologies that the ontology depends on (which is used to check the consistency of dependency by comparing the copy with the real definition).

In [KVH05b] the authors reports a prototype system for the ONKI framework. It consists of three core components: the development repository (ONKI GORepository) for the ontologies being edited; the public ontology library containing the set of published interrelated ontologies (ONKI Library); and the browsing service (ONKI Browser) for illustrating, finding, and importing concepts from the ONKI system ontologies.

In [BH04] the authors introduce Wiki@nt, an ontology building environment that supports the development of OSHOQP(D) ontologies collaboratively. OSHOQP(D) provides an expressive language to build ontology from autonomous, distributed, and possibly inconsistent ontology modules (see 2.3.1). An ontology module (package) is composed of one or more wiki pages (block). Multiple agents (for example, users) can edit the same content with version control and transaction management. The ontologies are loaded into or uploaded from a set of wiki pages and managed by an ontology repository. They rely on a wiki engine (JSPWiki [41]) for storing wiki pages, controlling the versions, and managing transactions, among others. To resolve inconsistencies among modules when integrating them, the authors assume that each package is consistent and propose to use a partial order specified at the package level. For the transaction management, besides relying on the wiki engine, Wiki@nt denies the write-access of agents to a page (and related pages) if this page is locked by some other agents. A similar tool support presented in [ADR06] is OntoWiki. The main goal of OntoWiki is to simplify the presentation and acquisition of instance data from and for end-users. OntoWiki enables

[41] `http://www.jspwiki.org`

the visual presentation of a knowledge base (ontology instances) as an information map, with different views on instance data. Ontowiki targets social collaboration aspects by keeping track of the changes, allowing commenting and discussing every single part of a knowledge base, enabling to rate and measure the popularity of content, logging the activity of users and performing semantic searches. Ontowiki is based technologically on Powl[42] since it provides an alternative user interface to the schema editor integrated in Powl. Note however, that Ontowiki design is independent and complementary to conventional Wiki technologies. Instead of mixing text editing with knowledge engineering, OntoWiki applies the Wiki paradigm of *making it easy to correct mistakes, rather than making it hard to make them* to collaborative knowledge engineering.

In [NCLM06] the authors propose to support different ontology evolution scenarios using Protégé plugins. In particular, they present the Change and Annotation Ontology (CHAO) described in the previous section (see 2.2.2.1) and two Protégé plugins: the Change-management plugin that provides access to a list of changes (instances of CHAO) and enables users to add annotations to changes, and the PROMPT plugin (also introduced in [NKKM04]) that provides comparisons between two versions of an ontology, allowing examining the list of users who performed changes and accepting and rejecting changes. Finally, the authors introduce the client/server mode in Protégé for synchronous editing by multiple users.

Based on this work, the Protégé team introduces in [TN07] an extension of the existing Protégé system that supports collaborative ontology editing (Collaborative Protégé). Besides the common ontology editing operations supported in Protégé, the extension enables the annotation of both ontology components and ontology changes. This functionality allows the users to comment and discuss about the content and changes of the ontology that they develop in common. Additionally, the Collaborative Protégé supports the searching and filtering of user annotations based on simple or complex criteria. Finally, the authors propose two types of voting mechanisms that can be used for voting change proposals (a *5-star* voting or a *Agree/Disagree* type of voting). However, the current prototype does not support a workflow for the voting mechanism.

Recently, the Protégé team took an interest in workflows for collaborative ontology development. In [TNTM08] the authors describe the architecture of Collaborative Protégé (already introduced in [TN07]). As mentioned before, Collaborative Protégé enables users to develop an ontology collaboratively and to hold discussions, to chat and to annotate ontology components and changes. The paper introduces the ontology modules (some of them under implementation) that drive the collaborative development process: (i) the Roles module to describe users, roles, operations and policies that apply to a certain ontology; (ii) the Workflows module provides a formal language for describing workflows for collaborative ontology development; (iii) the Ontology Components module provides a metalanguage for describing representational entities in different ontology languages;

[42] `http://powl.sf.net`

(iv) the Annotations module represents the different types of annotations that users make; and (v) the Changes module contains classes representing different types of changes that can occur in an ontology. Currently, however, there is a simple support for different users and roles (read/write permissions), and the workflow support is not implemented. Actually, in a parallel paper ([STNM08]) the authors discuss how they are implementing the collaborative workflow support in Protégé. The main steps in this support include the following: (i) the *description* step is the workflow ontology described in the previous section, (ii) *instantiation* is the step where developers specify which roles users have, what activities users with each role can perform, the order of activities, and the conditions that trigger new actions or new workflows, (iii) for the tool-generation and execution step, the instances of the previous step should be parsed to create an executable workflow structure to be run in a workflow execution engine that interacts with the Protégé environment and controls the flow of operations in which the distributed clients participate. They rely on JBoss Process Virtual Machine (PVM) as their workflow engine. Finally, users should then implement the generated activities. As we noted in Chapter 1, Protégé efforts to address the collaborative process during ontology development, which started last year ([TNTM08], [STNM08]), are contemporary and similar to the work we present in this thesis to address this issue, and that we reported originally as a public technical report in 2007.

Table 2.3 summarizes the temporal evolution of the aforementioned tools.

A summary of the features of the presented tools is presented at the end of this section in Table 2.4.

Table 2.3: Temporal Evolution of Collaborative Ontology Development Tools

Tool	**Year**
Ontolingua Server	1996
Tadzebao and WebOnto	1998
WebODE	2001
Hozo	2002
DILIGENT Tool	2004
Wiki@nt	2004
ONKI	2005
DOGMA-MESS tool	2006
OntoWiki	2006
Client/Server Mode in Protégé with change plugins	2006
Collaborative Protégé	2007

2.3.3 Limitations on Collaborative Ontology Development

In the previous sections we have presented existing approaches and tools supporting the collaborative ontology development. We have noted that the construction of knowledge collaboratively is nothing new for ontologies and we have distinguished different generations of tools supporting this process.

In the remainder of this section we summarize some of the issues related to the collaborative aspects for the ontology editing that have been just partially addressed (or not at all).

Although [Tem06] and [dMLM06] consider the collaborative development of ontologies in a distributed setting, **it is not clear how change requests are represented**, **nor is there any explicit tracking of the change operations** in the shared ontology that would be useful for local users when identifying the approved changes or comparing the shared ontology with the local copy. Besides, there is no history of the rejected changes. In both cases, **there is a central copy** of the ontology developed collaboratively, and **distributed users work with a local adaptation/specialization**, resulting in different versions of the ontology which might hamper the interoperability. Those local copies are eventually analyzed/merged **in a centralized manner** in order to update the main ontology, and then local users will eventually update (or not) their local copy, i.e., there is no automatic synchronization mechanism.

In [KVH05a] they claim the use of an RDF change ontology for the representation of changes; however, **it is not clear the kind of changes they support or what kind of information is tracked**; besides, the authors focus only on changes at the concept and ontology level. Additionally, the approach proposed **does not consider any kind of reviewing process** for the changes proposed nor does it consider how users interact in the process depending on their role. Finally, **the authors use a centralized approach** in which the ontologies and the change information are stored in a centralized server.

Even though in [SKKM03] the authors consider a distributed environment where users collaborate in the development of the final target ontology, **each component ontology can be developed locally by a single user**. Moreover, **the management of changes is done in a centralized manner**, i.e., users have to publish their ontologies on a shared space of a server computer. These authors consider a **limited set of change operations** restricted to concepts and according to the two dependencies they address. Finally, the approach they propose **does not take into account any kind of reviewing process** for the changes proposed nor they consider how users interact in the process depending on their role.

Although the wiki approaches ([BH04], [ADR06] [HBS06]) provide some informal (semi-formal) support for the collaborative ontology development, they clearly **lack** some important features, such as the **explicit representation of changes** that could be used to compare different version of ontologies, or the propagation of the changes since **the management of ontologies and changes is centralized**. These approaches **do not support a formal approach for a review**

process of the changes proposed, **nor do they consider how users, depending on their role, interact in the process**.

Recently, the Protégé team has made some progresses in collaborative support, but many features are still under development. In their early work in [NCLM06] and [TN07] they took into account the reviewing of changes (e.g., acceptance/rejection of changes), but **it was not clear what kind of roles (and related permissions) were considered, how those actions were traced, or what the process flow for reviewing was**. In their latest works, however, they have introduced the use of workflows ([SNTM08], [TNTM08] and [STNM08]), which they are currently implementing to represent the different states and actions that can take place during the collaborative ontology development. Similarly, those workflows allows to represent how users interact in the process depending on their role. Still, in the scenarios addressed, **the management of ontologies and related changes is centralized**. Therefore, **all actions of the envisioned workflow take place in a central copy of the ontology**. It is worth noting that the interest of the Protégé team in the use of workflows demonstrates that they also think this issue is an important aspect for supporting collaborative ontology development, as we stated earlier in the public deliverable [PHWd07], and that it is one of the hypotheses of this thesis.

Table 2.4 summarizes the analysis of the existing approaches and tools.

2.4 Summary

In this chapter, we have presented an exhaustive analysis of the state of the art of the topics dealt with in this thesis, and we have discussed their limitations.

In this section we provide an overall summary of the open research problems identified; we have focussed on those problems addressed in this thesis to provide a solution for the management of changes in distributed environments that supports collaborative ontology development in an organizational setting.

In such a collaborative scenario, where an ontology is being modified by many different actors who may be distributed and who can play different roles, change management is central. The first step towards an efficient management of changes is their identification. At a higher-level, **ontology metadata** is a key component to determine whether an ontology has changed or not, or to provide a general overview of how it has changed. However, previous efforts regarding the definition of an ontology metadata model have some limitations. For instance, most of them consist only of a list of property-value pairs, or they are implicitly defined within existing systems and repositories; consequently, there is still a lack of a standard or community-accepted ontology metadata vocabulary.

At a lower-level, **ontology change management** approaches deal with the representation of the required information to determine which were the specific changes applied to an ontology. However, existing change representation models are dependent on the underlying ontology model and they consider changes at the

Table 2.4: Summary of collaborative ontology development approaches and tools (— = Not Available; * = Not Implemented)

Feature	DILIGENT	DOGMA-MESS	ONKI	OntoWiki	Wiki@nt	Hozo	Collaborative Protégé
Formal change representation	—	—-	X (Limited)	—-	—-	—	X
Tracking of changes	X (Limited)	—-	X	X	X (Limited)	X (Limited)	X
Change history	—	—-	X (Limited)	X	—-	—	X
Ontology copies	One shared and multiple editable local adaptations	One core and multiple adaptations at different levels	One	One	One divided into several ontology modules	One divided into several editable component ontologies	One
Synchronization of ontology copies	Manual and possible not synchronized	Manual and possible not synchronized	—-	—-	—-	—	—-
Formal workflow	X (fixed)*	X (fixed)*	—	—	—-	—	X*
Change propagation	Shared copy to local adaptations, limited adaptation to shared	Main copy to local adaptations, limited adaptation to main	To dependent ontologies	—-	—-	Between all component ontologies	—
User Roles	X*	X*	—	—-	—-	—	X*
(C)entralized / (D)ecentralized	D although there is a main centralized copy	D although there is a main centralized copy	C	C	C although ontology modules are distributed	C although component ontologies are distributed	C
Tracking of Workflow Actions	—	—	—	—-	—-	—	—
Comments on ontology components	—	—	—	X	—-	—	X
Other collaboration features	—	—	—	discussions, ratings	—-	—	discussions, chat, ratings

entity level, i.e., concepts, properties and individuals.

Furthermore, the management of ontology changes involves several other activities, some of them particularly important to support collaborative ontology development in distributed environments. For instance, methods and strategies for capturing, maintaining and storing changes are necessary to support collaborative scenarios. Similarly, strategies for the propagation of changes are crucial to support a distributed control of changes. However, the existing approaches consider only the propagation to related ontologies, individuals and, in less detail, to dependent applications. Moreover, for the propagation of changes to related ontologies, the existing approaches consider only a central (main) copy of the ontology that is either replicated or divided into several component ontologies and, in general, changes are propagated only in one direction: from the main copy to its replicas. In all cases, the management of changes is performed in a centralized manner. Similarly, the propagation of changes to ontology related metadata has not been considered.

Finally, different methods and strategies should be taken into account during **collaborative ontology development**. On the one hand, typically within an organization, a team of ontology editors follow a well defined process for the coordination of change proposals. However, the existing solutions do not deal with this process (except from a recent conceptual work presented in [SNTM08]), and none of them provides any technological support for this task.

On the other hand, as there can be different collaborative scenarios (e.g., distributed editors working with a central copy of the ontology or with local copies of the same ontology), it is necessary to have strategies in place to support them. However, the existing solutions support only a centralized management of the ontology and its related changes. That is, (i) a shared/main copy of the ontology that is adapted/specialized by distributed users; (ii) a main copy of the ontology that is divided in sub-ontologies, each of them modified by distributed users; (iii) there is only a central copy of the ontology that all distributed users can modify. In every case, changes are applied and managed in the central copy of the ontology. In many of them not even a formal management of ontology changes (e.g., explicit representation and propagation) exists.

Even more important, there is no integrated approach (and technological support) that addresses all the previous issues.

Chapter 3

THESIS OBJECTIVES AND CONTRIBUTIONS

The scope of this PhD Thesis is the management of ontology changes in distributed and collaborative environments. The main contributions are the following:

i) The definition of a model for the representation of ontology metadata that can give support to this task.

ii) The development of models, methods and strategies for the management of ontology changes in distributed environments.

iii) The development of models and strategies to support collaborative ontology development.

As mentioned in the introduction, it should be noted that when we refer to ontology development, we refer particularly to the activities regarding the implementation and maintenance of the ontology (based on the terminology of existing ontology engineering methodologies (e.g., METHONTOLOGY [FLGPPSPS99], On-To-Knowledge [SSSS01] and NeOn Methodology [SFGP08]).

In the scope of this thesis, the ontology metadata model is a core component that allows identifying whether an ontology has changed (its metadata changed) and provides the basis for the other models proposed in this thesis, i.e., change representation model and collaborative workflow model. Using the models, methods and strategies of the second contribution, we can identify the particular changes applied (e.g., the type of changes and the chronological order) and propagate them to the required components in a distributed and collaborative environment. Finally, the models and strategies of the third contribution allow identifying how the changes were applied (e.g., the states of the changes and the policy of curation activities) and they also allow controlling the collaborative process during ontology development. Our contributions also include the technological support that implements the models, methods and strategies here proposed.

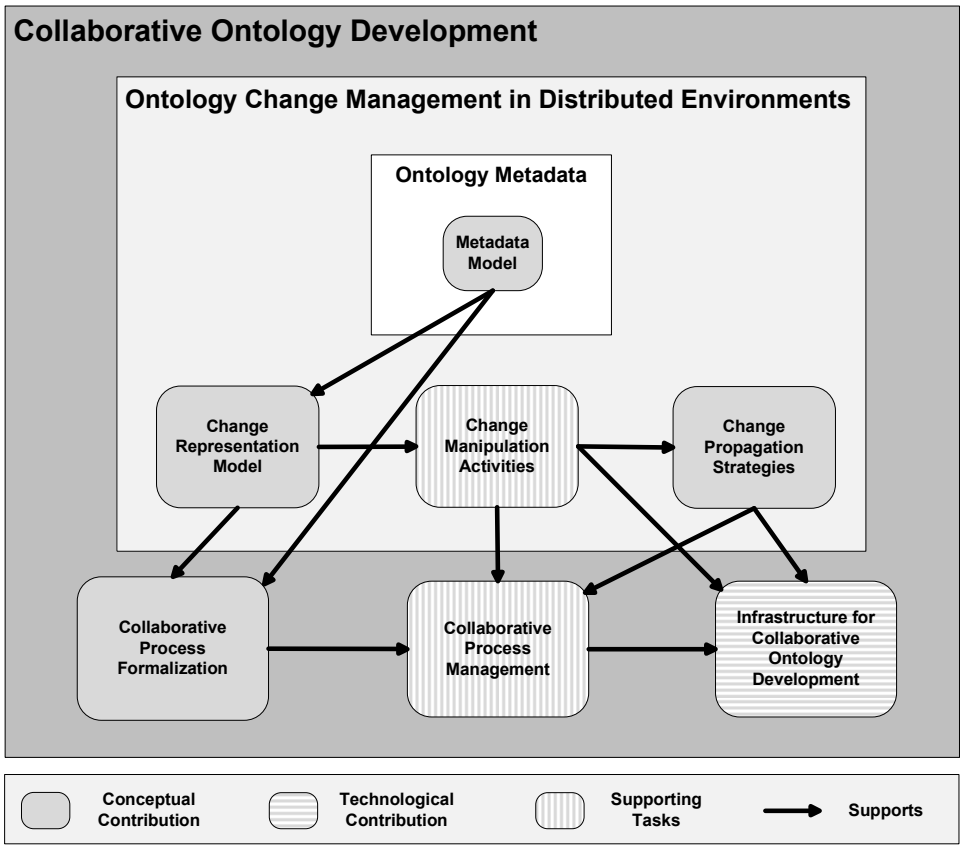


Figure 3.1: Thesis Overview Diagram

The diagram in Figure 3.1 depicts a general overview of these contributions and the relationship between them. In the diagram, the three rectangles represent the three research areas that we are address; the components in an outer rectangle rely on the components of the inner rectangle. The way the components are related to each other is illustrated by the arrows. In detail, the *ontology metadata model* is a core component that supports the other models in our work. The *change representation model* reuses and extends it, and the *collaborative process formalization* relies on the knowledge modeled by the metadata model and the change representation model. Furthermore, the change representation model supports the *changes manipulation activities* (e.g., logging, processing and storing) which in turn support the *change propagation strategies*. Similarly, the *collaborative process management* is supported by the collaborative process formalization, the change manipulation activities and the change propagation strategies. Finally, the previous activities and strategies (change manipulation, change propagation and collaborative process management) support the final *infrastructure for collaborative ontology development*.

3.1 Thesis Objectives

The general objective of the thesis is to support collaborative ontology development activities based on the management of changes in distributed environments. To achieve this overall goal, we have decomposed it in the following conceptual and technological objectives:

Conceptual Objectives

O1. The definition of **conceptual models that provide the foundations to support the management of ontology changes in distributed environments**:

- **O1.1.** A common and community-accepted vocabulary for describing ontologies (also known as ontology metadata model). Based on this model, one will be able to determine, for instance, if an ontology has changed, the relationship between different versions of the ontology, or how the ontology changed at a very coarse granularity level. This model does not provide fine-grained details about these changes between versions.
- **O1.2.** A language-independent model for the representation of ontology changes, which considers a more complete classification of changes compared to the existing approaches. This model allows describing the specific changes applied to an ontology and the chronological order in which they were applied.

O2. The development of **models, methods and strategies for the management of ontology changes in distributed environments to support collaborative ontology development**, including:

- **O2.1.** Strategies for the propagation of ontology changes in distributed environment to:
  - i) Distributed copies of the same ontology. Based on this strategy we will be able to support a distributed control of the changes in collaborative scenarios.
  - ii) Ontology related metadata. This strategy will allow us to keep up-to-date descriptions of ontologies to identify when an ontology has changed.
- **O2.2.** Methods and strategies for the manipulation of ontology changes and the identification of ontology versions to support complementary activities that have to be carried out to provide a complete solution for the management of changes. Examples of these activities are the process of capturing and logging changes and the maintenance of those changes in an appropriate knowledge base that supports advanced storage and retrieval operations.

O2.3. Formalization of the process of collaborative ontology development usually followed by organizations to coordinate, on the one hand, who can change the ontology, and on the other hand, when and how an ontology can be changed. This goal includes the definition of an appropriate model that will allows us to identify the different roles that ontology editors can play during the ontology development process and to determine the way in which changes have been performed, including the policy followed by the team of ontology editors to propose and accept/reject changes.

O2.4. Strategies for the management of the collaborative process to support complementary activities that have to be carried out to provide a complete solution to support the collaborative ontology development. Examples of these activities are the set of tasks required to enforce the process constraints.

Technological Objectives

O3. The development of **an infrastructure that implements the models, methods and strategies proposed for the management of changes in order to support collaborative ontology development in distributed environments**, showing its usability in real scenarios. The infrastructure should include the following components:

O3.1. A distributed ontology registry for the storage and administration of ontology metadata, including ontology change information.

O3.2. Changes manipulation component that supports the capturing, processing and logging of ontology changes.

O3.3. A collaborative workflow management component that is responsible for the application of the constraints imposed by the collaborative process that coordinates who, when and how can change an ontology.

O3.4. User related components for editing and visualizing ontologies (and related information) during collaborative ontology development. These components should provide the appropriate support to the possible actions/operations that users can perform according to the collaborative process.

3.2 Contributions to the State of the Art

In this thesis, we provide solutions to some of the open research and technological problems (see sections 2.1.3, 2.2.4, 2.3.3, 2.4) identified in the scope of this thesis. The solutions achieved for each of the particular goals have been organized and integrated in order to formulate the main contributions of the thesis (see Figure 3.1):

Ontology metadata model, change management in distributed environments and collaborative ontology development. Table 3.1 summarizes the mapping between the objectives identified in section 3.1 and the specific contributions. Additionally, this table summarizes for each contribution, the associated assumptions (c.f. 3.3), hypotheses (c.f. 3.4) and restrictions (c.f. 3.5) of the thesis.

Regarding the ontology metadata model contribution, this thesis presents new advances in the state of the art in the following aspects:

C1. The **Ontology Metadata Vocabulary -OMV-** for the description of ontologies, which reflects the requirements of the majority of ontology developers; this vocabulary has become a community-accepted vocabulary by facilitating the participation of different organizations and experts in the revision and refinement of the model and by allowing the extension of the vocabulary for particular applications or scenarios. OMV allows us to determine whether an ontology has changed (the OMV description changed) but it does not provide detailed information about the specific changes, i.e., specific classes or properties that have changed. This contribution tackles objective O1.1.

Our second contribution deals with some of the aspects of the management of changes: their representation and propagation. In particular, we present new advances in the state of the art in the following aspects:

C2. A layered model for the representation of changes that consists of a language-independent ontology at the top layer, called **Generic Change Ontology**, which can be specialized for different ontology languages in a lower layer. This ontology allows reaching objective O1.2. Our contribution also includes two specializations, i.e., for the OWL 2 ontology language (called **OWL2 Change Ontology**) and for the RDFS ontology language (called **RDFS Change Ontology**). Besides, the model proposes a classification of changes that considers the lowest-level atomic operations that can be performed in an ontology in addition to the other levels (entity, complex) already identified in the existing approaches (e.g., [Sto04] and [Kle04]). The model itself is implemented as an extension of OMV since it addresses a particular kind of ontology metadata, i.e., ontology changes. Moreover, both models complement each other: once we identify that an ontology has changed (using the OMV description), the (generic/specialized) Change Ontology allow us to describe the specific changes applied to the ontology. Figure 3.2 illustrates with a particular example the way these models complement each other. In the figure, the circles represent ontologies while the triangles represent metadata. So, the triangle *OMV O1* represents the metadata associated to ontology O1, the triangle *OMV O1'* represents the metadata associated to ontology O1', and the triangle *OWL2 Change Ontology O1* represents the changes applied from O1 to O1'.

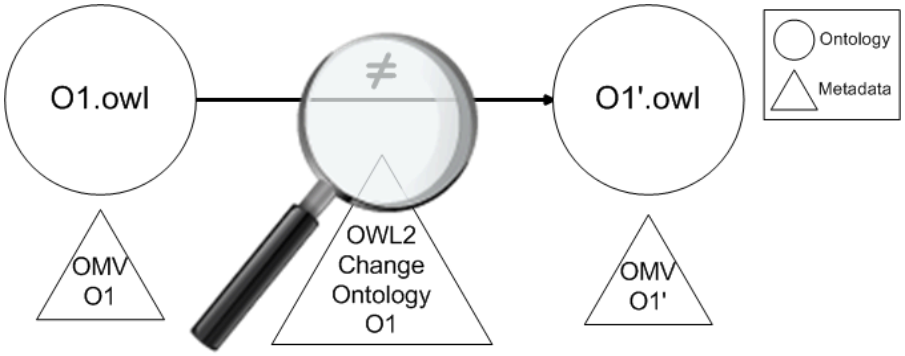


Figure 3.2: Conceptual models relationship: When ontology O1.owl changes to O1'.owl, OMV allows identifying that the ontology has changed. That is, the metadata associated to O1 (OMV O1) is different from the metadata associated to O1' (OMV O1'). Additionally, the OWL2 Change Ontology allows describing the specific changes applied to O1

C3. Strategies for the propagation of changes to distributed copies of the same ontology. We propose a hybrid process consisting in the combination of a push and pull mechanism that allows distributed copies of the ontology to be synchronized, thus enabling ontology engineers to work on the same version of the ontology even when they are working with local copies of it. Unlike the existing approaches, our strategy supports a completely distributed control of the ontology and its changes. This contribution addresses the first part of objective O2.1.

C4. Strategies for the propagation of changes to ontology related metadata in distributed environments. We propose to treat metadata annotations as first-class resources. We define an appropriate model for the management of the lifecycle of ontology metadata annotations in distributed contexts. Based on this model, we propose strategies for the evolution of metadata as a consequence of the changes performed in its related ontology, and for the synchronization of these related entities by means of a notification mechanism. This contribution addresses the second part of objective O2.1.

C5. The **distributed ontology registry called Oyster** that provides the technological support to the methods and strategies for the identification of ontology versions and the manipulation of ontology changes, including the capture (e.g., monitoring, processing and logging), storage and maintenance of changes (similar to the classification of active/traced/state-based versioning approaches discussed in section 2.2.1.2). The models and strategies implemented allows reaching objective O2.2. Also, Oyster provides the technological support for the propagation of changes to distributed copies of an ontol-

ogy by implementing the strategies proposed. Oyster itself is our contribution for objective O3.1.

Finally, our third contribution is the support of the collaborative ontology development process in an organizational setting, using the models and strategies proposed for the management of changes. Our solution is based on the formalization of the process that coordinates the proposal of changes, typical in many organizations. In particular we present new advances in the state of the art in the following aspects:

C6. A formal model based on the definition of collaborative workflows. This model is used for representing the process usually followed by different organizations to coordinate the collaborative activities for the proposal and implementation of ontology changes. A collaborative workflow allows modeling relations among designers, ontology elements, and collaborative tasks (according to [GLP+07]), but the workflow itself does not necessarily have to be developed in a collaborative manner. We propose to consider two different abstraction levels for the collaborative workflow (ontology element level and ontology level) and we describe a concrete model for a particular scenario. Additionally, we propose to implement this model as an ontology (called **Workflow Ontology**) by reusing some of the knowledge of the previous models; we present this ontology specialized for the scenario considered. This work is our contribution for objective O2.3. Finally, we provide some supporting strategies for the management of the collaborative process. We identify the tasks that have to be carried out, including those required to enforce the constraints specified by the collaborative process, and propose strategies to tackle them with the aim of achieving objective O2.4.

C7. An integrated infrastructure implemented within the NeOn Toolkit by means of a set of plugins and extensions (**Change Capturing Plugin and Collaborative Support Feature**) that supports the collaborative ontology development. The infrastructure implements the models and strategies proposed and relies in the distributed ontology registry Oyster for the management of ontology changes in distributed environments. Additionally, appropriate user interfaces have been implemented as part of the Collaborative Support Feature to support the possible actions/operations that users can perform according to the collaborative process. Unlike the existing solutions, our solution supports a formal collaborative process that coordinates the proposal of ontology changes and a completely distributed control for the management of changes. The change capturing plugin is our contribution to achieve objective O3.2, while the collaborative support feature allows reaching objectives O3.3. and O3.4.

The contributions are presented in the document as follows: First, in Chapter 4 we present the contribution C1: The metadata model for describing ontologies (OMV) that provides the foundation for the management of ontology changes.

Then, in Chapter 5 we present our solution for the management of changes in distributed environments. We present our layered model for the representation of changes (contribution C2), including the Generic Change Ontology and two specializations (OWL2 Change Ontology and RDFS Change Ontology). Next, we describe our strategies for the propagation of ontology changes to (i) distributed copies of the same ontology (contribution C3) and to (ii) the ontology related metadata (contribution C4). Subsequently, we present complementary methods and strategies supporting change manipulation tasks and the identification of ontology versions (contribution C5). Later, in Chapter 6 we present the contribution C6, our solution to support collaborative ontology development. We discuss how to formalize the collaborative process that coordinates the proposal of ontology changes, typical in many organizations, by means of a collaborative workflow model that considers two different abstraction levels (ontology element level and ontology level), and illustrate this model for a particular scenario. Next, we present the workflow ontology that implements the model, followed by the strategies for the management of this process during the ontology development. Finally, in Chapter 7 we present the evaluation of our work, including the description of the infrastructure that provides the technological support for the previous models and strategies. We present the distributed ontology registry, Oyster (contribution C5), followed by the Change Capturing Plugin and Collaborative Support Feature implemented within the NeOn Toolkit (contribution C7).

3.3 Assumptions

A1. The goal of a metadata vocabulary is the search for consensus among users.

A2. Ontologies are dynamic entities that evolve over time as a consequence of changes, for instance, in the domain modeled or in the domain experts knowledge.

A3. We consider scenarios where ontologies change maintaining consistency.

A4. One of the main goals of the model for the representation of ontology changes is to facilitate its reusability for different ontology languages and to maximize its usability among users and applications.

A5. The set of complex changes that can be performed over an ontology cannot be complete or exhaustive. One can always combine two or more operations to define operations of coarser granularity.

A6. The development and maintenance of ontologies has been transformed from a process traditionally performed by one ontology engineer into a process performed collaboratively by a team of ontology engineers, who may be geographically distributed and that play different roles.

A7. The collaborative process followed by the Food and Agricultural Organization[1] (FAO), for the development of ontologies, is a representative process for different organizations.

A8. The team of ontology engineers that work collaboratively in the development and maintenance of ontologies requires, in some scenarios, to use and modify local copies of the ontology.

A9. The centralized management of ontology changes is not adequate for some scenarios of collaborative ontology development or maintenance.

A10. The NeOn Toolkit provides a stable infrastructure for the development of ontologies that can be easily extended with additional functionalities.

3.4 Hypotheses

General Hypotheses

H1. The management of ontology changes in distributed environments supports the collaborative ontology development and maintenance.

H2. The management of ontology changes can be ontology-driven.

H3. The models, methods and strategies proposed can be implemented in an infrastructure that facilitates the work of ontology editors when developing or maintaining an ontology collaboratively.

Specific Hypotheses

H4. The metadata model for describing ontologies reflects the requirements of the majority of ontology users.

H5. The metadata model for describing ontologies can be extended and refined for particular application or scenarios.

H6. The layered approach for the representation of changes provides an ontology language-independent layer that can be reused and specialized for different ontology representation languages.

H7. The layered approach for the representation of changes unifies many of the features of different existing approaches and extend them by considering the atomic change operations that can be performed in an ontology language.

[1] `http://www.fao.org`

H8. The specialization of the representation model for the OWL 2 ontology language is complete, i.e., every possible OWL 2 operation can be expressed with the ontology.

H9. The collaborative process usually followed by organizations for the management of ontology change proposals can be modeled by means of collaborative editorial workflows.

H10. The models for the representation of changes and for the collaborative process are adequate with regard to the users' requirements.

H11. The strategies for the propagation of changes to distributed copies of an ontology supports a distributed control during the collaborative ontology development and maintenance.

H12. The lifecycle of ontology metadata annotations can be formally represented to manage the propagation of ontology changes to their related metadata.

H13. The implemented infrastructure is usable with regard to the efficiency, effectiveness and user's satisfaction.

3.5 Restrictions

R1. The models and strategies proposed for the management of changes only tackle some of the activities of the ontology evolution process. For instance, the discovery of changes, inconsistency problems, or evaluation techniques of the resulting ontology are not dealt with. However, even though we do not tackle inconsistency problems, we rely on the existing plugins of the ontology engineering platform, NeOn Toolkit, for dealing with them. Hence our approach is different from the ontology modification approach defined by [Yil06] and [Sto04], because we acknowledge the importance of keeping an ontology consistent while applying changes. Additionally, we tackle some aspects of ontology versioning related to the identification and maintenance of different ontology versions.

R2. We do not propose conflict identification or resolution mechanisms during the propagation of changes to distributed copies of the same ontology. Currently, we can only minimize conflicts by the periodic execution of the synchronization process in an automatic manner.

R3. We do not include models or technological support for the argumentation of changes, but our solution can be easily integrated with existing argumentation solutions, e.g., Cicero.

R4. We do not consider open environments, such as the Web, for the collaborative ontology development and maintenance. The scenarios we address, are

those typical in closed environments like organizations, where the collaborative ontology development and maintenance usually follows a pre-defined process for the coordination of the change proposals, and the history of ontology changes can be traced during the process.

R5. According to the dimensions identified in [NCLM06] (see section 2.3.1) to classify scenarios for collaborative ontology evolution, our solution can be categorized as continuous editing (support of undo operations in the latest ontology version) with curation activity support and monitored (the tool records the changes and metadata about these changes). Additionally, our solution supports both synchronous and asynchronous editing modes. However, unlike the characteristics of these collaboration modes, we do not require to have a common central copy of the ontology. Moreover, the monitored mode of our solution makes it similar to the classification of active/traced/state-based versioning approaches defined in [Lia06] and [LM07] (see section 2.2.1.2).

R6. We only consider semantic web languages (e.g., OWL and RDFS) for the representation of ontology changes.

R7. We do not provide an algebra of change operations.

R8. The change representation model only considers axiomatic changes, i.e., lexical changes, such as renaming labels are not supported yet.

R9. We deal with OWL 2 ontologies instead of OWL 1. The reason is partly because the new version of the Web Ontology Language is rapidly becoming the new recommendation, and partly because this language takes into account some aspects regarding the maintenance of ontology versions.

R10. The NeOn Toolkit does not support yet all OWL 2 operations.

Table 3.1: Mapping Between Thesis Objectives and Thesis Contributions With Associated Assumptions, Hypotheses and Restrictions

Objective	**Contribution ([Assumptions],[Hypotheses[a]],[Restrictions])**
O1.1. A common and community-accepted vocabulary for describing ontologies.	**C1.** Ontology Metadata Vocabulary -OMV- ([**A1**], [**H4**,**H5**], [**R1**,**R3**]).
O1.2. A language-independent model for the representation of ontology changes.	**C2.** Generic Change Ontology ([**A2**,**A4**,**A5**], [**H6**,**H7**,**H8**,**H10**], [**R1**, **R3**,**R6**,**R7**,**R8**]).
O2.1. Strategies for the propagation of ontology changes.	**C3.** Strategies for the propagation of changes to distributed copies of the same ontology & **C4.** Strategies for the propagation of changes to ontology related metadata in distributed environments ([**A2**,**A3**,**A8**,**A9**], [**H11**,**H12**], [**R1**,**R2**]).
O2.2. Methods and strategies for the manipulation of ontology changes and the identification of ontology versions.	**C5.** Change manipulation and identification of ontology versions ([**A2**, **A3**,**A8**,**A9**], [**H11**,**H12**], [**R1**,**R4**,**R5**]).
O2.3. Formalization of the process of collaborative ontology development.	**C6.** A formal model based on the definition of collaborative workflows, implemented in Workflow Ontology ([**A6**,**A7**], [**H9**,**H10**], [**R3**,**R4**,**R5**]).
O2.4. Methods and strategies for the management of the collaborative process.	**C6.** Collaborative process management ([**A6**,**A8**,**A9**], [**H9**], [**R4**,**R5**]).
O3.1. A distributed ontology registry.	**C5.** Oyster ([**A9**], [**H11**,**H12**,**H13**], [**R2**,**R5**]).
O3.2. Changes manipulation component.	**C7.** Change Capturing Plugin ([**A8**,**A9**,**A10**], [**H13**], [**R2**,**R3**,**R4**,**R5**, **R9**,**R10**]).
O3.3. A collaborative workflow management component.	**C7.** Collaborative Support Feature ([**A8**,**A10**], [**H13**], [**R3**,**R4**,**R5**]).
O3.4. User related components.	**C7.** Collaborative Support Feature ([**A8**,**A10**], [**H13**], [**R3**,**R5**,**R10**]).

[a]General hypotheses **H1**, **H2** and **H3** apply to all contributions.

Chapter 4

ONTOLOGY METADATA FOR CHANGE MANAGEMENT

The need of a vocabulary for describing ontologies has been addressed in the past by previous efforts. However, as we discussed in section 2.1.3, existing vocabularies have some limitations (e.g., most of them consist only of a list of property-value pairs, or they are implicitly defined within existing systems and repositories) and, consequently, there is still a lack of a standard or community-accepted ontology metadata vocabulary.

Hence, in this chapter, we present our metadata model for describing ontologies and related entities. As we discussed in the previous chapter, this model is a core component in the context of this thesis. First, it allows identifying whenever an ontology has changed by a simple inspection of its metadata description instead of performing the time-consuming task of analyzing the ontology itself, i.e., when the ontology changes, its metadata changes accordingly. Second, it provides a high level overview of how an ontology changed (e.g., additional classes or properties were defined and the domain was specialized). Third, it is the base model for the other models proposed in this thesis that specialize or reuse it (as we explain later in sections 5.1 and 6.2.1).

In a more general scope, an ontology metadata model has other benefits. In particular for ontology reuse, ontology metadata plays an important role in the widespread dissemination of ontology-driven technologies (e.g., increases ontology quality and reduces the cost of ontology development). Currently, in most cases, existing ontologies are in pure form without any additional information (e.g., author, covered domain and format). This is in part due to the lack of standards for describing ontologies, which at the same time limit the development of fully-fledged ontology repositories on the Web. Hence, in order to solve this situation, we need first to have a community-accepted vocabulary for describing ontologies that reflects the requirements of the majority of users and that can be easily extended for specific purposes. Furthermore, we will need tools compliant with this vocabulary that facilitate the exchange and reuse of ontologies.

Thus, the first step towards the definition of our ontology metadata model was the specification of requirements. For this analysis we considered the broader scope of ontology reuse, as it subsumes the description of the ontology that helps to identify whether an ontology changes and that provides the general overview of how it changes. Note that the ontology information necessary to facilitate ontology reuse (e.g., domain modeled in the ontology, technical information and statistical measures), should be updated whenever an ontology changes. For example, when additional classes are defined in an ontology, its related metadata may need to be updated in several ways, e.g., increase the number of classes in the ontology, specialize the domain of the ontology, update the ontology contributors and include additional key classes. Therefore, by analyzing the ontology metadata, one can easily identify if an ontology changed or have a general overview of how it changed.

In the following sections, we start by describing the requirements identified during the development of our ontology metadata model (section 4.1). Since those requirements cover the general scope of ontology reuse, when appropriate, we remark how such requirements are useful in the specific context of identifying ontology changes. After the analysis, we detail our metadata model in section 4.2. As a result of our initial analysis, in addition to contributing to the identification of ontology changes, our metadata model provides a contribution to the general scope of ontology reuse.

OMV constitutes contribution **C1** in 3.2, which addresses objective **O1.1** in 3.1.

4.1 Ontology Metadata Requirements

As a result of a systematic survey of the state of the art in the area of ontology reuse, we have elaborated an inventory of requirements for the metadata model. Besides analytical activities, we conducted extensive literature research, focused on theoretical methods [PM01, GPS99, LTGP04], and also on case studies on reusing existing ontologies [UHW+98, RVMS99, PBMT05]. Our aim was to identify the real-world needs of the community with respect to a descriptive metadata format for ontologies. Further on, the requirements analysis phase was complemented by a comparative study of existing (ontology-independent) metadata models and tools, such as ontology repositories and libraries that (implicitly) make use of some form of ontology metadata.

Several aspects to be considered in ontology metadata representation are definitely similar to those of other more general metadata standards such as Dublin Core. Differences arise, however, if we consider the semantic nature of ontologies, which are much more than plain Web information sources. For instance, as we mentioned in section 2.1.3, aspects related to the application scenario, scope, purpose, or evaluation results are essential when describing ontologies. Also, besides structural and technical information, ontologies have to be described in terms

of descriptive metadata, such as provenance information, ontology categorizations, underlying methodologies or knowledge representation paradigms that are specific for ontologies. The main requirements identified in this process step are the following:

Accessibility: Metadata should be accessible and processable for both humans and machines. Whereas the human-driven aspects are ensured by the usage of natural language concept names, the machine-readability requirement can be implemented by the usage of Web-compatible representation languages (such as XML or Semantic Web languages, see below). Furthermore, having metadata in processable format will facilitate the implementation of tools that use or manage ontology related metadata (e.g., ontology changes).

Usability: A metadata model should 1) reflect the needs of the majority of ontology users, as reported by existing case studies in ontology reuse, but at the same time 2) allow proprietary extensions and refinements in particular application scenarios (e.g., ontology change management). From a content perspective, usability can be maximized by taking into account multiple metadata types, which correspond to specific viewpoints on the ontological resources and are applied in various application tasks. Despite the broad understanding of the metadata concept and the use cases associated to each definition, several key aspects of metadata information have already established across computer science fields [NIS04]:

- **Structural metadata** relates to statistical measures on the graph structure underlying an ontology. In particular we mention the number of specific ontological primitives (e.g., number of classes and individuals). The availability of structural metadata influences the usability of an ontology in a concrete application scenario, because size and structure parameters constrain the type of tools and methods that are applied to aiding the reuse process. For instance, as it has been analyzed in the past (e.g., [GHT06]), most ontology reasoners have still scalability issues when dealing with large ontologies. Furthermore, structural metadata provides core information to identify when an ontology changes (e.g., a different number of classes or individuals).
- **Descriptive metadata** relates to the domain modeled in the ontology in form of keywords, topic classifications, textual descriptions of the ontology contents, etc. This type of metadata plays a crucial role in the selection of appropriate reuse candidates, a process that includes requirements with respect to the domain of the ontologies to be reused. Moreover, descriptive metadata is highly useful when identifying ontology changes from a high level point of view (e.g., the domain has been specialized and the description has been updated).

- **Administrative metadata** provides information to help manage ontologies, such as when and how it was created, rights management, file format, and other technical information. Obviously, information like the date of modification of the ontology is also useful to identify when an ontology has changed.

Interoperability: Similarly to the ontology it describes, metadata information should be available in a form that facilitates metadata exchange among applications. While the syntactical aspects of interoperability are covered by the usage of standard representation languages (see "Accessibility"), the semantical interoperability among machines handling ontology metadata information can be ensured by means of a formal and explicit representation of the meaning of the metadata entities (by conceptualizing the metadata vocabulary itself as an ontology).

4.2 OMV - Ontology Metadata Vocabulary

This section presents the ontology metadata vocabulary –OMV–; the first part provides an overview of the core design principles applied to the development of the OMV metadata model; then, we describe in detail the core of such a model; next, we present implementation and practical aspects; finally, we provide an introduction to the OMV extensions.

4.2.1 Core and Extensions

Following the usability constraints identified during the requirements analysis, we decided to design the OMV scheme modularly, distinguishing between the OMV Core and various OMV Extensions. The former captures information that is expected to be relevant to the majority of ontology reuse settings. However, in order to allow ontology developers and users to specify task –or application– specific ontology-related information, we foresee the development of OMV extension modules, which are physically separated from the core scheme while remaining compatible to it.

4.2.2 Metadata Organization and Categorization

In the following we present the organization and categorization of metadata (entities) in two dimensions, which provide a structured overview of the OMV ontology.

Property Appropriation

We organize metadata entities according to the impact on the prospected reusability of the described ontological content as presented in the following list:

- **Required** - mandatory metadata elements. Any missing entry in this category leads to an incomplete description of the ontology.

- **Optional** - important metadata facts, but not strongly required.

- **Extensional** - specialized metadata entities, which are not considered to be part of the core metadata scheme.

Property Categorisation

Complementary to the previous classification, we organize the metadata elements according to the type and purpose of the information contained as follows:

- **General** - elements providing general information about the ontology.

- **Availability** - information about the location of the ontology (e.g., its URI or URL where the ontology is published on the Web).

- **Applicability** - information about the intended usage or scope of the ontology.

- **Format** - information about the physical representation of the resource. In terms of ontologies, these elements include information about the representation language(s) in which the ontology is formalized.

- **Provenance** - information about the organizations contributing to the creation of the ontology.

- **Relationship** - information about relationships to other resources. This category includes versioning, as well as conceptual relationships such as extensions, generalization/specialization and imports.

- **Statistics** - various metrics on the underlying graph topology of an ontology (e.g., number of classes).

- **Other** - information not covered in the categories listed above.

Note that the classification dimensions introduced above (appropriation and categorization) are intended to be considered when implementing several metadata support facilities. The first dimension is relevant for a metadata creation service since it ensures a minimal set of useful metadata entries for each of the described resources. The second can be used in various settings mainly to reduce the user-perceived complexity of the metadata scheme, whose elements can be structured according to the corresponding categories.

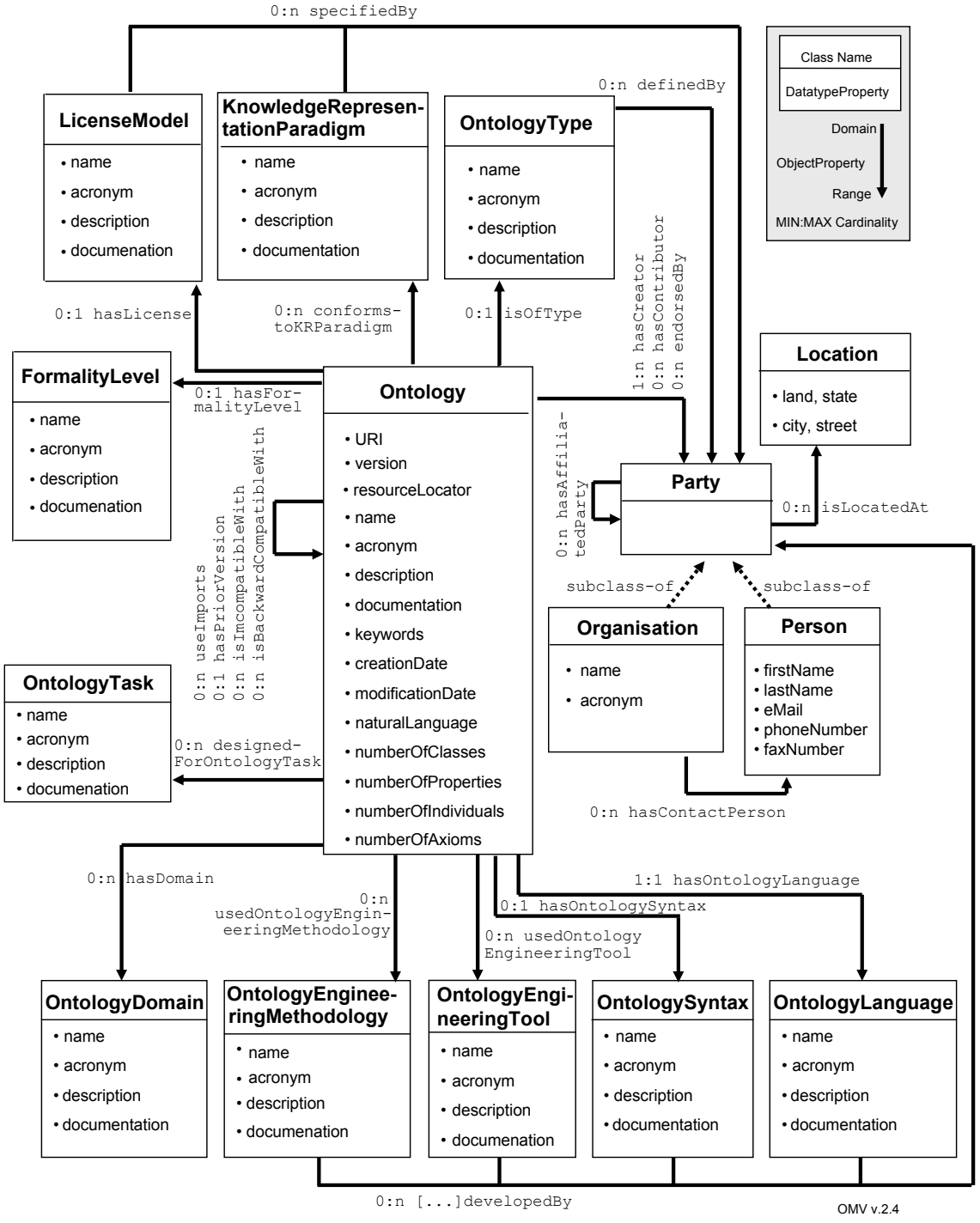


Figure 4.1: OMV overview

4.2.3 OMV Core Metadata Entities

The main classes and properties of the OMV ontology are illustrated in Figure 4.1[1].

Besides the main class `Ontology`, the metadata model contains elements describing various aspects related to the creation, management and usage of an ontology. We will briefly discuss these in the following. In a typical ontology engineering process, `Person`(s) or `Organization`(s) develops ontologies. We group these two classes under the generic class `Party` by a `subclass-of` relation. A `Party` can have several locations by referring to a `Location` individual and can `create`, `contribute` to ontological resources, i.e., `Ontology` implementations. Review details and further information can be captured in an extensional OMV module. Further on we provide information about the engineering process the ontology originally resulted from in terms of the classes `OntologyEngineeringMethodology`, `OntologyEngineeringTool` and the attributes `version`, `status`, `creationDate` and `modificationDate`. Again these can be elaborated as an extension of the core metadata scheme. The usage history of the ontology is modeled by classes such as the `OntologyTask` and `LicenceModel`. The scheme also contains a representation of the most significant intrinsic features of an ontology. Details on ontology languages are representable with the help of the classes `OntologySyntax`, `OntologyLanguage` and `KnowledgeRepresentationParadigm`. Ontologies might be categorized along a multitude of dimensions. One of the most popular classification differentiates among *application*, *domain*, *core*, *task* and *upper-level* ontologies. A further classification relies on their level of formality and types of Knowledge Representation (KR) primitives supported, introducing *catalogues*, *glossaries*, *thesauri*, *taxonomies*, *frames*, etc., as types of ontologies. The former categories can be modeled as individuals of the class `OntologyType`, while generic formality levels are introduced with the help of the class `FormalityLevel`. The domain the ontology describes is represented by the class `OntologyDomain` that references a pre-defined topic hierarchy such as the DMOZ hierarchy. Further content information can be provided as values of the attributes `description`, `keywords`, and `documentation`. Finally, the metadata scheme gives an overview of the graph topology of an `Ontology` with the help of several graph-related metrics represented as integer values of the attributes `numberOfClasses`, `numberOfProperties`, `numberOfAxioms`, `numberOfIndividuals`.

4.2.4 Ontological Representation

Following the accessibility and interoperability requirements, as well as the nature of the metadata, which is intended to describe ontologies, the conceptual model

[1]Please notice, that not all classes and properties are included. The ontology is available for download in several ontology formats at `http://omv.ontoware.org/`

designed in the previous steps (4.2.1, 4.2.2 and 4.2.3), was implemented in OWL[2]. We decided to use OWL for the representation of our metadata model because it was estimated that an implementation as XML-Schema or DTD would restrict the functionality of the ontology management tools using the metadata information (mainly in terms of retrieval capabilities) and would impede metadata exchange at semantical level. Further on, a language such as RDFS does not provide a means to distinguish between required and optional metadata properties. The implementation was performed manually by means of a general-purpose ontology editor.

Additionally, a metadata element is modeled either by means of classes and individuals or by means of valued properties. The decision for one of these two alternatives was justified by the complexity of the corresponding metadata element. If the value/content of a metadata element can be easily mapped to conventional data types (numerical, literal, list values), the metadata element is usually represented as a DatatypeProperty. Complex metadata elements which do not fall into the previous category are modeled by means of additional classes linked by ObjectProperties.

Finally, OMV implements the appropriate properties of metadata entities by different means: the required and optional metadata entities are implemented in OMV core with the appropriate cardinality constraint, while the extensional metadata entities are implemented in the different OMV extensions.

4.2.5 Identification, Versioning and Location

An important issue that has to be addressed when describing ontologies is the ability to identify and manage multiple versions and physical representations of one ontology. The OWL ontology language itself did not provide the means to address this issue until the latest release of OWL 2[3], which partially addresses it (see section 2.2.1.2): OWL 2, which uses International Resource Identifiers (IRIs) [DS05] to identify ontologies (and their elements), extends OWL 1 that uses Uniform Resource Identifiers (URIs) [BLFM05]. Hence, in OWL 2, an ontology may have a version IRI, which is used to identify the version of the ontology. The version IRI may, but need not be equal to the ontology IRI. However, the specification provides no mechanism for enforcing these constraints across the entire Web.

Versioning In general, a version is a variant of an ontology that is usually created after applying changes to an existing variant. Therefore, we need a way to unambiguously identify the different versions, as well as to keep track of the relationships between them. Based on [KF01], we consider that changes in ontologies are caused by (i) changes in the domain, (ii) changes in the shared conceptualization and (iii) changes in the specification of the shared conceptualization, i.e.,

[2] In the remainder of this thesis when OWL appears without any version information, it refers to OWL 1. As opposed, when referring to OWL 2 we explicitly note it.

[3] `http://www.w3.org/TR/2008/WD-owl2-syntax-20081202/`

implementation. Taking the definition of an ontology as a specification of a conceptualization, (i) and (ii) are semantic changes that lead to the creation of a new conceptualization, while (iii) is just a change in the representation of the same conceptualization (also known as a new revision) (e.g., updates of natural language descriptions of ontology elements). In any case, the change(s) leads to a different physical representation of the ontology, i.e., different version. Consequently, it should be possible to identify each of those versions.

However, as we mentioned above, OWL 1 does not distinguish at all between the notion of an ontology and a version of an ontology. It may thus be that different versions of an ontology carry the same logical URI. Only in the latest specification of OWL 2, it is proposed that the ontology IRI and the version IRI together identify a particular version from an ontology series, i.e., the set of all the versions of a particular ontology identified using a common ontology IRI. In each ontology series, exactly one ontology version is regarded as the current one. Consequently, typically (as OWL 2 has just been released) existing ontologies are identified by an URI, which according to [BLFM05] is a compact string of characters for identifying an abstract or a physical resource, but in practice different versions of same ontology share the same URI.

In OMV we describe a particular representation of an ontology, i.e., an ontology in a particular version at a particular physical location. That means that every different version of an ontology has a different OMV related metadata.

Currently however, most ontologies either do not provide any version information at all or the ontology editors explicitly do not want to change the version of the ontology after making some changes. In those cases, whenever the ontology changes, the related OMV annotation will have to be updated accordingly instead of creating a new OMV individual (e.g., update the ontology modification date).

Resource Location In addition to the issue of versioning, an ontology (or a version of an ontology) can be located at different locations. Thus, ontologies with the same logical URI (IRI in OWL 2) may exist at different physical locations, possibly even with different content. Similar to versioning, only the latest OWL 2 specification provides some conventions for the location of an ontology: each ontology is associated with an ontology document, which physically contains the ontology stored in a particular way. In an ontology series identified by IRI *OI*, the document containing the current version Vc of that series should be accessible from *OI*, and if available, from its version IRI *VcI* as well. A particular version Vx of *OI*, with version IRI *VxI*, should be accessible only from *VxI*.

In practice, however, (as OWL 2 has just been released) the location of the ontology is not necessarily equal to the URI of the ontology. So, following the approach for versioning, we rely on a composite identifier consisting of the logical identifier, i.e., URI (ontology IRI in OWL 2), plus optional version identifier (version IRI in OWL 2), and a resource locator that specifies the actual physical location.

Of course, the optional version identifier and the resource locator can be combined, such that we end up with a tripartite identifier (URI, version, resource locator).

OMV identity Based on the previous discussion, we propose the following composite URI to identify an OMV individual that should be treated just as one possible approach, i.e., the system implementing OMV can choose its own OMV identity:

```
Ontology URI + ?[version=<version >;]location=<resourceLocator >#metadata
```

where the *resourceLocator* is the physical location of the ontology, i.e., the resourceLocator property, and *version* is the ontology version, i.e., the version property

Illustrative Example In order to clarify the discussion, consider the following scenario: Initially, we have the first implementation of ontology OWLODM (`http://owlodm.ontoware.org/OWL1.0`) which provides a metamodel for the ontology language OWL 1.0. A fragment of the OMV description for OWLODM version 1.0 is the following:

```
<omv:Ontology rdf:about="&j;OWL1.0?version=1.0;location=
http://ontoware.org/frs/download.php/307/owl10.owl#metadata">
    <omv:URI rdf:datatype="&xsd;string">http://owlodm.ontoware.org/OWL1.0
    </omv:URI>
    <omv:version rdf:datatype="&xsd;string">1.0</omv:version>
    <omv:resourceLocator rdf:datatype="&xsd;string">
    http://ontoware.org/frs/download.php/307/owl10.owl</omv:resourceLocator>
    <omv:acronym rdf:datatype="&xsd;string">OWLODM</omv:acronym>
    <omv:description rdf:datatype="&xsd;string">OWL Object Definition
    Metamodel (ODM) allows interoperability of OWL ontologies with
    MOF-compatible software environments</omv:description>
    <omv:name rdf:datatype="&xsd;string">OWL Ontology Definition Metamodel
    </omv:name>
    <omv:numberOfClasses rdf:datatype="&xsd;unsignedInt">35
    </omv:numberOfClasses>
    <omv:numberOfProperties rdf:datatype="&xsd;unsignedInt">22
    </omv:numberOfProperties>
    <omv:creationDate rdf:datatype="&xsd;string">2007-02-12
    </omv:creationDate>
    <omv:hasCreator rdf:resource="#PeterHaase"/>
    <omv:hasDomain rdf:resource="&c;Knowledge Representation"/>
    ...
</omv:Ontology>
```

A change in the domain modeled by OWLODM (the definition of OWL 1.1) was reflected in a new *version* of the OWLODM ontology, namely version 1.1. This change led to a semantic change of the ontology (change of type (i)), and therefore a new URI was defined for OWLODM (`http://owlodm.ontoware.org/OWL1.1`). A fragment of the OMV description for OWLODM version 1.1 is the following ("–" highlights the lines different from the previous OMV description):

```
<omv:Ontology rdf:about="&j;OWL1.1?version=1.1;location=
http://ontoware.org/frs/download.php/365/owl11.owl#metadata">
  --<omv:URI rdf:datatype="&xsd;string">http://owlodm.ontoware.org/OWL1.1
    </omv:URI>
  --<omv:version rdf:datatype="&xsd;string">1.1</omv:version>
  --<omv:resourceLocator rdf:datatype="&xsd;string">
    http://ontoware.org/frs/download.php/365/owl11.owl</omv:resourceLocator>
    <omv:acronym rdf:datatype="&xsd;string">OWLODM</omv:acronym>
    <omv:description rdf:datatype="&xsd;string">OWL Object Definition
    Metamodel (ODM) allows interoperability of OWL ontologies with
    MOF-compatible software environments</omv:description>
    <omv:name rdf:datatype="&xsd;string">OWL Ontology Definition Metamodel
    </omv:name>
  --<omv:numberOfClasses rdf:datatype="&xsd;unsignedInt">76
    </omv:numberOfClasses>
  --<omv:numberOfProperties rdf:datatype="&xsd;unsignedInt">35
    </omv:numberOfProperties>
  --<omv:creationDate rdf:datatype="&xsd;string">2007-08-09
    </omv:creationDate>
    <omv:hasCreator rdf:resource="#PeterHaase"/>
    <omv:hasDomain rdf:resource="&c;Knowledge Representation"/>
    ...
</omv:Ontology>
```

Finally, a new version of OWLODM (version 1.2) was released as a result of a refinement. In this case, the change was at the level of the specification of the ontology (change of type (iii)), in particular the renaming of a property and hence the URI was not updated. A fragment of the OMV description for the OWLODM version 1.2 is the following ("–" highlights the lines different from the previous OMV description):

```
<omv:Ontology rdf:about="&j;OWL1.1?version=1.2;location=
http://ontoware.org/frs/download.php/366/owl11.owl#metadata">
    <omv:URI rdf:datatype="&xsd;string">http://owlodm.ontoware.org/OWL1.1
    </omv:URI>
  --<omv:version rdf:datatype="&xsd;string">1.2</omv:version>
  --<omv:resourceLocator rdf:datatype="&xsd;string">
    http://ontoware.org/frs/download.php/366/owl11.owl</omv:resourceLocator>
    <omv:acronym rdf:datatype="&xsd;string">OWLODM</omv:acronym>
    <omv:description rdf:datatype="&xsd;string">OWL Object Definition
    Metamodel (ODM) allows interoperability of OWL ontologies with
    MOF-compatible software environments</omv:description>
    <omv:name rdf:datatype="&xsd;string">OWL Ontology Definition Metamodel
    </omv:name>
    <omv:numberOfClasses rdf:datatype="&xsd;unsignedInt">76
    </omv:numberOfClasses>
    <omv:numberOfProperties rdf:datatype="&xsd;unsignedInt">35
    </omv:numberOfProperties>
  --<omv:creationDate rdf:datatype="&xsd;string">2007-08-10
    </omv:creationDate>
    <omv:hasCreator rdf:resource="#PeterHaase"/>
    <omv:hasDomain rdf:resource="&c;Knowledge Representation"/>
    ...
</omv:Ontology>
```

As we can see from the previous simple example, the URI is not enough to identify individually each version of the ontology. Besides, in practice not every semantic change leads to the definition of a new URI (as in this example). Even

more, in this example the URI plus the version was enough to identify each physical implementation; however, it could also be possible that the same ontology version was located at two (or more) different physical locations, where each of them could have even different content. In that case we would also need the location of the ontology to identify a particular implementation, i.e., URI plus version plus location.

4.2.6 OMV Extensions

The OMV core metadata is intended to evolve towards a commonly agreed scheme for Semantic Web ontologies. In contrast to this ambitious goal, we are aware that for specific domains, tasks or communities extensions in any direction might be required. These extensions should be compatible to the OMV core, but in the same time they should fulfill the requirements of a domain, task or community-driven setting.

The character of an OMV extension is a metadata ontology itself that imports the OMV core ontology. There are no restricting modeling guidelines to be met. However, the technical report of OMV [PHH08] provides a basic inventory of design decisions and guidelines, which are recommended to be applied for the extension modules.

Some of the existing OMV extensions were developed in collaboration with different institutions. The available extensions[4] are the Generic Change Ontology, which models changes to an ontology (described in section 5.1), the lexOMV extension [MPdCSF+07] that models the linguistic or multilingual data contained in the ontology, the modules extension that represents the description of ontology modules [dHR+08], the peer extension that captures information of peers sharing metadata about ontologies and related entities (e.g., mappings and modules) [WHP07], and the mapping extension that describes mappings between heterogeneous ontologies.

Figure 4.2 illustrates OMV core along the current OMV extensions and their relationships.

4.3 Summary

As we can see from the description of OMV in 4.2, we address the requirements identified in section 4.1 by different means: first, OMV is implemented in OWL to address the accessibility and interoperability requirements. Second, to address the usability requirement OMV is designed modularly, distinguishing between the OMV Core and various OMV Extensions. Third, OMV core captures the key aspects of the ontology metadata information, including structural metadata though several data properties (e.g., `numberOfClasses`, `numberOfProperties`, `numberOfIndividuals` and `numberOfAxioms`),

[4] OMV Extensions are also available at `http://omv.ontoware.org`

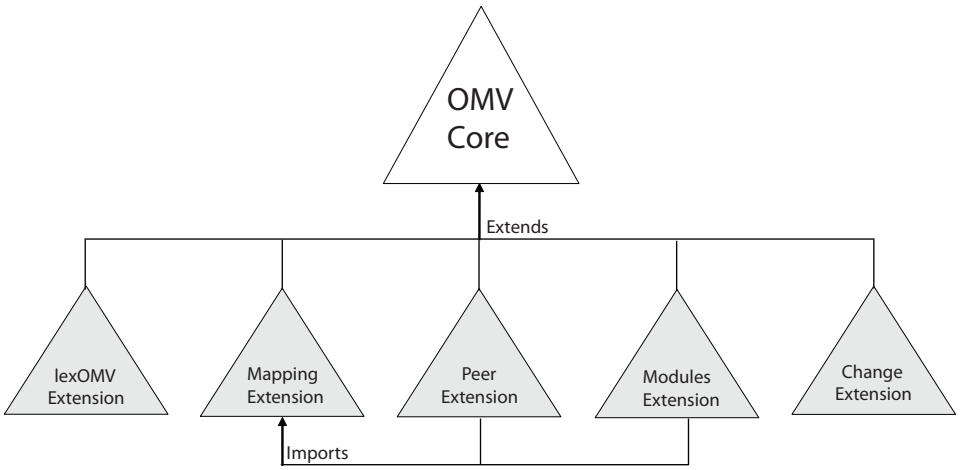


Figure 4.2: OMV Extensions

descriptive metadata though several data properties (e.g., `keywords`, `description`, `notes`, `acronym` and `documentation`) and object properties (e.g., `hasDomain`, `isOfType`, `endorsedBy`, `designedForOntologyTask` and `hasFormalityLevel`) and administrative metadata through several data properties (e.g., `creationDate`, `modificationDate`, `version` and `resourceLocator`) and object properties (e.g., `hasCreator`, `hasContributor`, `hasLicense`, `usedOntologyEngineeringMethodology`, `hasOntologySyntax`, `hasOntologyLanguage`, `conformsToKRParadigm` and `usedOntologyEngineeringTool`). Additionally, OMV implements the appropriation properties of metadata entities described in section 4.2.2 by means of cardinality constraints and the implementation of extensions.

The modular approach of OMV **allows reflecting the requirements of the majority of ontology developers but, at the same time, allows proprietary extensions and refinements for particular application scenarios**. Additionally, as we discussed in section 4.1, **OMV allows us to determine if an ontology has changed and provides a high-level overview of how it has changed**. Consequently, OMV covers contribution **C1**, which addresses objective **O1.1** of this thesis (see Chapter 3).

Furthermore, **OMV plays an important role in the ontology reuse task** by facilitating the discovery and exchange of ontologies, fostering the widespread dissemination of ontology-driven technologies and the development of fully-fledged ontology repositories on the Web.

In Chapter 7 we introduce, among others, the technological support we provide for OMV that illustrates some practical applications of OMV.

Chapter 5

CHANGE MANAGEMENT IN DISTRIBUTED ENVIRONMENTS

In this chapter, we present our solution for managing ontology changes in distributed environments. It includes the definition of a layered model, formally represented as an ontology, for describing changes in ontologies (detailed in section 5.1). This model sets the basis for the methods and strategies that we provide for the manipulation and propagation of changes in distributed environments. In our solution, the manipulation of changes (described in section 5.2) includes supporting activities for capturing (e.g., monitoring, processing and logging), storing and maintaining changes based on a distributed registry, as well as the management of ontology versions. Based on the previous models and methods, we present (in section 5.3) strategies for the propagation of changes to (i) distributed copies of the same ontology, each of them possibly in a different physical location and edited by a different user, and to (ii) the ontology related metadata.

Regarding the contributions introduced in section 3.2, which address the corresponding objectives in section 3.1, we present here the following: section 5.1 describes contribution **C2** (which addresses objective **O1.2**), section 5.2 is part of contribution **C5** (which addresses objective **O2.2**) and section 5.3 presents contributions **C3** and **C4**, which address objective **O2.1**.

5.1 Ontology Change Representation

As presented in section 2.2.1.1, *ontology change representation* is one of the tasks identified by most of the ontology evolution approaches and refers to the representation of a request for a change formally and explicitly as one or more ontology changes ([Sto04]). Although we can find several approaches for the representation of changes, they have still some limitations (as we discussed in section 2.2.4). In particular, they are dependent on the underlying ontology model, and they are

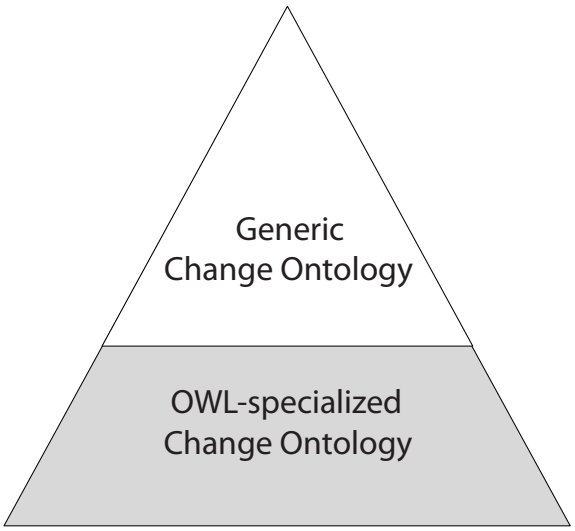


Figure 5.1: Change Representation Layered Approach for OWL Ontology Model

considering changes at the entity level (concepts, properties, individuals).

Hence, we propose a **layered model for the representation of ontology changes** that integrates many of the features of existing change ontologies. This model consists of a generic change ontology, **independent of the underlying ontology model**, that models generic operations in a taxonomy of changes that are expected to be supported by any ontology language (based on the ontology components identified by Gruber in [Gru93]). For instance, the operation "Add Class" should be supported by any ontology language. Note that depending on the knowledge representation paradigm, a class might be referred to as a concept, but operations at higher levels of abstraction (e.g., "Move Subtree") might even be referred as the same independent of the knowledge representation paradigm.

Besides, the generic change ontology defines some of its properties with an unconstrained range **to avoid dependencies on a specific ontology language**. Hence, unlike existing approaches, we provide a layered approach (similar to the approach proposed by [EH03] in which our generic change ontology can be specialized for specific ontology languages while still providing a common, independent model for the representation of ontology changes. Figure 5.1 illustrates this idea for the OWL ontology model.

Compared to existing approaches, the generic change ontology proposes a more **fine-grained taxonomy of ontology changes**, i.e., atomic, entity and composite. We argue that even though the elementary (atomic) changes proposed in existing approaches are introduced as operations that cannot be subdivided into smaller operations, in all cases they are considering changes at the entity level (concepts, properties, individuals). Hence, we provide an additional lower level for the type of ontology change, i.e., atomic change, that represents the actual "atomic"

operation that can be performed in a specific ontology model. The **atomic change** in our generic change ontology includes a property (with an unconstrained range) to associate it to the specific atomic elements. A specialization of the generic change ontology can then constrain the range of that property to the specific ontology language atomic elements. For instance, in OWL ontologies, that lower level should be associated to the actual set of possible OWL axioms, while in RDFS ontologies that lower level should be associated to the actual RDF statements (triples). Having this lower level in the classification will provide a direct mapping between the user action and the ontology operations.

The next level in our classification is the entity level, which allows associating ontology changes to ontology elements. Similar to the atomic change, the **entity change** in our generic change ontology includes a property (with an unconstrained range) to associate it to the specific ontology elements. A specialization of the generic change ontology can then constrain the range of that property to the specific ontology language elements. As we can see from the previous discussion, our entity change corresponds to the elementary (atomic) change in the literature (e.g., [Sto04] and [Kle04]) and therefore we can reuse and refine existing proposals. Note that the generic change ontology only provides a list of the entity changes expected to be supported by every ontology language. However, as entity changes can be expressed by one or many atomic changes, the *exhaustive* list of possible entity changes depends on the underlying ontology model and, therefore, it might only be represented in specializations of the generic change ontology.

Finally, similar to previous approaches (e.g., [Sto04] and [KN03]), **composite changes** represent a group of changes applied together that constitute a logical entity, such as move a tree of classes and merge a set of siblings. It is evident, as it has been also noted in the literature, that it is not possible to have an exhaustive list of composite changes, i.e., one can combine entity changes and composite changes in many different ways. Therefore, in our ontology we provide only some of the most common operations.

Besides the taxonomy of ontology changes, **the generic change ontology also models when the change was made, who made it, and how it was made**. This is similar to what is proposed in [Sto04] and refers to the provenance of changes. For instance, to keep track of the actual sequence of changes (the order in which changes were performed), our ontology relies on two elements: each change is linked to its predecessor (as in a linked list) and we keep a pointer to the last change in the ontology history.

Furthermore, the generic change ontology provides the means to support not only the tracking of changes but also the information that identifies the original and the current version of the ontology after applying the changes (critical for the management of ontology versions).

Finally, **the generic change ontology has been implemented as an OMV extension** (see section 4.2) for two reasons: first, as we described in 1.1, we consider ontology changes as a special kind of ontology metadata and second, it relies on

and uses some of the knowledge defined in OMV. For instance, we reuse the `Ontology` class to identify the ontology versions, and we use the `Person` class to identify the author of the change.

In the following we present the generic change ontology and two specializations (for OWL 2 and RDFS). These ontologies are available for download at `http://omv.ontoware.org`.

5.1.1 Generic Change Ontology

The main classes and properties of the generic change ontology are illustrated in Figure 5.2.

The class `Change` is the most important of our ontology. It models the hierarchy of ontology changes, which organizes changes according to their type and includes all changes that are independent of the underlying ontology model. Hence, it has three subclasses: `AtomicChange`, `EntityChange` and `CompositeChange`. The atomic changes are further decomposed into additive changes (`Addition`) and removal changes (`Removal`). Following the approach proposed by [Kle04], we modeled each entity change operation as a class and defined subsumption relations between these classes thus defining a hierarchy of entity operations. For example, all entity changes related to classes are grouped within the class `ClassChange` and, in a similar way, with the other types of ontology elements (e.g., properties and individuals). Note that the classes used to organize the entity operations are abstract classes that should not be instantiated. For the composite changes we provide only a number of classes that represent the most common composite operations. To associate the atomic changes and entity changes to the appropriate atomic elements/operations (e.g., axioms) and entity elements, we use the object properties `appliedAxiom` and `hasRelatedEntity` respectively. As we mentioned, those properties in the generic change ontology do not have any specified range; ranges have to be specified in the specialization. Furthermore, to express that an entity change consists of one or more axiom changes, we associate both classes using the property `consistOfAtomicOperation`, and to express that a composite change is a combination of other changes we define the property `consistOf`. Additionally, to group all the changes made to a particular ontology version (see 5.2.2) we defined the class `changeSpecification` and associated it to the `Ontology` class from the OMV core to specify the previous and current version of the ontology before and after the changes. To specify who made a particular change, we relate the `Change` class with the `Person` class from the OMV core. Also, to keep the track of the actual sequence of changes (the order in which changes where performed), the object property `hasPreviousChange` provides the required link between different changes, and a `Log` class provides the pointer to the last change in the ontology history. Finally, similar to [Sto04] we keep information supporting decision-making, such as cost, relevance and priority. The cost of a change determines the required effort to perform the change (e.g., number of derived operations necessary to complete the requested change).

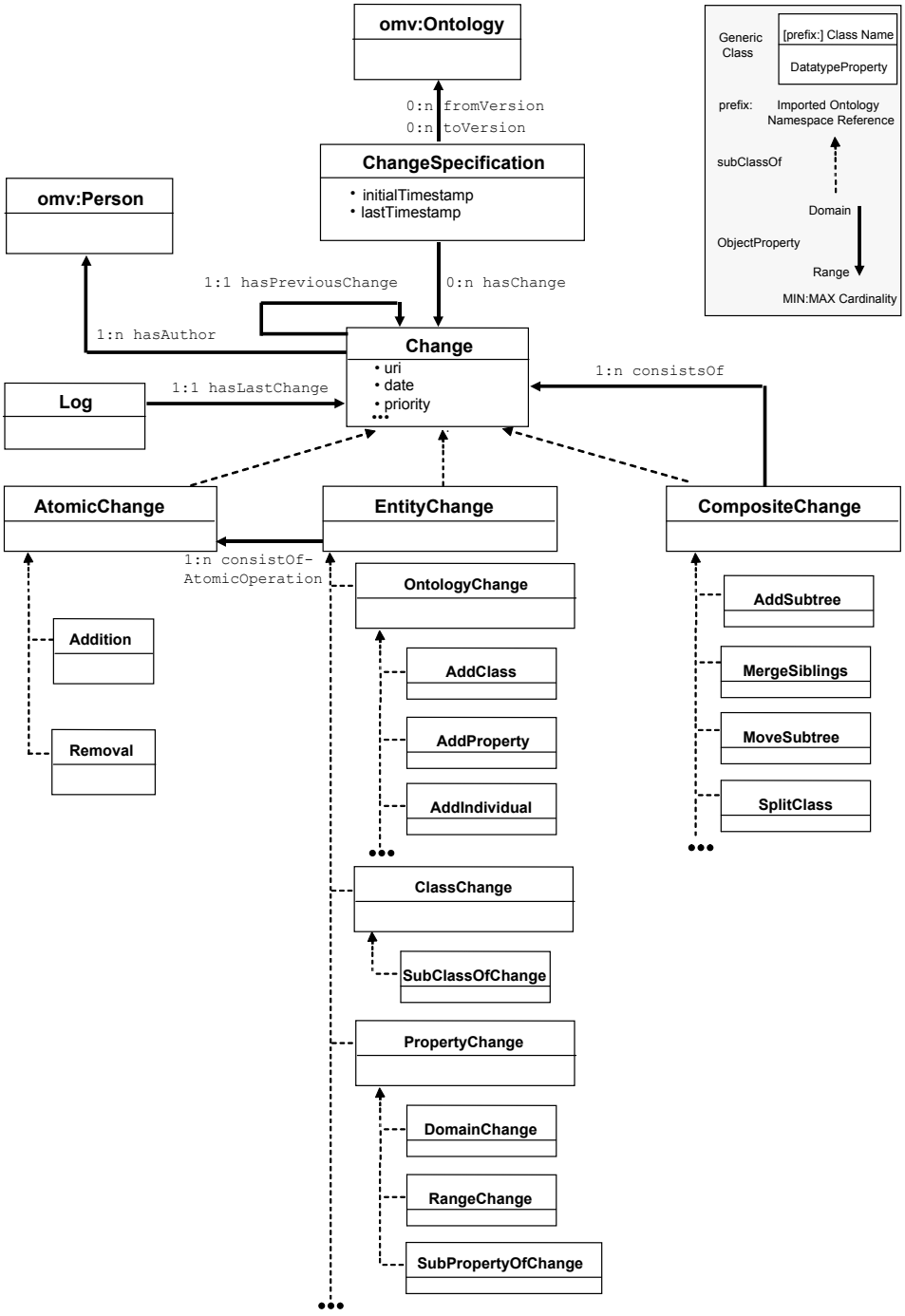


Figure 5.2: Main Classes and Properties of Generic Change Ontology

The relevance describes whether and how the change can fulfil the requirements. Consider for example a class deletion. Its cost is estimated based on the ontology

structure (e.g., the number of the subclasses and the total number of the individuals of these classes), so if the relevance is low and the cost is too high it would be better to avoid to perform that change unless it has a very high priority.

Note that our generic change ontology unifies features of the most relevant approaches but, in contrast to them, it consists of a layered approach and considers a lower granularity level for the classification of changes, i.e., atomic changes representing the unitary operations that can be applied on an ontology. Hence, this verifies hypothesis H7 (cf. 3.4).

5.1.2 OWL 2 Change Ontology Extension

For our OWL 2 extension, we decided to use a metamodel instead of using directly its knowledge representation ontology[1] (KR ontology) to refer to the OWL 2 elements. The reason for using a metamodel is that it facilitates the task of defining the relationship between the elements of the generic change ontology with the elements of the ontology model: a metamodel is a precise definition of the constructs and rules (syntactic and semantic properties) of a modeling language[2]. Hence, a metamodel is the model of the modeling language, which reflects the abstract syntax and semantics of the language.

For instance, in the OWL 2 KR ontology, the *owl:equivalentClass* axiom is a built-in property that links a class description to another class description:

```
<rdf:Property rdf:ID="equivalentClass">
    <rdfs:label>equivalentClass</rdfs:label>
    <rdfs:subPropertyOf rdf:resource=
    "http://www.w3.org/2000/01/rdf-schema#subClassOf"/>
    <rdfs:domain rdf:resource="#Class"/>
    <rdfs:range rdf:resource="#Class"/>
</rdf:Property>
```

There is no explicit information that this construct is a type of axiom (in particular a class axiom), and that it requires at least two arguments (two class descriptions). In contrast, in the OWL 2 metamodel all this information is captured by the model as follows:

```
<owl:Class rdf:ID="EquivalentClasses">
    <rdfs:subClassOf rdf:resource="#ClassAxiom"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#equivalentClasses"/>
            <owl:minCardinality rdf:datatype=
            "&xsd;nonNegativeInteger">2</owl:minCardinality>
            <owl:onClass rdf:resource="#Description"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="ClassAxiom">
```

[1] http://www.w3.org/2002/07/owl

[2] Definition adapted from http://www.metamodel.com/staticpages/index.php?page=20021010231056977

```
    <rdfs:subClassOf rdf:resource="#Axiom"/>
</owl:Class>
```

The explicit definition of the taxonomy of the language constructs and their properties allows describing changes related to similar OWL 2 elements (e.g., axioms and entities) in an efficient manner.

In our implementation, we rely on the OWL 2 metamodel specified as part of the Networked Ontology Model[3] in [HBP+07].

The Networked Ontology Model is defined using a metamodeling approach based on MOF (Metaobject Facility [OMG06]). The metamodel consists of individual modules for the individual aspects of networked ontologies. The main modules are (1) a metamodel for the OWL 2 ontology language, (2) a rule metamodel, (3) a metamodel for ontology mappings, and (4) a metamodel for modular ontologies. The metamodel is grounded by translations to specific logical formalisms that define the semantics of the networked ontology model.

Yet, for the representation of changes in OWL 2 ontologies, we focus on the first part, i.e., the OWL 2 metamodel.

5.1.2.1 Overview of the OWL 2 Ontology Model

The OWL 2 metamodel has a one-to-one mapping to the functional-style syntax of OWL 2 and thereby to its formal semantics. We will not provide a full specification of the OWL 2 metamodel in this section, we restrict instead to a presentation of the main concepts needed to understand the change representation.

Note that the OWL 2 metamodel is based on the structural specification and functional-style syntax of the OWL 2 Web Ontology Language in the W3C Working Draft 11 April 2008[4].

Ontologies and Axioms An OWL ontology is defined by a set of axioms, of which OWL 2 provides six different main types as shown in Figure 5.3.

An ontology has an ontology URI that defines it uniquely plus a (possibly empty) set of imported ontologies, and a set of annotations. Additionally, the ontology's elements and the axioms can be annotated.

Figure 5.4 shows the OWL 2 metamodel presentation for ontologies, its axioms and annotations as metaclasses. The association *ontologyAxiom* connects the ontology to its axioms, whereas the associations *ontologyAnnotation* and *axiomAnnotation* connect ontologies and axioms respectively to their annotations. The class *Ontology* has an attribute to identify the ontology, *URI*, whereas the attribute *URI* of the class *Annotation* specifies the type of annotation.

[3] Note that any other OWL2 metamodel could have been used, since our OWL 2 extension is not dependent on any specific OWL 2 metamodel

[4] http://www.w3.org/TR/2008/WD-owl2-syntax-20080411/

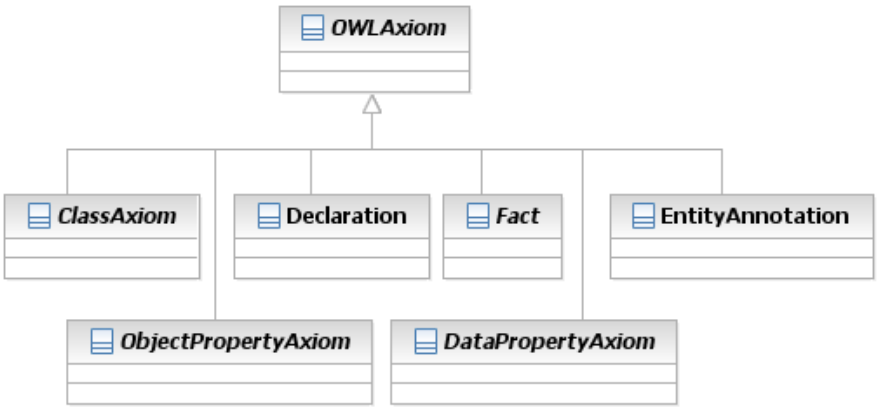


Figure 5.3: OWL 2 Axioms ([HBP+07])

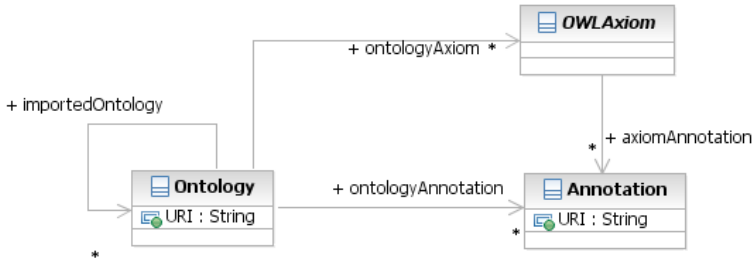


Figure 5.4: OWL 2 metamodel: ontologies ([HBP+07])

Entities Entities are the fundamental building blocks of OWL 2 ontologies. OWL 2 has five entity types: *data types*, *OWL classes*[5], *individuals*, *object properties* and *data properties*. A datatype is the simplest type of data range. The second entity, a class, is a simple axiomatic class description classifying a set of instances. These class instances are called individuals and are also classified as OWL entities. At last, an object property connects an individual (belonging to a class) to another individual, whereas a data property connects an individual to a data value (belonging to a data range).

The OWL 2 specifications highlight entities as the main building blocks of an OWL ontology and its axioms. Hence, the OWL 2 metamodel defines them as first-class objects in the form of metaclasses. Figure 5.5 presents an abstract metaclass *OWLEntity* which is defined as supertype of all types of entities. The five specific types of entities are specified as subtypes of *OWLEntity*: *Datatype*,

[5]OWL provides two classes with predefined URI and semantics: owl:Thing defines the set of all objects (top concept), whereas owl: Nothing defines the empty set of objects (bottom concept).

OWLClass, *ObjectProperty*, *DataProperty* and *Individual*. An attribute *URI* of the abstract metaclass *OWLEntity* is inherited by all subclasses to identify the entity.

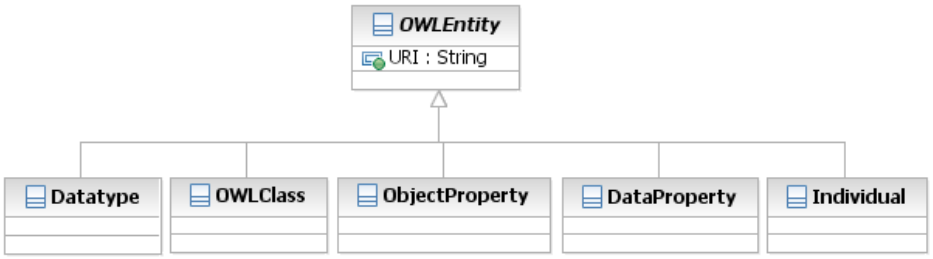


Figure 5.5: OWL 2 metamodel: entities ([HBP+07])

Just like ontologies and axioms, entities can be also annotated. OWL 2 categorizes such entity annotations as axioms. Hence an entity can be involved in two types of annotation: an annotation of the entity itself, or an annotation of such entity annotation as an axiom.

Descriptions Finally, OWL 2 provides an expressive language for forming descriptions. A *description* is an abstract superclass for all class definition constructs, and it is specialized into the following 18 different types (an *OWLClass* is both a type of entity and a type of description). We refer the reader to the W3C Working Draft of the structural specification and functional-style syntax of the OWL 2 Web Ontology Language[6] for additional information.

- owlClass
- objectUnionOf
- objectIntersectionOf
- objectComplementOf
- objectOneOf
- objectAllValuesFrom
- objectSomeValuesFrom
- objectExistsSelf
- objectHasValue
- objectMinCardinality
- objectMaxCardinality
- objectExactCardinality
- dataAllValuesFrom
- dataSomeValuesFrom
- dataHasValue

[6] http://www.w3.org/TR/2008/WD-owl2-syntax-20080411/

- dataMinCardinality
- dataMaxCardinality
- dataExactCardinality

5.1.2.2 OWL 2 Change Ontology

To extend the generic change ontology to build the OWL 2 change ontology, we had to perform two main tasks: first, we specified the range of the unconstrained object properties `appliedAxiom` and `hasRelatedEntity`. That is, we associated the `AtomicChange` and `EntityChange` classes from our generic change ontology with the *OWLAxiom* class and the *OWLEntity* class from the OWL 2 metamodel. Figure 5.6 illustrates the main classes and properties of the change ontology for OWL 2. Note that since the OWL 2 metamodel was previously implemented as an ontology named *OWL-ODM*[7], we use the prefix *owlodm* in the figure to refer to elements of that ontology.

Hence, following the above description of the OWL 2 metamodel, there are six main types of axioms. Four out of the six main axiom types (*classAxiom*, *objectPropertyAxiom*, *dataPropertyAxiom* and *fact*) are further specialized into subtypes: 4 class axioms, 13 objectProperty axioms, 6 dataProperty axioms and 7 fact axioms. For a complete description of all axioms we refer again the reader to the OWL 2 W3C Working Draft[8].

The two main axiom types not specialized (*declaration* and *entityAnnotation*) plus all the axiom subtypes represent the possible atomic operations that can be performed over an OWL 2 ontology, i.e., add/remove axiom. Consequently, we have 32 different types of atomic operations.

Second, the taxonomy of entity-level changes has been extended to model the particular changes for the OWL 2 ontology language. Hence, the extended taxonomy includes changes for OWL elements such as objectProperties (e.g., add/remove EquivalentObjectProperties and functionalObjectProperty) or dataProperties (e.g., add/remove disjointDataProperties and functionalDataProperty) among others. Note that the composite-level changes were not extended as they represent composite operations that are expected to be supported in any ontology representation language (such as move element or remove tree) and, therefore, they are modeled in the generic change ontology. Figure 5.7 illustrates part of the taxonomy of entity changes that is specialized for the OWL 2 ontology model. Note that this figure provides some specific examples of change types, whereas Figure 5.6 shows only the top-level change types.

[7] `http://owlodm.ontoware.org/`
[8] `http://www.w3.org/TR/2008/WD-owl2-syntax-20080411/`

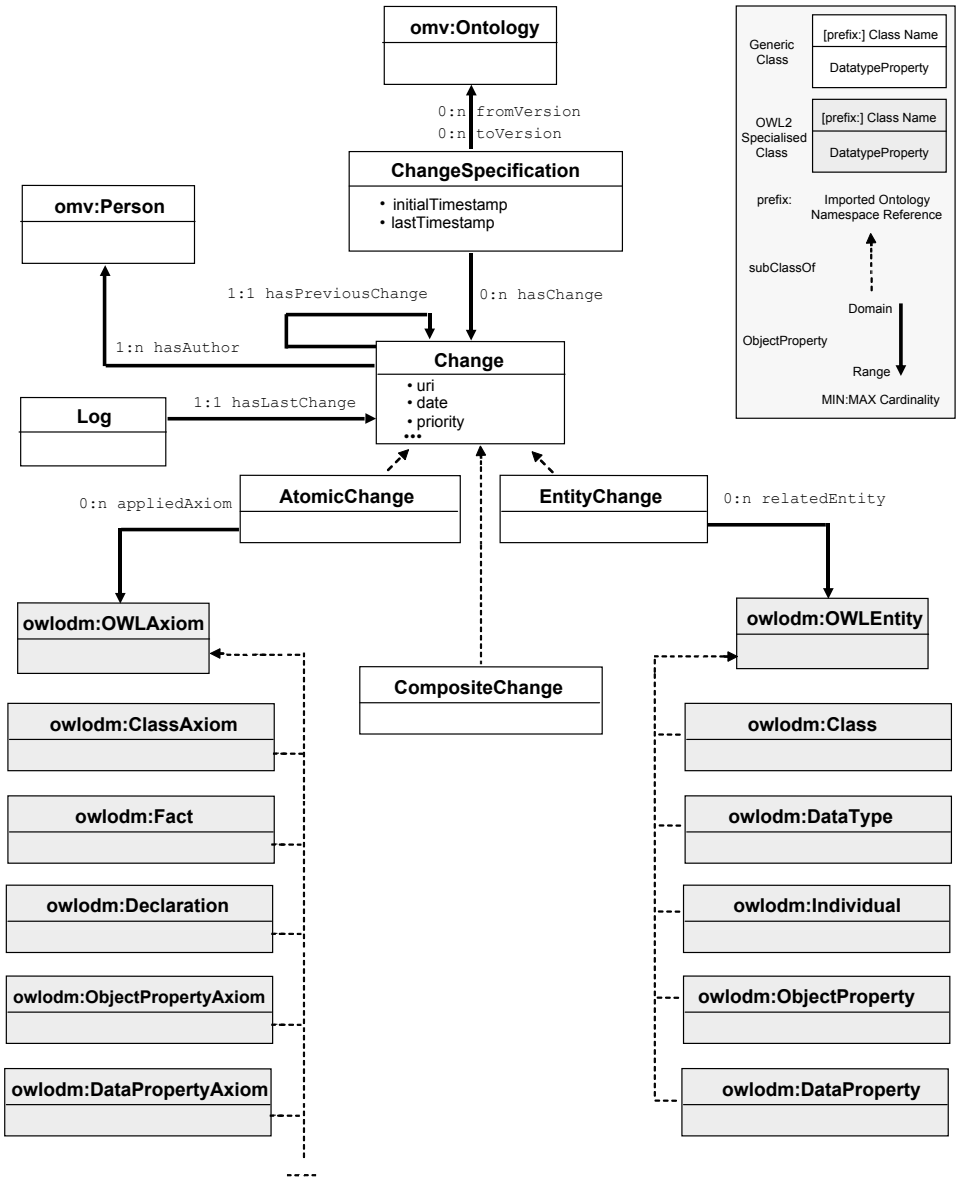


Figure 5.6: Main Classes and Properties of OWL 2 Change Ontology

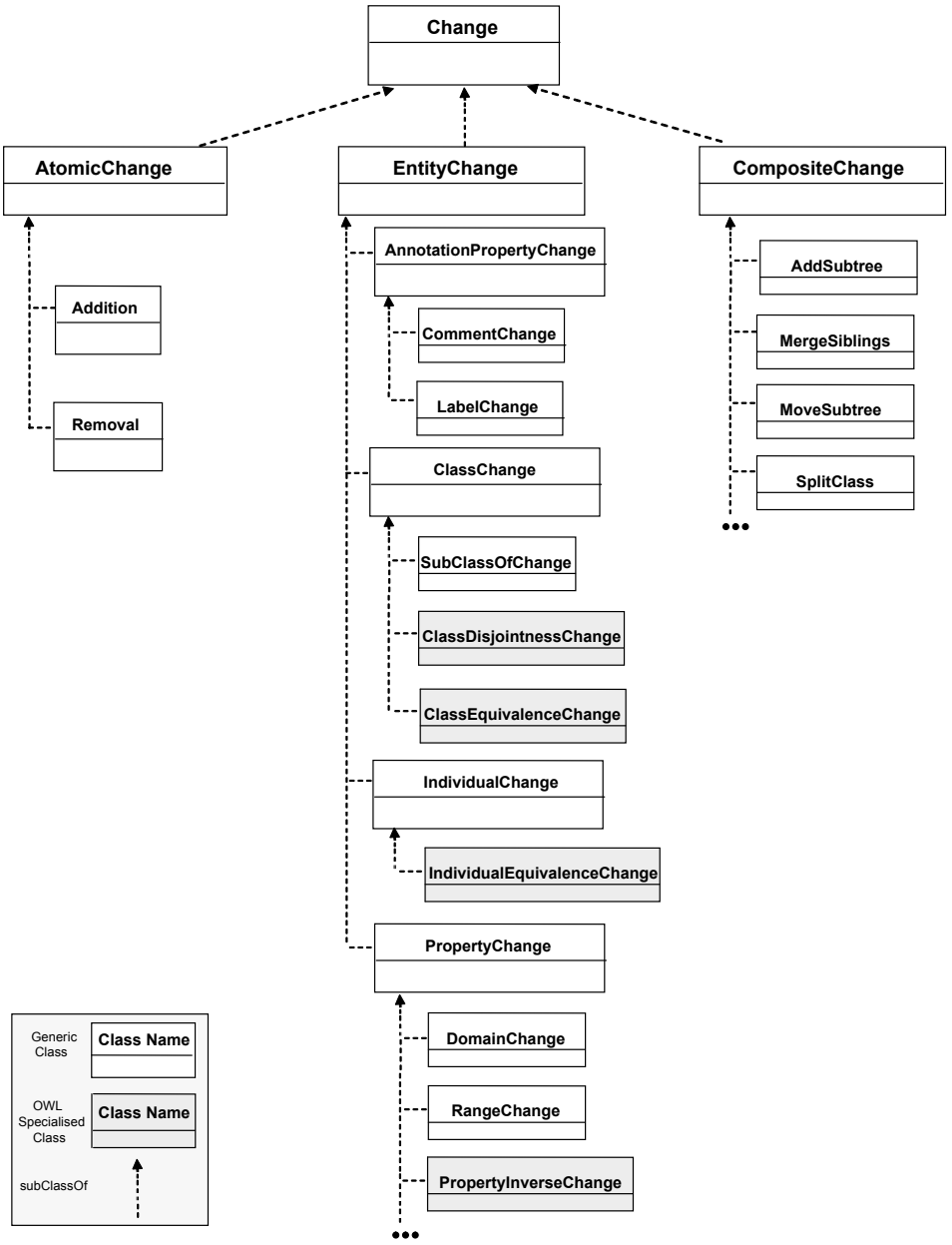


Figure 5.7: Excerpt of the taxonomy of changes for both the generic change ontology and the OWL 2 specialization

5.1.3 RDFS Change Ontology Extension

In order to prove our hypothesis H6 (cf. 3.4), we briefly present here another extension of the generic change ontology for the RDFS ontology language. For this extension, we rely on the RDF metamodel defined as part of the OMG adopted

specification[9] for the Ontology Definition Metamodel (ODM) (see [OMG08]). The reason for using a metamodel, instead of using directly its KR ontology[10] is the same as the one we explained for the OWL 2 extension.

The ODM is defined as a family of MOF metamodels, mappings between those metamodels and mappings to and from UML, and a set of profiles that enable ontology modeling through the use of UML-based tools. The metamodels that comprise the ODM reflect the abstract syntax of several standard knowledge representation and conceptual modeling languages that have either been recently adopted by other international standards bodies (e.g., RDF and OWL by the W3C), are in the process of being adopted (e.g., Common Logic and Topic Maps by the ISO), or are considered industry de facto standards (non-normative ER and DL appendices).

The RDF metamodel consists of three individual metamodels: RDFBase, RDF Schema (RDFS) and RDFWeb. The RDFBase package reflects core concepts required by virtually all RDF applications. The RDFS metamodel includes the concepts defined in the RDFBase package and extends them to support the vocabulary language defined in the RDF Schema specification. The RDFWeb package includes additional concepts that are not specific to RDF but that define web document content.

Hence, for our purpose, i.e., the representation of changes in RDFS ontologies, we focus on the first two packages, namely, RDFBase and RDF Schema (RDFS).

5.1.3.1 Overview of the RDFS Ontology Model

Similar to its OWL 2 counterpart, in this section we will not provide a full specification of the metamodels, instead we restrict to a description of the main concepts needed to understand the change representation.

Ontologies, Axioms and Entities An RDFS ontology can be seen as a set of triples. In fact, in the RDFWeb Package an RDF document is associated to the set of triples it contains. A triple contains a subject node, a predicate and an object node. Nodes are RDF URI references, RDF literals or blank nodes. Each triple represents a statement of a relationship between the things denoted by the nodes that it links. The assertion of an RDF triple says that some relationship, indicated by the predicate, holds between the things denoted by subject and object of the triple. In the RDF metamodel, the triples are defined as first-class objects in the form of the metaclass *RDFTriple*. In our context, a triple represents the atomic unit that can be added/removed to/from an RDFS ontology.

Additionally, according to the RDF metamodel, all things described by RDF are called resources. This is the class of everything. All other classes are subclasses of this class. An RDF resource has 7 first level subclasses: *RDFSClass*, *RDFProperty*, *RDFSContainer*, *RDFList*, *RDFStatement*, *Node* and *NamedGraph*. Figure

[9] Note that any other RDFS metamodel could have been used, since our RDFS extension is not dependent on any specific RDFS metamodel

[10] `http://www.w3.org/2000/01/rdf-schema`

5.8 presents the taxonomy for RDFSResource. Hence, in our context, the RDFS entities are RDF resources. However, it is worth mention that the last three types of resources (RDFStatement, Node and NamedGraph) were particularly interesting for our purpose (represent changes in RDFS ontologies). An RDFStatement provides a way to make statements about triples or describe statements without asserting them. Hence an RDF statement is associated to the triple it reifies, if such a triple exists (association *ReificationForTriple*). The Node represents the subject or object of a triple. Finally, the NamedGraph is a uri reference and RDF graph pair. It provides a way to name an RDF graph; however, NamedGraphs are not a part of RDF and hence ODM tools are not required to support this element, and they may change in future revisions. In the next section we describe our RDFS extension for the generic change ontology and explain how we have dealt with the previous resources.

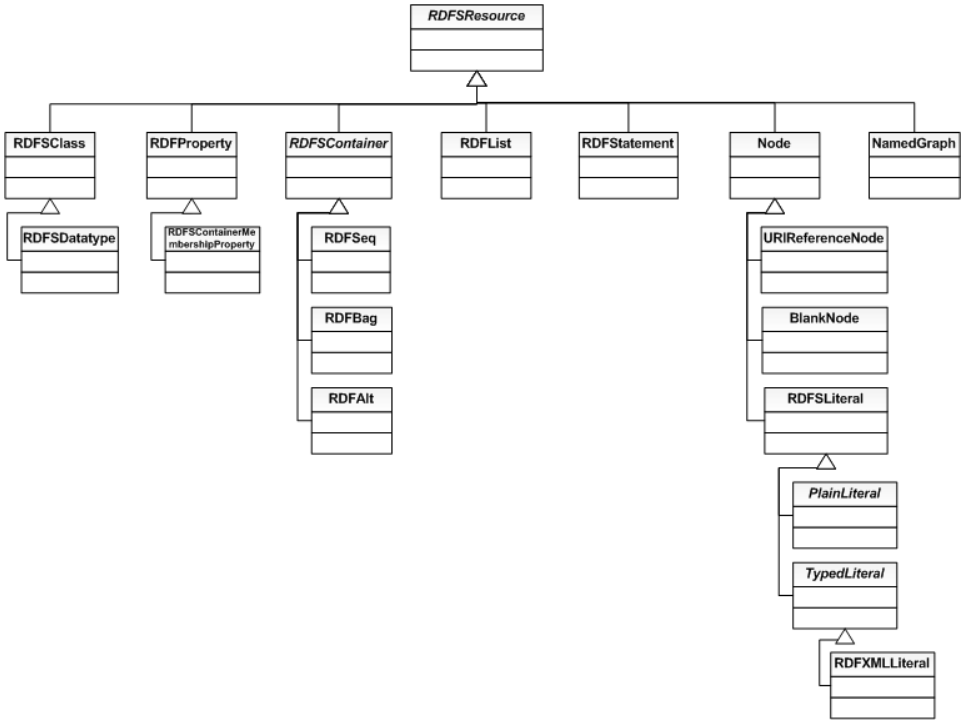


Figure 5.8: RDFS metamodel: RDFResource ([OMG08])

5.1.3.2 RDFS Change Ontology

Similar to the previous extension, we had to perform two main tasks to extend the generic change ontology: first, we specified the range of the unconstrained object properties `appliedAxiom` and `hasRelatedEntity`. That is, we associated the `AtomicChange` and `EntityChange` classes from our generic change on-

tology with the *RDFTriple* class and the *RDFSResource* class from the RDF metamodel. Figure 5.9 illustrates the main classes and properties of the change ontology for RDFS.

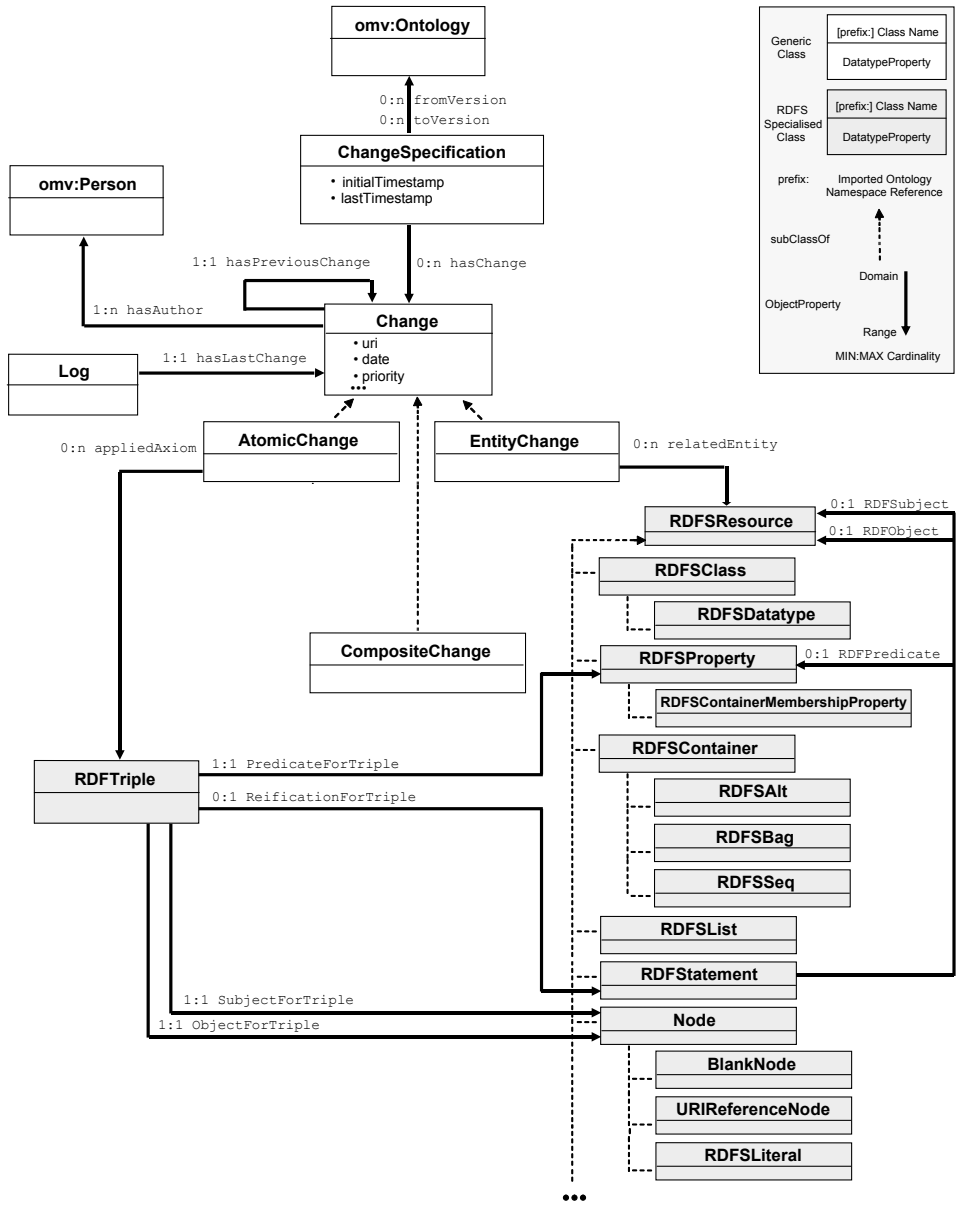


Figure 5.9: Main Classes and Properties of RDFS Change Ontology

Hence, as we mentioned above, the atomic operations that can be performed over an RDFS ontology are those of adding or removing triples.

Second, the taxonomy of entity-level changes has been extended to model the particular changes for the RDFS ontology language. Hence, the extended taxon-

omy includes changes for RDFS resources such as ontology changes (add/remove RDFSContainer, RDFList, etc.), list changes (add/remove firstOf, restOf, etc.) or utility changes (add/remove isDefinedBy, seeAlso, etc.) among others. Regarding the special resources mentioned above (RDFStatement, Node and NamedGraph), we made the following decisions:

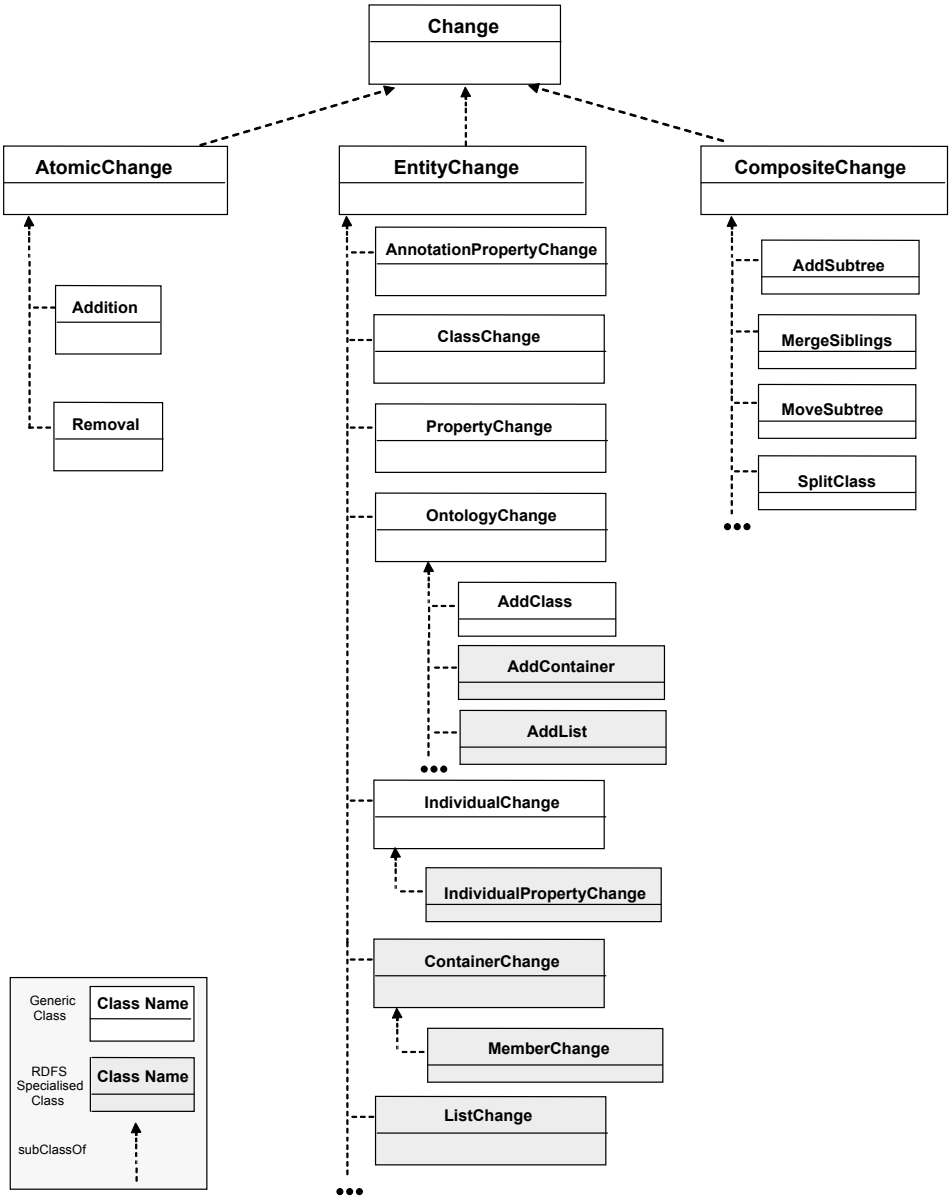


Figure 5.10: Excerpt of the taxonomy of changes for both the generic change ontology and the RDFS specialization

- We do not include changes of RDF statements at the entity level since they are mostly useful for reification of triples and we want to keep separated the atomic level changes (triples) from entity changes.
- We treat nodes in the same way as other entities because the class Node groups different types of entities (e.g., Literals and blank nodes).
- We do not support NamedGraphs as they are not even part of RDF.

Finally, note that the composite-level changes were not extended as they represent composite operations that are expected to be supported in any ontology representation language (such as move element, or remove tree) and, therefore, they are modeled in the generic change ontology.

Figure 5.10 illustrates part of the taxonomy of entity changes that is specialized for the RDFS ontology model. Note that this figure provides some specific examples of change types, whereas Figure 5.9 shows only the top-level change types.

5.2 Ontology Change Manipulation

The manipulation of changes includes supplementary methods and strategies that support the complete management of changes in distributed environments. Next, we discuss the capturing of ontology changes, including their monitoring, processing, logging and storing, and the management of different ontology versions.

5.2.1 Ontology Change Capturing

Changes in ontologies need to be captured and stored in a certain format. The definition of the change ontology presented in the previous section, allows storing changes about a certain ontology in a machine-understandable format. However, there should be appropriate methods to capture the changes in the evolving ontology and store them in an appropriate manner.

In a controlled scenario where several ontology engineers are working collaboratively on a set of ontologies, the editing activities are performed directly using the ontology editor interface. Thus, the process of capturing changes can be described in the following steps: first, a change in an ontology from the ontology editor (step 1), should fire an ontology change monitor (step 2). Then, in step 3, the monitor calls an ontology change processing component, responsible to collect all the information about the change (e.g., author of the change, time of the change and type of change). Next, in step 4, the collected information is passed to the ontology change encoder where the change is represented according to the change ontology by creating the appropriate individual(s). Finally, the change individual(s) is stored in an appropriate place for future processing and propagation by the ontology change logger (step 5). This procedure is depicted in Figure 5.11.

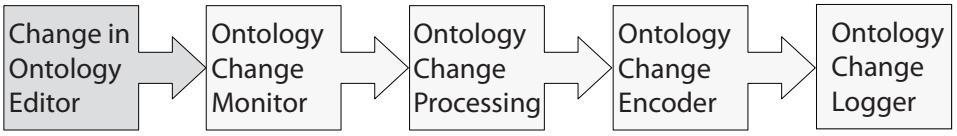


Figure 5.11: Procedure of Change Capturing

Note that the task of storing a change individual involves additional subtasks, such as updating the information of the chronological order of changes and the propagation of changes to related entities. In our approach, we propose that such tasks should be carried out by an ontology registry system.

5.2.2 Ontology Version Management

An important aspect that has to be taken into consideration when dealing with ontology changes is the management of ontology versions. As we can see from section 2.2.1.2, there are different definitions and understandings of what ontology versioning is. So, for this thesis, we consider ontology versioning as follows: it is the mechanism in charge of the ontology change management, which includes facilities for keeping track of ontology changes and the ability to identify and maintain different variants of ontologies and their dependencies, with support to undo/redo operations (e.g., rollback to a previous variant).

To keep track of ontology changes we generate a log that maintains the history (and order) of applied changes as a sequence of individuals of our proposed change ontology (see previous section). According to the classification of versioning given in [LM07], our approach is known as change(operation)-based versioning because we treat changes as first-class entities. The management of that ontology meta-information (applied changes) is the responsibility of the ontology registry, which is in charge of the administration of all the ontology related metadata (see below in section 5.2.3). Furthermore, the information about the precise changes performed can be used to easily compute the difference between variants or to implement a multiple undo/redo mechanism.

Furthermore, as described in 2.2.1.2, the issue of identifying the different variants (versions) of an ontology is not a trivial one. For instance, even the OWL ontology language itself did not provide the means to identify and manage multiple versions and physical representations of one ontology until the latest release of OWL 2, where this is partially addressed. So, even though ontologies are supposed to be identified by a URI (as OWL 2 has just been released), in practice different versions of an ontology carry the same logical URI. Also as we noted before, typically ontologies either do not provide any version information at all or the ontology editors explicitly do not want to change the version of the ontology after making some changes.

However, since we are considering a controlled environment (typical within

an organizational setting) where we are logging changes on the ontologies as they occur, we can assume that we will always have the version information. Hence, in our solution we rely on the identification of ontologies that we proposed in the specification of OMV (see section 4.2) which consists of a tripartite identifier: the ontology URI, the ontology version (if present), and the ontology location. Additionally, we also rely on OMV for representing the relationship between the different ontology versions (e.g., prior version and compatible with). Finally, the specific set of changes from one ontology version to the next one is captured by the change representation model (described in 5.1) and maintained by the distributed registry with the rest of the ontology related metadata.

Therefore, in our scenario, after a first version of an ontology is obtained, this will normally enter into a maintenance phase in which it could be modified for several reasons (see section 4.2.5). This phase normally follows a well-defined process for the coordination of changes (e.g., who can propose/approve changes or when, depending on the state of the ontology) and the approval of a new ontology release (e.g., a whole new version or an update of the current version). Hence, we propose the following approach for the management of ontology versions:

- The first stable version of an ontology (the ontology that satisfies all requirements) becomes version 1. The version 1 of the ontology is considered an approved ontology (which in some cases is also published in the Internet).

- After an ontology has been approved or published, the first change to that version will automatically create a new version, with a different version information (*N+1*), unless the editor explicitly specifies that the modified version will not become a new version; in this case, the modified version will keep the same version information (*N*). In any case, this (unapproved) version (the new one *N+1* or the current modified one *N*) can receive many changes (by many ontology editors) without changing the version information until it becomes approved/published.

- In case an authorized editor wants to publish an approved ontology version, s/he can decide to keep the version information (do not change it) or to specify a new version information. In the former case, the version information will be the one chosen when the ontology was first modified (see above). In the latter case, the version information can be anything the user specifies (e.g., using a company versioning schema).

5.2.3 Ontology Registry for Change Management

As we already mentioned before, the storage and maintenance of the change information is the responsibility of an ontology registry. The registry stores and manage ontology metadata information which includes information about changes. Hence, the registry should

- Keep the change history for different ontologies, i.e., change log. The change history of an ontology consists of a set of individuals of our generic change ontology and/or any of its specializations, each representing a change applied to the ontology.
- Maintain the chronological order of changes. The set of individuals should be maintained in a linked list, where each change individual is linked to the previous one. Hence, when a new change is registered, it will become the last change in the list by pointing its previous change to the current last change in the list.
- Support different versions of ontologies at different locations. The registry should be able to provide the information that describes not only the different versions but also the specific changes that were applied from one version of the ontology to the next one.
- Provide access and advanced search capabilities to retrieve changes. Particularly, all of these capabilities should be available programmatically to facilitate its integration with other systems.
- Propagate changes to support, for instance, the synchronization of distributed copies of ontologies or the update of ontology related entities (e.g., metadata). For example, the registry should implement the strategies described in next section.

5.3 Change Propagation

As described in section 2.2.1.1, *ontology change propagation* is another of the tasks identified by most of the ontology evolution approaches and refers to the activity of updating all of the ontology dependent artifacts ([Sto04]). Existing approaches for the propagation of changes consider the propagation to related ontologies, to individuals and, in less detail, to dependent applications. As we discussed in section 2.2.4, there are several limitations in this area. For instance, the propagation of changes to ontology related metadata has not been considered. Besides, for the propagation of changes to related ontologies, existing approaches consider only a central (main) copy of the ontology that is either replicated or divided into several component ontologies and, in general, changes are propagated only in one direction: from the main copy to its replicas. The only exception occurs when the ontology is divided into several component ontologies, but in this case the management of changes is centralized.

Hence, we consider two different kinds of change propagation in our distributed scenario: (i) the propagation of changes to distributed ontology copies, which supports a distributed management of changes and the propagation of changes from any copy of the ontology to other copies, and (ii) the propagation of changes to ontology related metadata.

5.3.1 Change Propagation to Distributed Ontology Copies

One of the goals of propagating ontology changes in a distributed scenario is to keep distributed copies of the same ontology synchronized. In order to propagate changes, first we need to have those changes stored in an appropriate system. In the previous section, we presented our solution for the capturing of ontology changes that takes care of (i) monitoring the changes in the ontology editor; (ii) processing and (iii) encoding these changes by means of the change representation model (individuals of the generic/specialized change ontology); and (iv) storing them in an appropriate system (e.g., *ontology registry*).

Once we have the required changes in a machine-understandable format, the system should propagate them to the distributed copies of the ontology. In other words, changes should be propagated to each node in the distributed network that maintains a copy of the ontology (and wants to receive those changes). There are two possible approaches for the propagation: push or pull. The benefits and disadvantages of both approaches have already been analyzed (e.g., [Sto04]). Since we are considering a distributed environment where we cannot guarantee the availability of nodes, we propose a combination of a pull and push mechanisms that we call *synchronization*. The synchronization is based on the time when the changes were originally applied (timestamp). Consequently, it is important that the distributed nodes have their system times synchronized (e.g., with a time server) and that the timestamps have a high precision (to avoid many changes occurring at the same time).

For the *pull mechanism*, we propose that, periodically, nodes will contact other nodes in the network to exchange updated information. For this task, in addition to the changes themselves, i.e., individuals of the change ontology in chronological order, each node maintains in its local log explicit information about (i) which ontologies the node is tracking, (ii) the last change it knows for each of the tracked ontologies and (iii) the set of changes URIs applied to the same ontology (version). Note that, if the local node does not has yet any change information about a tracked ontology, the last change known by the local node for that ontology is equivalent to null, and the set of changes URIs applied to that ontology is an empty set.

During the pull mechanism, the following rules apply: (i) nodes can only update their local registry (log), and (ii) nodes can pull changes applied to ontologies from any remote node. The reason for (i) is that each node can only update the information it owns. Besides, as each node performs the same pull mechanism periodically, if the contacted remote node has an outdated information, eventually it will contact a node with updated information and modify its own information accordingly.

So, the process that takes place when node x contacts node z is as follows:

For each ontology that is tracked in both nodes x and z, the first step is to compute the difference between the set of changes URIs applied to the ontology (e.g., O_i) in remote node z and the corresponding set in local node x. If the difference is the empty set, the set in x is either (i) **equal** or (ii) a **superset** of the corresponding

group in z. If the difference is not the empty set, at least one of the changes in z is not in x, i.e., (iii) x is **not synchronized** with z[11].

Case (i) indicates that nodes are synchronized with each other and, therefore, the local node x does not have to do anything.

In case (ii), the local node x knows about all the changes that the remote node z knows, but additionally it also knows about other changes. Hence, following the rules described above, x does not have to do anything.

In case (iii), the remote node z knows about one or more changes that the local node x does not know about. Hence, the local node x retrieves from z all the unknown changes (*pull propagation*), i.e., the changes corresponding to the URIs of the computed difference between z and x. Then, a local conflict detection mechanism should determine whether there is any change that is conflicting with the current changes. In case there are conflicting changes, a conflict resolution mechanism should resolve them. In the simplest case, this mechanism should remove the conflicting changes from the set and notify the presence of the conflict (see below). Note that entity and composite changes should be treated as atomic operations. For instance, in an OWL 2 ontology, the entity change "Add Individual" will be in conflict if any of its two corresponding atomic changes ("Add Declaration" and "Add ClassMember") is in conflict.

Next, for each change C_i, x registers it in its local log. However, unlike the normal logging operation described in Sections 5.2.1 and 5.2.3, C_i is not necessarily appended at the end of the local log. Instead, C_i is placed at the corresponding position in the chronological order. That is, C_i is placed after the last change whose timestamp is equal-to or older than C_i timestamp, before the first change whose timestamp is newer than C_i timestamp. Besides, the pointer to the last change in the local log is not updated unless C_i is actually added at the end of the local log. Note that C_i is registered with its original information (e.g., author, timestamp and state) except from the link to its previous change, which may need to be modified after merging changes from different nodes. You can further observe that the unknown changes can be processed in any order. However, it would be advisable to process them in chronological order, starting with the first (oldest) unknown change, in order to minimize the modifications of the previous change information.

Hence, for each change, x performs the following tasks: first, it determines the corresponding previous change (PC_{C_i}) of C_i in the local log (the last change whose timestamp is equal-to or older than C_i timestamp). Thus, the following situations can occur:

- PC_{C_i} = null.
- PC_{C_i} = last change in the local log (LC_{O_i}).
- $PC_{C_i} \neq$ null and $PC_{C_i} \neq LC_{O_i}$.

[11] $x \cap z = \emptyset$ or $x \cap z = x$ (x is a subset of z) or $x \cap z =$ subset of x and subset of z

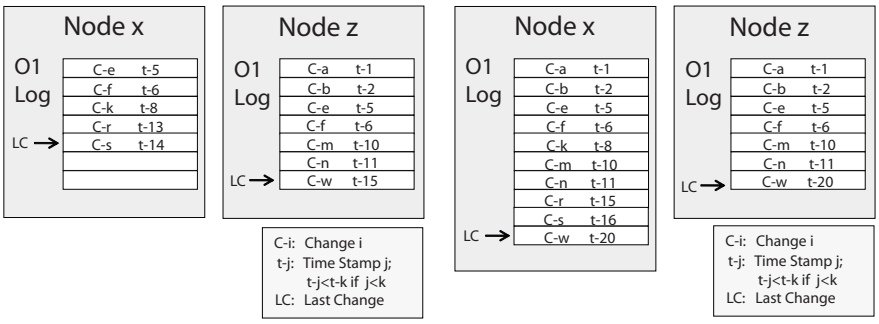
Second, x identifies the current next change of PC_{C_i} ($NC_{PC_{C_i}}$) in the local log (the first change whose timestamp is newer than C_i timestamp). Thus, the following special situations can occur:

- If PC_{C_i} = null, $NC_{PC_{C_i}}$ = null if local log for O_i is empty or $NC_{PC_{C_i}}$ = first change in the local log (FC_{O_i}).
- If PC_{C_i} = LC_{O_i}, $NC_{PC_{C_i}}$ = null.

Finally, x adds C_i in the local log between PC_{C_i} and $NC_{PC_{C_i}}$. As a result, C_i will be registered as follows:

- At the *beginning of the local log* if PC_{C_i} = null.
- At the *end of local log* if PC_{C_i} = LC_{O_i}.
- Somewhere in the *middle of the local log* if $PC_{C_i} \neq$ null and $PC_{C_i} \neq LC_{O_i}$.

Note that the last change in the local log is not updated after registering the change except when PC_{C_i} = LC_{O_i} (the change was registered at the end of the log), or when the local log for O_i was empty.



(a) Before synchronization (b) After synchronization

Figure 5.12: Illustration of the Synchronization Process. Logs for ontology O_1 in nodes x and z before and after synchronization, if changes are not conflicting

To illustrate the process described above, Figure 5.12 depicts the logs for ontology O_1 in nodes x and z before and after the synchronization. In the figure, the time when the change was originally applied is denoted by *t-i*; a change with timestamp *t-i* was applied after a change with timestamp *t-j* if $j < i$. As we can see in Figure 5.12(a), before the synchronization, node x has five changes, while node z has seven changes. When x contact z, the synchronization process calculates the difference between the remote log and the local log. In this example, five changes in z are unknown by x and, consequently, x retrieves them from z. Assuming none of these changes is conflicting, then, each of them is registered in the local log at the corresponding position in the chronological order. In Figure 5.12(b), after the

synchronization, the log in x has ten changes. This example illustrates the three possible scenarios described above, i.e., changes placed at the beginning, at the end, or somewhere in the middle of the local log. Finally, note that before the synchronization, both x and z knew about changes *C-e* and *C-f*. For instance, these two changes were originally applied in a third node, and both x and z have previously synchronized with this node.

The mechanisms for the identification and resolution of conflicts are out of the scope of this thesis (see restriction R2 in Section 3.5). However, our strategy allows minimizing conflicts by running the synchronization process periodically in an automatic manner. Besides, the complete description of changes that we keep in the logs can be used in conflict resolution mechanisms to detect conflicts or to propose resolution strategies (e.g., based on the timestamp of the conflicting changes). We refer the reader to [PHWd07] for a discussion on how to deal with some potential conflicts.

The following observations can be drawn: first, note that for this process it is essential to have the appropriate support to retrieve only the required changes instead of the complete list in order to make the process as efficient as possible.

Second, the registration of each change should be processed in a serialized manner. For instance, if a local node applies a new change while registering a change from a remote node, the new change should be queued for later processing.

Third, the periodicity of the synchronization process should be configurable. Imagine, for instance, a situation where an ontology editor does not want to update (for some reason) his local copy of the ontology with changes from other nodes. Having the possibility to configure the periodicity of the process and even to stop it, if necessary, would solve this problem.

Fourth, if for some reason the timestamps of the changes are not available, the process previously described works with the following additional considerations: the local node should process the unknown changes in order, starting with the first (oldest) unknown change. Additionally, each change should be placed in the same location as in the remote log. That is, the previous change of C_i should be the same in the local log as in the remote log (PC_{C_i} = the original value of the object property `hasPreviousChange` of C_i). The reason is that, due to the distributed nature of the environment considered, nodes can synchronize with other nodes in a totally unpredictable manner, i.e., nodes contact other nodes in different order and nodes can leave/enter the network. As a consequence, when a local node contacts a remote node, its local log could have some information not available in the remote log, some information also available in the remote log, and some missing information from the remote log. Hence, after the synchronization, the local log can have changes in a different order, but it will have all the changes of the remote node.

Finally, we propose an optional *push mechanism* during the synchronization process in which nodes push (periodically) their changes to a specific node in the network so that if the node goes offline before other nodes pull the new changes,

the node changes are not lost or unavailable during the node down time. Note that the longer changes from one node are unavailable, the higher the probability of conflicts with changes in other nodes. Hence, pushing changes to a particular node will minimize those problems. The process for pushing changes is the same as the one described for the pull mechanism, except that the roles of the nodes are inverted: the push (remote) node is treated as the local node, and the local node is treated as the remote node in the process described.

5.3.2 Change Propagation to Ontology Metadata

Currently, an ontology should not be considered as an independent and isolated entity but rather as part of a network of complex relationships and dependencies. In our context of ontology changes, a change in an ontology does not only affect the ontology itself but also all its related artifacts (e.g., ontology individuals, dependent ontologies, applications, mappings and metadata).

In the remainder of this section we will focus on one of the aspects that has not been considered yet in the literature (see 2.2.4): the propagation of ontology changes to its related metadata. Moreover, we have considered a distributed scenario where the metadata is not centralized in one registry but distributed across many nodes.

5.3.2.1 Ontology Metadata

Ontology metadata annotations are entities associated with a particular representation of an ontology that comply with a shared interpretation of the metadata (e.g., ontologies, rule sets and controlled lists of terms). For instance, OMV defines the semantic interpretation for the ontology metadata and is represented as an ontology itself. In this case, an ontology metadata annotation is an OMV individual that is associated to the corresponding ontology specification[12] (identified by an URI – plus the version and location – (see Chapter 4)) and that obviously complies with the OMV schema (see Figure 5.13). Following the approach in [CAM+07], we propose to treat such metadata annotations as resources themselves, i.e., first class citizens, with their own lifecycle.

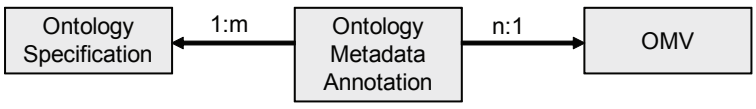


Figure 5.13: Conceptual model for Ontology Metadata Annotation

Ontology metadata is a dynamic entity that can change throughout time. First, ontologies themselves are continuously evolving. If we consider the ontology

[12]The ontology specification refers to the specification of the shared conceptualization

as the formal, explicit specification of a shared conceptualization [SBF98], it is clear that ontologies might change whenever one of the elements of the definition changes (as we introduced in section 1.1). Second, the semantic interpretation for the metadata (e.g., the OMV) might also change. OMV, as we described in Chapter 4, is an ontology itself whose domain are the ontologies (it is meta-ontology) and, therefore, it might change for the same reasons explained before. Third, the metadata itself may change, that is, it may need to be corrected or complemented with additional information. Consider, for example, an automatic tool that extracts and creates metadata, which in turn provides additional information, or a human expert that provides additional information.

In any case, the aforementioned situations may lead to invalid metadata. In the first case, when the annotated ontology changes, it will either become a new version (with a different version information) or, if specified by the editor, it will remain the same version (with the same version information). In the former case, a new metadata annotation should be created, probably using information from the metadata annotation from the previous ontology version (note that in this case the metadata annotation associated to the previous ontology version does not become invalid). In the latter case, the new version should be properly re-annotated with updated metadata. In the second case, when the semantic interpretation of the metadata changes, every metadata annotation should be reevaluated to determine if they are still valid. Finally, changes in the metadata itself (third case), should not invalidate it.

5.3.2.2 Ontology Metadata Lifecycle

The ultimate goal of propagating ontology changes to metadata is to keep metadata up-to-date. Since we are considering ontology metadata annotations as first class entities, this task includes defining an appropriate model for the management of the lifecycle of the ontology metadata annotations, managing the evolution and change of metadata and ontologies in distributed contexts, and synchronizing adequately the evolution of all these related entities by means of notification mechanisms.

Based on [CAM+07] and [MAC+06], we propose a state machine (see Figure 5.14) that defines a set of states and transitions associated to it, for the management of the metadata lifecycle. In particular, we extend and customize the proposed model for two reasons: first, we consider a specific kind of metadata, i.e., ontology metadata. As proposed by the authors, the basic model can be extended by introducing sub-states, resulting in finer-grain definition of the behavior of specific types of metadata. Second, the basic model considers a centralized setting where all the metadata is stored in a central registry. In our scenario, the ontology metadata will be distributed and managed by an appropriate distributed registry.

In the following explanation, the ontology metadata annotation (the OMV individual) is denoted by OI. The ontology annotated by that individual is denoted by AO_{OI} and the OMV schema (the shared interpretation of the metadata) is denoted by OMV. Similarly, the content (payload) of the metadata annotation is denoted

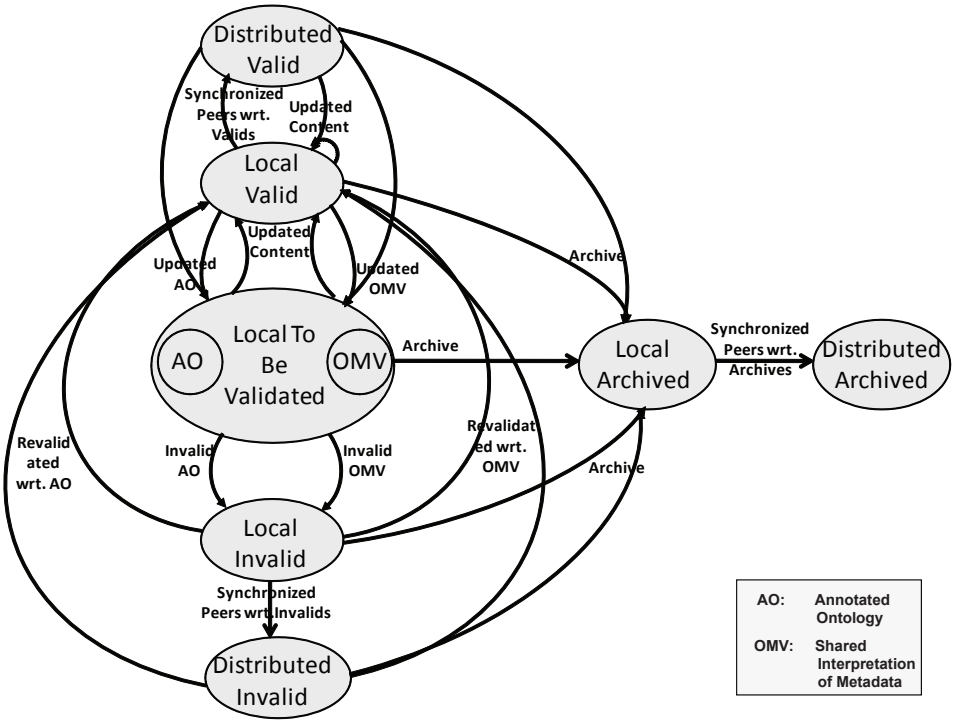


Figure 5.14: Ontology Metadata Annotation Lifecycle

by $content_{OI}$.

Initially, when the ontology metadata annotation is created (an OMV individual), it is in *Local Valid* state because it is originally created in one (any) local node. The possible state transitions events are the following:

- Changes in the described ontologies, i.e., $AO_{OI} \rightarrow AO'_{OI}$
- Changes in the OMV, i.e., $OMV \rightarrow OMV'$
- Changes in the metadata annotation content, i.e., $content_{OI} \rightarrow content'_{OI}$

The first two previous events cause the transition of the ontology metadata annotation to either one of two possible *To Be Validated* state: *To Be Validated AO* or *To Be Validated OMV*. These are transitory states in which the ontology metadata annotation is awaiting to be validated. The validation process is usually an automated procedure that updates (if possible) the contents of the annotation and which results in a decision as to whether the updated ontology metadata annotation represents a new valid annotation, i.e., describes correctly the current version of the annotated ontology and complies with the current OMV schema. Changing the metadata annotation content should not invalidate it. Therefore, this action does not change the state. Next, we explain in detail the two possible transitory states:

- For the *To Be Validated AO*, such procedure determines whether the existing annotation describes correctly the updated version of the annotated ontology, or if the annotation content can be automatically updated accordingly or not. Consider, for example, the approach for the management of ontology versions within an organizational setting, described in section 5.2.2. After the approval of a new ontology release that leads to an update of the current version, we have to verify if the associated metadata still describes correctly the ontology version, or if it can be automatically updated to reflect the changes (e.g., adding a new class to the ontology will lead to update (at least) the *numberOfClasses* property in the metadata annotation from N to N+1).
- For the *To Be Validated OMV*, the procedure should determine whether the new version of OMV can still be used to interpret the old metadata, i.e., it is still a valid OMV individual, or if the annotation content can be automatically updated accordingly or not.

The decision taken in any of the *Local To Be Validated* state can send the metadata annotation back to a *Local Valid* state (when it is still a valid annotation or the automatic process could update it accordingly), or to a *Local Invalid* state (when it is not a valid annotation and the automatic process could not update it accordingly).

When the metadata annotation is in *Local Invalid* state, it could, however, go back to the *Local Valid* state after a manual re-validation process with respect to the annotated ontology or to the OMV schema (or both), depending on which one causes the metadata annotation to become invalid. In the case the annotated ontology changed and the metadata could not be updated accordingly, an ontology editor can manually update the content of the metadata annotation accordingly. Similarly, if the OMV changed and the automated process failed to update the metadata annotation accordingly, an ontology editor can manually re-validate it.

The *Local Archived* state indicates that a metadata annotation is still available for inspection, but it is not active (it is not part of the metadata annotations used by the registry for searching). Usually this state allows to obtain "snapshots" of the metadata annotations available in a certain time.

One rare possible situation is that the ontology becomes obsolete/invalid and, consequently, unavailable/destroyed ($AO_{OI} \rightarrow \emptyset$). In this case, the validation procedure is always assumed to fail, leading to a invalid state (first *Local Invalid* and after synchronization *Distributed Invalid*). From there, the metadata annotation is usually removed from the active set of metadata annotations and sent to be archived ($Archive(OI)$) or destroyed ($OI \rightarrow \emptyset$), for example, when we do not want to save a copy of that information anymore.

The remaining states of our proposed state machine deal with the distributed environment we are addressing. The *Distributed Valid* state indicates that the metadata annotation is valid in every node of the distributed environment that stores that annotation (all nodes are synchronized). For example, when a metadata annotation is created, it is created in one (any) node and becomes *Local Valid*. After nodes are synchronized and the metadata annotation becomes available (and

valid) in all the appropriate nodes, the metadata annotation becomes *Distributed Valid*. From the *Distributed Valid* state, we have the same possible state transitions events as in the *Local Valid* state, i.e., $AO_{OI} \rightarrow AO'_{OI}$, $OMV \rightarrow OMV'$ and $content_{OI} \rightarrow content'_{OI}$. The first two events cause the same transition of the metadata annotation as described above. The only difference is when changing the metadata annotation content, which in this case causes the transition of the metadata annotation to *Local Valid* state. The reason is that even if the content update does not invalidate the metadata annotation, it is performed in one (any) node and this change has to be again synchronized with all corresponding nodes.

The *Distributed Invalid* state indicates that the metadata annotation is invalid in every node of the distributed environment that stores that annotation (all nodes are synchronized). Similar to the previous situation, initially a metadata annotation becomes *Local Invalid* in one (any) node, and only after nodes are synchronized, it will become *Distributed Invalid*. From this state, the metadata annotation could go back to the *Local Valid* state after a manual re-validation process with respect to the annotated ontology or to the OMV schema (or both) as described above. Note that we are going back to the *Local Valid* and not to the *Distributed Valid* state due to the same reasons explained above.

Finally, in a similar way, the *Distributed Archived* state indicates that the metadata annotation is archived in every node of the distributed environment that stores that annotation (all nodes are synchronized). Again, initially a metadata annotation is *Local Archived* in one (any) node, and only after nodes are synchronized, it will become *Distributed Archived*.

Table 5.1 summarizes the events, the state transitions and the validation results (when applicable) of the state machine.

Table 5.1: Events, state transitions and validation actions for the ontology metadata lifecycle state machine

Event	State before event	State after event	State after validation
$AO_{OI} \rightarrow AO'_{OI}$	Local Valid	To Be Validated AO	Local Valid/Local Invalid
$AO_{OI} \rightarrow AO'_{OI}$	Distributed Valid	To Be Validated AO	Local Valid/Local Invalid
$OMV \rightarrow OMV'$	Local Valid	To Be Validated OMV	Local Valid/Local Invalid
$OMV \rightarrow OMV'$	Distributed Valid	To Be Validated OMV	Local Valid/Local Invalid
$content_{OI} \rightarrow content'_{OI}$	Local Valid	Local Valid	N/A
$content_{OI} \rightarrow content'_{OI}$	Distributed Valid	Local Valid	N/A
$AO_{OI} \rightarrow \emptyset$	N/A	To Be Validated AO	Local Invalid
$OI \rightarrow \emptyset$	N/A	N/A	N/A
$Archive(OI)$	N/A	Local Archived	N/A
$Revalidate(OI)$	Local Invalid	Local Valid	N/A
$Revalidate(OI)$	Distributed Invalid	Local Valid	N/A
$Synchronization$	Local Valid	Distributed Valid	N/A
$Synchronization$	Local Invalid	Distributed Invalid	N/A
$Synchronization$	Local Archived	Distributed Archived	N/A

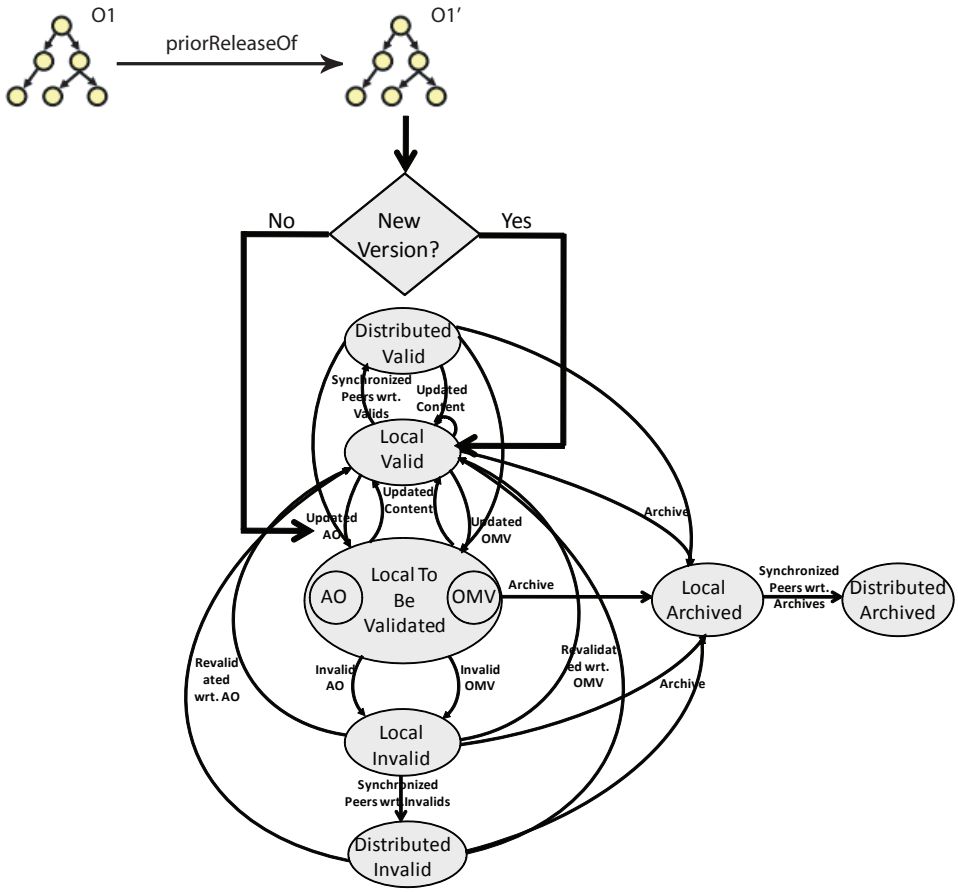


Figure 5.15: Relationship Between Approval of Ontology Release & Metadata Lifecycle

5.3.2.3 Propagation of Ontology Changes to Ontology Metadata

Once we have defined an appropriate model for the management of the lifecycle of the ontology metadata annotations, the next step to keep the annotations updated is the management of the changes in ontologies and metadata in distributed contexts (how those changes are related) and the synchronization of these related entities by means of notification mechanisms. As presented in section 5.2.2, organizations usually follow a well defined process for the coordination of ontology changes and the approval of a new ontology release. Hence, our goal is to clarify how the approval of a new ontology release is related to the lifecycle model for metadata annotations.

Figure 5.15 illustrates such a relationship between the approval of a new ontology release and the state machine described in the previous section (see Figure 5.14).

Whenever an ontology release is approved (or published), either a new version of the ontology is created (the version information is changed from N to N+1) or the same ontology version is updated (the version information remains the same) (see section 5.2.2).

If a new version is created, then a new metadata annotation should be created, i.e., OI_{N+1}. The metadata annotation associated to the previous ontology version (OI_N) remains the same because the corresponding ontology did not change. Note that the OI_{N+1} might be created reusing the information from OI_N. As we explained above, after creating OI_{N+1}, it will be in *Local Valid* state.

When the version information remains the same after some changes in the ontology (event $AO_N \rightarrow AO'_N$), the corresponding metadata annotation (OI_N) should be sent to the *Local To Be Validated* state as described above.

Notification & Synchronization As introduced in the previous section, the changes in an annotated ontology that results in an updated content of the same ontology version ($AO_N \rightarrow AO'_N$ event) may affect its related metadata (OI_N). Consequently, those changes should be propagated to the corresponding metadata annotation.

Since we consider a distributed environment, we propose to decompose the problem in two steps: first, ontology metadata annotations should be notified whenever a change occurs in the corresponding ontology. The notification automatically sends the metadata annotation into the *To Be Validated AO* state where it will be evaluated if it is still a valid annotation, or if it is possible to automatically update it accordingly (see previous section). If is not possible to automatically update the annotation, it will become invalid even though after a manual intervention it might become valid again. Second, the distributed annotations should be synchronized. Independently of the state of the annotation (if the annotation became valid after its content was updated – automatically or manually – or if the annotation became invalid), every node in the network aware of that annotation should have the same information.

For the first step, following the approach in [CAM+07] we propose to use a set of notification mechanisms based on WS-Notification. Therefore, we propose a pre-defined topic associated to ontology changes. In Figure 5.16 we illustrate how the Ontology Metadata Annotation (OI) is subscribed to notifications of change in its corresponding ontology[13] (specified by their last modification time), which will probably make it change its state.

When the OI receives a notification, it will automatically change its state to *Local To Be Validated*, and it will activate a re-validation procedure that will perform the following operations:

- Validate the content of the OI with respect to the changes in the corresponding annotated ontology.

[13]Figure 5.16 also shows how the ontology metadata annotation is subscribed to notifications of changes in the OMV schema

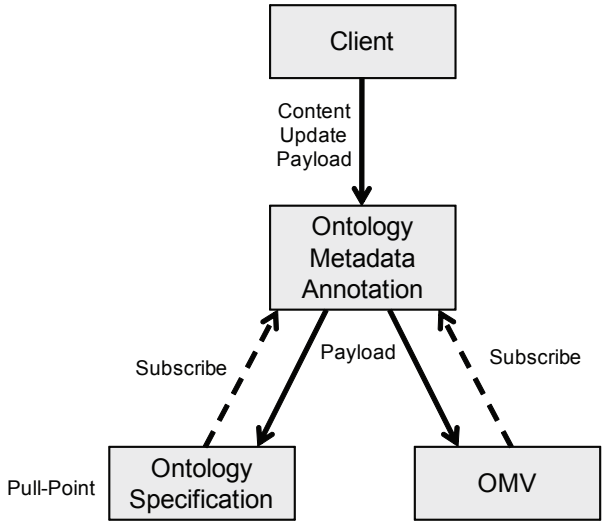


Figure 5.16: Notification of Ontology Metadata Annotations changes

- If validation determines that the OI is valid (the changes in the ontology did not affect it), then the OI is sent to the *Local Valid* state.
- If validation determines that the OI is invalid (the changes in the ontology did affect it), then an automatic process of re-annotation is triggered. After the process finishes, the OI is again validated.
  - If validation determines that the OI is valid (the OI could be updated successfully by the automatic process), then the OI is sent to the *Local Valid* state.
  - If validation determines that the OI is invalid (the OI could not be updated successfully), then the OI is sent to the *Local Invalid* state.

When an OI is in the *Local Invalid* state, a manual re-annotation can still be performed that will make it change its state to *Local Valid* as explained previously.

One additional consideration for the notification mechanism is whether to use a push or pull approach. The benefits and disadvantages of both approaches has been already analyzed in many places (e.g., [Sto04]). Since we consider a distributed environment for the management of ontology metadata annotations, we propose to use the WS-Notification mechanism with a pull-style notification [OAS06].

For the synchronization of the nodes in the distributed environment, we propose that periodically nodes contact other nodes in the network to exchange updated information. For this task, each metadata annotation in the registry should have the information about the last modification time of the associated ontology version. Hence, during the synchronization the following process occurs when node A contacts node B:

- For every OI that is maintained in both A and B. If the metadata annotation maintained in node A associated to ontology X (A_{OI_X})has a lower modification time than the corresponding in B (B_{OI_X}), then update the local annotation. Otherwise do not do anything.

Note that only local annotations can be updated. If the node contacted has an out-of-date information, then we do not do anything as we should not modify remote information. Eventually, the remote node will contact a node with updated information and modify its own information accordingly.

The previous discussion illustrates how the formal representation of the lifecycle of ontology metadata annotations supports the propagation of ontology changes to their related metadata, verifying hypothesis H12 (cf. 3.4).

5.4 Summary

In this chapter we have presented our solution for the management of changes in distributed environments. It addresses some of the limitations identified in the state of art in this area.

First, for the representation of ontology changes (section 5.1), we provide **a layered model that consists of a generic change ontology independent of the underlying ontology language** that can be reused and specialized for the different ontology languages. A language-independent model that can be easily extended fosters its reusability and interoperability between different applications and scenarios. Besides, **our model considers ontology changes at the lowest level of granularity** (the real atomic operations) instead of at the entity level like in existing approaches. This supports a more efficient processing of changes (to implement redo/undo operations, comparison between ontology versions, etc.) and provides a more detailed information of how an ontology changed, as well as the specific consequences of operations at a higher level. This model covers contribution **C2**, which addresses objective **O1.2** (see Chapter 3). Our contribution also includes the development of **two extensions for two different ontology languages (OWL 2 and RDFS)**.

Second, with respect to the propagation of changes, we address two open research problems in the literature: On the one hand, **we propose an strategy for the propagation of changes to distributed copies of the same ontology** (section 5.3.1), which addresses current unsupported scenarios for collaborative ontology development where distributed editors can work with a local copy of the same ontology version. This strategy covers contribution **C3** that addresses the first part of objective **O2.1**.

On the other hand, **we present an strategy for the propagation of changes to ontology related metadata** (section 5.3.2). Having updated ontology metadata supports an accurate and efficient identification of changed ontologies, as well as the more general task of ontology reuse. This strategy covers contribution **C4** that addresses the second part of objective **O2.1**.

Our solution includes also the **definition of methods and strategies for complementary activities** required for the management of ontology changes (section 5.2). We define the process for capturing ontology changes (including their monitoring, processing and logging), and their storage and maintenance in a distributed registry. Additionally, we propose how to identify and deal with ontology versions. This work covers the first part of contribution **C5**, which addresses objective **O2.2**.

In Chapter 7 we introduce, among others, the technological support we provide for the management of changes in distributed environments that supports collaborative ontology development.

Chapter 6

COLLABORATIVE ONTOLOGY DEVELOPMENT SUPPORTED BY CHANGE MANAGEMENT

Previous efforts to support collaborative ontology development produced relevant methodological and technological results. However, as we have discussed in section 2.3.3, there are still some limitations in the existing approaches and tools. First, they support only a centralized management of the ontology and its related changes, addressing one of the following cases:

i) A shared/main copy of the ontology that is adapted/specialized by distributed users.

ii) A main copy of the ontology that is divided in sub-ontologies, each of them modified by distributed users.

iii) A central (and only) copy of the ontology that all distributed users can modify.

Second, they do not support the process followed by organizations for the coordination of the change proposals, which specifies the type of actions/operations ontology editors can perform depending on their role and on the state of the ontology (except from a recent conceptual work presented in [SNTM08]).

Third, in many of them there is not even a formal management of ontology changes (e.g., explicit representation and propagation of changes) such as the one we presented in the previous chapter. Even more important, there is no integrated approach (and technological support) that addresses all the previous issues.

This chapter presents our solution to support collaborative ontology development in an organizational setting. It is based on a formalization of the collaborative process that coordinates proposals for changes among ontology editors and that is

supported by the models, methods and strategies for change management in distributed environments described in Chapters 4 and 5. As a result, our solution supports both a centralized and distributed management of the ontology and its changes.

In order to achieve our goal, we first identified the most relevant requirements to support collaborative ontology development based on the analysis of the processes typically followed by organizations in the development and maintenance of ontologies. For this analysis, we considered different processes and scenarios for collaborative ontology development and previous efforts in the state of the art (see 2.3). We illustrate the outcome of this analysis, as well as the models and strategies proposed, with a concrete example based on a case study of the NeOn project[1]: the fisheries ontologies lifecycle from FAO [MGGPISK07].

In section 6.1, we start with an overview of the collaborative scenarios that we identified during our analysis. Then we briefly introduce our running example followed by the summary of the derived requirements. Next, in section 6.2, we discuss how the collaborative process can be formalized by means of a collaborative workflow model and illustrate it with the particular collaborative process in our running example. Similarly, we discuss how the workflow model can be represented and implemented in a workflow ontology and illustrate it with our running example. Finally, in section 6.3, we describe the strategies for the management of the workflow during ontology development.

Regarding the contributions introduced in section 3.2, which address the corresponding objectives in section 3.1, in this chapter we present contribution **C6** in sections 6.2 and 6.3, which addresses objectives **O2.3** and **O2.4**.

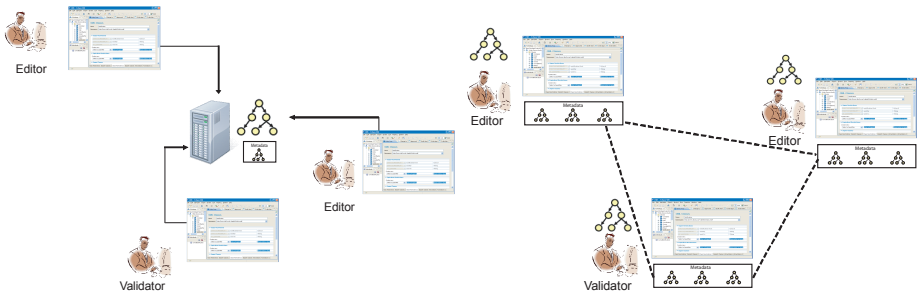
6.1 Analysis of Collaborative Ontology Development

As mentioned before, the collaborative development of ontologies within an organization usually follows a pre-defined process that specifies who (depending on the user role), when (depending on the ontology state) and how (what actions/operations) an ontology can change. However, the details of this process, as well as the configuration of the collaborative scenario, can vary from one organization to another (e.g., FAO[2], Gene Ontology (GO) project[3] and caGrid project[4]).

Therefore, based on existing collaborative settings for the maintenance of ontologies in different organizations, we identified a number of representative scenarios (some of them not supported by existing methods and tools) that we envision to support. Figure 6.1 depicts those scenarios.

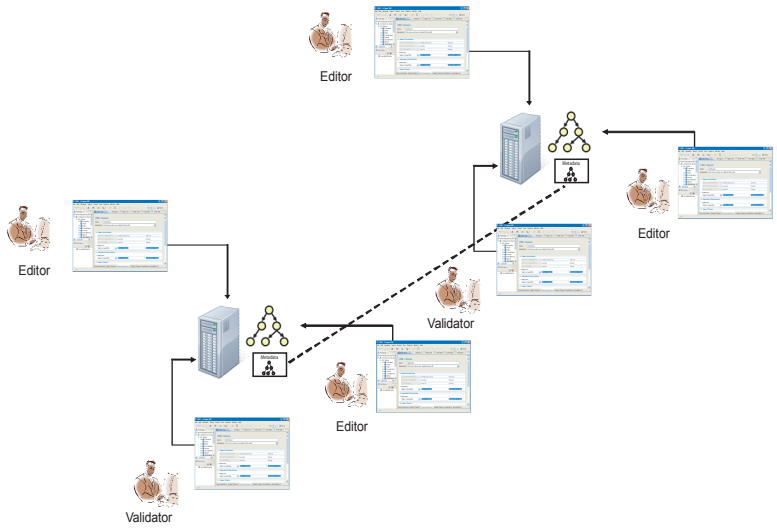
As illustrated in the figure, in each scenario, a team of ontology editors works collaboratively in the development/maintenance of an ontology (O1), following a

[1] http://www.neon-project.org/
[2] http://www.fao.org/
[3] http://www.geneontology.org/
[4] http://www.cagrid.org/



(a) Scenario A: Centralized Ontology Management

(b) Scenario B: Distributed Ontology Management



(c) Scenario C: Hybrid Ontology Management

Figure 6.1: Typical Collaborative Ontology Development Scenarios

pre-defined process that coordinates the proposal of ontology changes. However, in each case, the management of the ontology and its associated changes is performed in a different way, supporting different setups and characteristics of the collaborative environment.

i) In *Scenario A* (c.f. Figure 6.1(a)), the management is centralized. There is only one copy of the ontology, stored in a central server, that can be edited by all (distributed) members of the team from their (remote) PC station. The change information about the ontology is also stored in a central registry.

ii) In *Scenario B* (c.f. Figure 6.1(b)), the management is distributed. There are "n" copies of ontology "O1", where "n" is the number of ontology editors working on that ontology. Members of the team are physically distributed and each of them has his own copy of the ontology located at his (remote) PC station. The change information about the ontology is also stored in a distributed manner (e.g., in a local registry of each member PC station). The local PCs (nodes) exchange information about ontology metadata and synchronize the ontology changes. Changes received from other nodes are applied locally in the ontology copy to keep the distributed copies synchronized.

iii) *Scenario C* (c.f. Figure 6.1(c)) is a hybrid between scenarios "A" and "B". The team of ontology editors can be divided in at least two different groups: "X" and "Y". Each of these groups is working in an environment with the characteristics of scenario A. Additionally, the groups have between them an environment with the characteristics of scenario B. That is, each group has a centralized ontology server (and metadata provider), which in turn is treated as the ontology editor's PCs in configuration B.

As we can see from the previous description, scenarios "B" and "C" are not supported yet by existing methods and tools as they rely on a distributed management of the ontology and its related changes (see section 2.3.3).

Using as basis the analysis of the identified scenarios, as well as previous efforts in state of the art, we derived a set of requirements to support such collaborative settings for ontology development. We summarize those requirements after introducing our running example.

6.1.1 Running Example

In the organization considered, ontology engineers are developing an ontology-based information system to facilitate the assessment of fisheries stock depletion by integrating the variety of information sources available. In this context, one of the goals is the development of an application for the management of the fishery ontologies and their lifecycle.

There are several actors involved in the ontology engineering process, such as experts in ontology modeling who are in charge of defining the original skeleton

of the ontology, ontology editors who are in charge of the everyday editing and maintenance of the ontologies, and subject matter experts who know about the domain to be modeled. Additionally, validators are subject experts who can move a change to production state for external availability. Ontology development follows a well-defined collaborative process, which needs to be supported in the engineering environment. This process allows ontology editors to consult, validate and modify the ontology collaboratively, keeping track of all changes in a controlled manner. Finally, once editors in charge of validation consider the ontology final, they are authorized to release it and make it available to end users and systems.

6.1.2 Requirements for Collaborative Ontology Development

The requirements identified for collaborative ontology development specify the specific functionality that needs to be supported for this task by collaborative tools. Some of the requirements that we introduce in this section are similar to the ones that have been already identified in the past (see the analysis of [NCLM06] in section 2.3). However, in our work, we identified additional requirements to support the process followed typically by many organizations to coordinate the collaborative ontology development. When appropriate, we illustrate the requirement using our running example.

6.1.2.1 Lifecycle Requirements

An ontology lifecycle consists of the activities carried out during the development and maintenance of the ontology, including the conceptualization, population and evolution.

Hence, a collaborative ontology development solution should implement the necessary mechanisms to allow ontology editors to consult, modify and validate ontologies, during their lifecycle. Additionally, in some cases, ontologies may also be published once they are fully validated.

Furthermore, the solution should ensure that the aforementioned activities are carried out in a controlled and coherent manner. Hence, it should be responsible for the **coordination of who** (depending on the user *role*) **can do what** (what kind of *actions*) **and when** (depending on the *state* of the ontology elements – classes, properties and individuals – and the role of the user).

The lifecycle activities are performed by the ontology engineers and editors in charge of the modeling and everyday editing and maintenance work of the ontologies. Each user is assigned a specific role (which has associated permissions) by the organization, based on his expertise and responsibilities. Depending on the user permissions, he can be in charge of developing specific fragments of ontologies, revising work done by others, or developing new versions of ontologies. Ontology engineers are specialized in ontology modeling techniques and issues, while ontology editors know about the ontologies' domain, but they usually know little or nothing about ontology software or design issues. In this thesis, we fo-

cus our attention on the users responsible of the implementation and maintenance of the ontology (ontology editors). For instance, one approach for assigning the user role can be driven by the module of the ontology the user is responsible for ([KSKM07]). In our running example, an ontology editor can be assigned one of the following roles:

- *Subject experts* (SE) know about specific aspects of the ontology domain and are in charge of adding or modifying ontology content.
- *Validators* (V) revise, approve or reject changes made by subject experts; they are the only ones who can copy changes into the production environment for external availability. They have a broader knowledge of the ontology domain and have at least some knowledge about design issues.

Additionally, a special role *Viewer* allows members of this role only to visualize information.

To enforce the permissions constraints, the solution should (i) support the different *user roles* and (ii) request that users identify themselves to the system before using it. Furthermore, to control when the ontology editors are allowed to work with an ontology element, in addition to the user roles, every ontology element is required to have a *state*. Ontology editors can change the state depending on their role. For instance, the possible states ontology elements can have in our running example are *Draft* for the proposed additions or updates, *To Be Approved* for the proposed changes that are ready to be reviewed by a validator, *Approved* for the accepted changes, *To Be Deleted* for the proposed removals and *Deleted* for the definitive removals.

6.1.2.2 Activities Requirements

The activities required to be supported by the collaborative ontology development solution include the operations (or possible *actions*) that the ontology editors are allowed to perform depending on their roles and the state of the ontology elements.

Edit ontology element.

Insert ontology elements. Usually, only ontology editors who are expert in the ontology domain are allowed to add new elements to the ontology. However, there may be cases in which non-expert ontology editors should be allowed to insert ontology elements.

Update ontology elements. Editors can update ontology elements, depending on their role and the state of the element. For instance, in our running example, a SE can only update elements in ”Draft” or ”Approved” state.

Delete ontology elements. Editors propose elements for deletion. In general this is not a definitive action, and it has to be authorized by an appropriate editor.

Validate change. Authorized ontology editors should be able to validate changes proposed by other ontology editors and decide whether to approve or reject them. As discussed above, an ontology editor is authorized to validate changes depending on his/her user role.

Change state of ontology element. While inserting, updating and deleting elements, their state is automatically changed by the system. There are other cases where a specific action from editors is required to change the state of ontology elements. For instance, in our running example, SE's need to explicitly send elements in "Draft" state to the "To Be Approved" state.

Publish ontology. In some organizations, authorized editors are allowed to copy an ontology from the test/validation environment (Intranet) to the production environment (Internet). By doing so, the system automatically assigns the right version to the published ontology following a versioning scheme.

6.1.2.3 Visualization Requirements

Besides supporting the different activities to modify or validate ontology elements or to publish ontologies, ontology editors require to consult the corresponding information generated by those actions, as follows:

View change history. Editors need to be able to view the logs of ontology changes and their related information including the history notes, e.g., argumentation of the tracked changes.

View based on states and user role. The interface should be able to provide different data views based on the user role.

View use statistics. Editors can view information about an ontology regarding how the ontology has been used or evolved throughout the time, e.g., provenance, editors, frequency of changes and the fragment/domain of the ontology changed most rapidly.

View ontology statistics. Authorized users can view statistics of the ontology being edited, e.g., depth of the class hierarchy; number of child nodes; number of relationships and properties; number of concepts per branch.

6.1.2.4 Change Management Requirements

Since the whole collaborative scenario is driven by changes in ontologies, the following requirements can be identified for the management of those changes:

Representation of changes. A main requirement is the explicit representation of the changes that editors are able to perform to ontologies. The representation should ensure the accessibility and interoperability with other components, for example, the collaborative process and ontology metadata; it should also ensure the maintenance of the chronological order of the changes to support, for example, the undo/rollback operations or the reconstruction of performed operations (e.g., when syncrhonizing/propagating changes). Additionally, to facilitate the previous tasks and provide an efficient link between what the user sees (ontology elements) and what the system manages internally (e.g., axioms), the representation should provide a flexible classification of changes that considers the actual "atomic" operations that can be performed over ontologies, in addition to operations at the element level (to support the different states that each ontology element can have during the

collaborative ontology development process) or the complex operations that have been considered in the past (see section 2.2.2). Information about changes should include, for instance, the operation performed, the time of the operation, the user, the element associated, the previous change and the description.

Capture ontology changes. Changes should be captured automatically from the ontology editor. The capturing of changes involves different activities such as the monitoring, processing, encoding (at the right granularity level) and logging of changes. Additionally, changes should be stored and maintained in an appropriate knowledge base.

Change Propagation and notification. After new changes are submitted to the ontology, the editors involved in this collaborative process should be informed when they log into the system. Each author (or the coordinator) should be able to view changes made by other authors, even without editing permission (see scenarios B and C from Figure 6.1).

6.1.2.5 Versioning Requirements

An additional requirement is the management of ontology versions. The first modification to an approved/published ontology automatically changes the current version. This modified version of the ontology will either become a new version (with a different version information) or, if specified by the editor, remain the same version. In any case, versions need to be uniquely identified.

6.1.2.6 Concurrency Control and Conflict Resolution Requirements

An important issue that has to be addressed in this collaborative scenario is to ensure the integrity of the ontology via concurrency control mechanisms and appropriate means for the resolution of conflicts whenever two or more editors submit changes to the same element concurrently.

6.1.3 Summary

As we can see from the derived requirements, we have addressed some of them in Chapter 5. In particular, the management of changes including their representation, capturing and propagation, the management of different ontology versions and to some extent the concurrency control by the synchronization of local ontology copies. Additionally, the automatic periodic execution of the synchronization process minimizes the conflicts during collaborative ontology development. However, explicit conflict resolution mechanisms are out of the scope of this thesis. Hence, in the remainder of this chapter we provide solutions to support the lifecycle and activities requirements, and in Chapter 7, we address the visualization requirements with our technological support.

6.2 Workflow Model

Based on the analysis presented in the previous section we found out that during the collaborative ontology development process, some of the possible actions and states apply at different levels of abstraction. Therefore, we propose to consider this process at two different levels: the element level and the ontology level. In each level, we need to specify the way in which the different elements involved in the collaborative process (roles, states, actions) are related to each other.

According to [GLP+07], a collaborative workflow allows modeling relations among designers, ontology elements, and collaborative tasks. Hence, we propose the use of collaborative workflows to formalize the collaborative ontology development process. We refer to this particular kind of collaborative workflow as *collaborative editorial workflow* (or just workflow).

Although the workflows can be used independently of the underlying ontology model, the specific set of ontology elements depend on the ontology model. In our solution we are mainly considering the OWL 2 ontology model, in which an OWL ontology consists of a set of axioms and facts (see section 5.1.2.1). Facts and axioms can relate to classes, properties or individuals, and hence, these are the set of ontology elements we are considering.

As previously discussed, the details of the collaborative ontology development process (the specific roles, actions, etc.) depend on the organization setting. Therefore, to illustrate our approach, in the rest of this section we discuss our solution for the particular scenario of our running example.

Illustration with running example

Figures 6.2 and 6.3 show the two different workflow levels (element and ontology level) for our running example. States are denoted by rectangles and actions by arrows. The information in parentheses specifies the actions that an editor can perform depending on its role, where "SE" denotes Subject Expert, "V" denotes Validator and "-" denotes that the action is performed automatically by the system.

The possible states that can be assigned to ontology elements (see Figure 6.2) are the following:

- *Draft*: This is the state assigned to any element when it passes first into the workflow, or when it was approved and then updated by a subject expert.

- *To be approved*: Once a "SE" is confident with a change in draft state, the element is passed to the "To Be Approved" state and remains there until a "V" approves/rejects it.

- *Approved*: If a "V" approves a change in an element in the "To Be Approved" state, it passes to the "Approved" state. Additionally, this is the default state for every element of the initial version of a stable ontology.

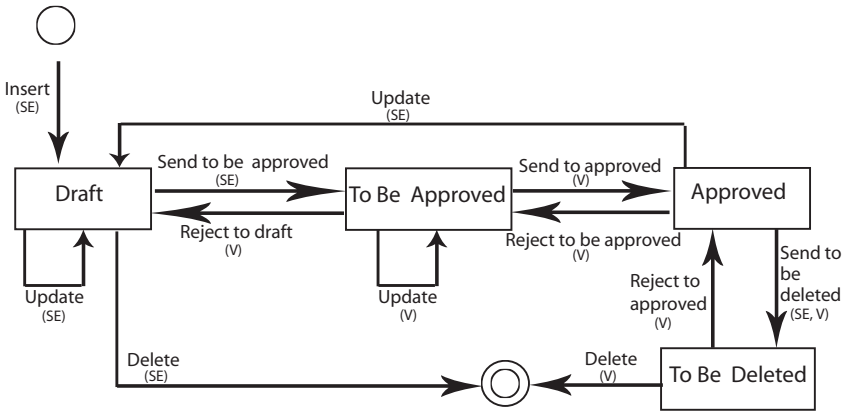


Figure 6.2: Collaborative Editorial Workflow At The Element Level

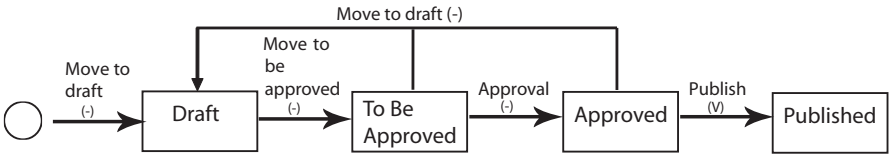


Figure 6.3: Collaborative Editorial Workflow At The Ontology Level

- *To be deleted*: If a ”SE” considers that an element needs to be deleted, the item will be flagged with the ”To Be Deleted” state and removed from the ontology although only a ”V” will be able to definitively delete it.

- *Deleted*: This is the state assigned to any element when a ”V” definitely deletes it after its state was ”To Be Deleted”. Additionally, this is the state assigned to any element when a ”SE” deletes it after its state was ”Draft”. Note that this state is represented by the doubled circle in the figure.

The ontology state (see Figure 6.3) is automatically assigned by the system (denoted with ”-” in the figure), except from the ”published” state. The ontology state is based on the state of its elements, i.e., the state of an ontology is given by the ”lower” (less stable) state of any of its elements. For instance, it can happen that in the same ontology there are elements in ”draft” state and elements in ”to be approved” state, then the state of the ontology should be ”draft”. Hence, the ontology states are defined as follows:

- *Draft*: Any change to an ontology in any state automatically sends it into draft state.

- *To be approved*: When all ontology elements in an ontology version are in "To Be Approved" state (or deleted), the ontology is automatically sent to "To Be Approved" state.

- *Approved*: When all ontology elements in an ontology version are in "Approved" state (or deleted), the ontology is automatically sent to "Approved" state. Additionally, this is the default state of the initial version of a stable ontology.

- *Published*: Only when the ontology is in "Approved" state, it can be sent by a validator to "Published" state.

In our running example, the workflow starts after getting a stable populated ontology that satisfies all the organizational requirements. Hence, we assume that the initial state of this stable ontology (and all its elements) is "Approved"[5].

Note that during the workflow, actions are performed either

- *Implicitly*: For instance, when a user updates an element, he does not explicitly performs an update action. In this case it has to be captured from the user interface. Action is recorded after the operation is successfully executed.

- *Explicitly*: For example, Validators explicitly approve or reject proposed changes. Action is recorded immediately when the user explicitly performs the action.

The actions that an ontology editor is allowed to perform depend on its role. Table 6.1 summarizes the actions that subject experts and validators can perform at the element level, along with the state of the element before and after the execution of the action.

Table 6.2 describes the possible actions at the ontology level, along with the state of the ontology before and after the execution of the action. Note that, as we described above, most of these actions are automatically performed by the system (shown as "–" in the column "Role").

As we can see from the previous discussion, the collaborative ontology development process highly depends upon the operations performed to the ontology itself (the changes performed by ontology editors). Hence, similarly to our change representation model, we decided to model the workflow elements (roles, states, actions) using an (OWL-Lite) ontology (workflow ontology) that allows the formal and explicit representation of knowledge in a machine-understandable format. Furthermore, having both models (ontology changes and workflow) formalized as ontologies will facilitate the representation of the tight relationship that exists between them. For instance, consider a user with role "subject expert" that "inserts" a new ontology "class" to the ontology. That "class" will receive automatically the

[5]In a different scenario, the workflow could start with an empty ontology (without elements), which we could assume that will be by default in "Approved" state

Table 6.1: Collaborative editorial workflow actions at the element level

Action	**Previous State**	**Next State**	**Role**	**Remarks**
Insert	—	Draft	SE	This action triggers the start of the workflow.
Update	Draft	Draft	SE	—
Delete	Draft	—	SE	The item will be automatically deleted.
Send to be approved	Draft	To Be Approved	SE	This action moves the responsibility of the item from the subject expert to the validator. While an item is in the ”To be approved” state, the subject expert cannot modify it.
Reject to draft	To Be Approved	Draft	V	The validator rejects the change.
Update	To Be Approved	To Be Approved	V	An update done by a validator does not need to be double checked by other validators, so the element will remain in the same state as it was.
Send to approved	To Be Approved	Approved	V	The validator accepts the change.
Update	Approved	Draft	SE	This action triggers the start of the workflow.
Send to be deleted	Approved	To Be Deleted	SE, V	—
Reject to be approved	Approved	To Be Approved	V	The validator wants the change to be reviewed again.
Update	Approved	Approved	V	An update done by a validator does not need to be double checked by other validators, so the element will remain in the same state as it was.
Reject to approved	To Be Deleted	Approved	V	The validator does not agree with an element previously proposed ”To be deleted”.
Delete	To Be Deleted	—	V	The element is removed from the ontology.

Table 6.2: Collaborative editorial workflow actions at the ontology level

Action	Previous State	Next State	Role	Remarks
Move to draft	—	Draft	–	This action should be performed automatically by the system when the first change is submitted to an ontology version in ”Approved” or ”Published” state.
Move to be approved	Draft	To Be Approved	–	This action should be performed automatically by the system when all changes to an ontology version are in ”To Be Approved” state (or deleted).
Approval	To Be Approved	Approved	–	This action should be performed automatically by the system when all changes to an ontology version are in ”Approved” state (or deleted).
Publish	Approved	Published	V	The validator can decide to publish an ontology only if it is in ”Approved” state (copy the Approved ontology into the production environment and probably change version information).

"draft" state. All the information related to the process of inserting a new ontology element will be captured by the workflow ontology, while the information related to the particular element inserted along with the information about the ontology before and after the change is captured by the change ontology. Additionally, the workflow process also relies on OMV to refer to ontologies and users.

6.2.1 Workflow Ontology

The main classes and properties of the workflow ontology along with the relationships with the other ontologies in our approach (e.g., OMV and Generic Change Ontology) are illustrated in Figure 6.4. In the figure, the elements above the horizontal line are generic elements of the workflow ontology that are independent of the collaborative process details, while the elements below the line are specific for our running example (see below).

The different roles that ontology editors can have are modeled as individuals of the `Role` class that is related to the `Person` class of the OMV core ontology (a person has a role).

To explicitly model the separation between the possible states of ontology elements (classes, properties and individuals) and the possible states of the ontology itself, the `State` class is specialized in two subclasses (`EntityState` and `OntologyState`). Similarly to the roles, the possible values of the states are modeled as individuals of their respective subclass. Furthermore, the two subclasses of `State` allow representing the appropriate relationships at the element and ontology level: to specify that an ontology element has a particular state, we rely on the class `EntityChange` from the change ontology which is associated to a particular ontology element (as described in section 5.1) and associate it with subclass `EntityState`; to specify that an ontology has a particular state, we rely on the class `Ontology` from the OMV core and associate it with the subclass `OntologyState`. Note that the state is not assigned to the actual object (e.g., OWL ontology element or OWL ontology) but to the referring metadata entity (entity change or the ontology metadata) for two reasons: first, it is the actual referring entity the one that modifies the state of the object, and second, all this information is managed by the ontology registry that stores all related ontology metadata but not the ontologies themselves.

Finally, for the actions there is also a separation between the possible actions at the element level and actions at the ontology level. Hence, the `Action` class is specialized in two subclasses (`EntityAction` and `OntologyAction`). To track the whole process (and keep the history) of the workflow, the possible actions are modeled as subclasses of the appropriate `Action` subclass. Similar to the states, the two subclasses of `Action` also allow representing the appropriate relationships at the element and ontology level. To specify that an action was performed over a particular ontology element, the subclass `EntityAction` is associated with class `EntityChange`. However, as illustrated above with our running example, actions at the ontology level are usually performed automatically

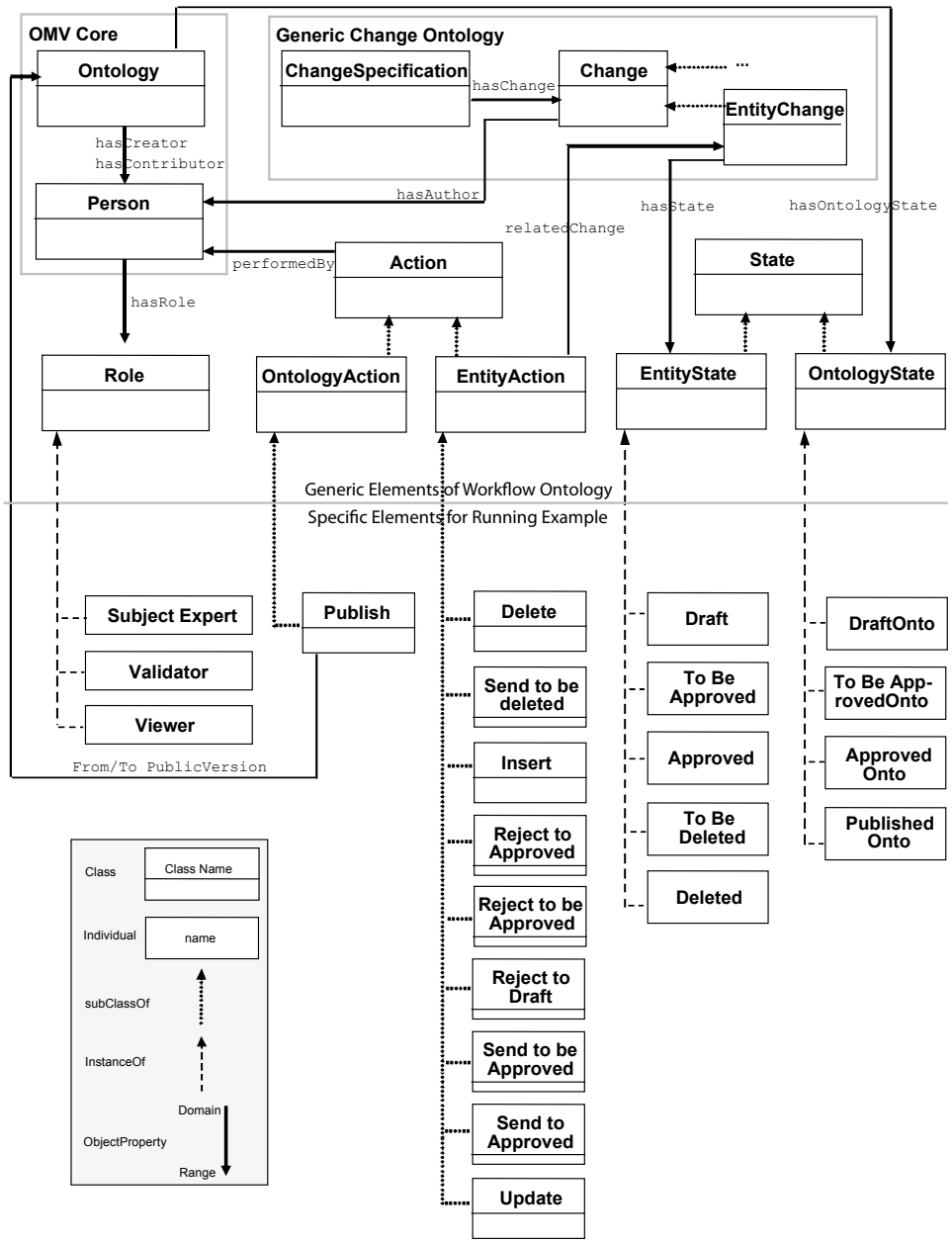


Figure 6.4: Workflow Ontology

by the system. The explicit actions at the ontology level are dependent on the specific workflow details. Therefore, we do not specify any association between the generic subclass `OntologyAction` and class `Ontology`; associations should only be specified, if necessary, between the explicit actions (at the ontology level)

of the specific workflow and the class `Ontology` (see below).

Illustration with running example

As we can see in Figure 6.4, the three different roles of our running example (Subject Expert, Validator, Viewer) are modeled as individuals of the class `Role`.

Similarly, the five possible states that can be assigned to ontology elements (Draft, To Be Approved, Approved, To Be Deleted and Deleted) are modeled as individuals of the class `EntityState` and the four possible states for an ontology (Draft, To Be Approved, Approved and Published), as individuals of the class `OntologyState`.

Finally, the nine actions that ontology editors can perform over ontology elements (Insert, Update, Delete, Send To Be Approved, Send To Approved, Send To Be Deleted, Reject To Draft, Reject To Be Approved, Reject To Approved) are modeled as subclasses of the class `EntityAction`. Additionally, the only explicit action that can be performed to the ontology (Publish) is modeled as subclass of the class `OntologyAction`. Notice that, as described in the previous section, the action "Publish" may change the version of the ontology. Therefore, it is associated to the class `Ontology` to specify the previous and next public version of the ontology.

Note that in the current implementation of our workflow ontology, the constraints of the collaborative process in our running example are not explicitly declared as part of the ontology. Those constraints are handled by the workflow management strategies discussed in next section. The reason for taking a procedural approach instead of a declarative one was mainly due to time limitations. Future work in this area should evaluate the practicability of such a declarative approach (see Chapter 8).

6.3 Workflow Management

The workflow management consists on the execution of several tasks to support the collaborative ontology development.

- It should request users to identify themselves. They should provide at least their username and corresponding role.
- It should enforce the constraints imposed by the collaborative workflow. That is, every time an ontology editor performs a workflow action (e.g., insert new element or submit a change to be approved), the following tasks should be carried out:
  - i) To verify that the ontology editor has the appropriate permissions. This can be performed by verifying the role of the user. For instance, the collaborative workflow may constrain that only subject experts can insert new ontology elements (as in our running example).

ii) To verify the state of the corresponding ontology element (if any) for the proposed action. For this task, we need to evaluate the current state of the element before applying the requested action. For instance, a change in an ontology element may not be approved if it is in draft state.

iii) To verify the provenance of the action. For instance a subject expert may not be allowed to submit changes from another subject expert to be approved. For this task, the author of the change has to be compared with the current user.

- It should support transaction management capabilities. This means that if the performed action is rejected, all of its effects have to be undone.

- It should create and store the appropriate individuals of the workflow ontology, if an action is successfully executed. For this task, the information about the workflow action has to be collected (e.g., author of the change, related ontology element, type of action and time), processed, encoded and logged in an appropriate knowledge base. This knowledge base should be in charge of the storage and maintenance of the workflow ontology individuals (the same as for individuals of the generic/specialized change ontology, described in section 5.2).

- It should perform all consequences of an action if it is successfully executed. For instance, if a subject expert submit one change to be approved, the state of the change has to be updated to "To Be Approved". This task also involves storing the corresponding state of the related element along with the workflow ontology individual.

Additionally, the workflow management should provide the appropriate interfaces for the users to visualize and perform all the required actions. For instance, ontology editors (e.g., subject experts and validators) should be able to have different views of the ontologies developed collaboratively, depending on their role.

Illustrative example Lets consider the following simple scenario:

Subject expert "John" adds class1 to ontology O1_V1[6] and decides to create a new version of that ontology (V2), which is assigned automatically *Draft* state. As it is the first change to an ontology version, an individual of the class *ChangeSpecification* is created (changeSpecification1) to group all changes applied to that version. Additionally, to log the corresponding change, the particular individual of the *Change* class is created (addClass1) with an associated *Draft* state and is associated to changeSpecification1 and to the actual class added (class1). Finally, the corresponding action (insert1) is created and associated to the corresponding change. An extract of the individuals of the generic change ontology and the workflow ontology created could be as follows:

[6]Assume the V1 is the first stable version of that ontology and hence has an *Approved* state

```
<owlchange:ChangeSpecification rdf:ID="changeSpecification1">
 <owlchange:hasChange rdf:resource="#addClass1"/>
 <owlchange:fromVersion rdf:resource="#O1_v1"/>
 <owlchange:toVersion>
  <omv:Ontology rdf:ID="O1_v2">
   <omv:version rdf:datatype="http://www.w3.org/2001/XMLSchema#string">2.0</omv:version>
   <hasOntologyState> <OntologyState rdf:ID="DraftOntology"/> </hasOntologyState>
   ...
  </omv:Ontology>
 </owlchange:toVersion>
 ...
</owlchange:ChangeSpecification>
<Insert rdf:ID="insert1">
 <relatedChange>
  <owlchange:EntityChange rdf:ID="addClass1">
    <owlchange:relatedEntity rdf:resource="#class1"/>
    <hasState> <EntityState rdf:ID="Draft"/> </hasState>
    <owlchange:hasAuthor>
     <omv:Person rdf:ID="John">
      <hasRole rdf:resource="#SubjectExpert"/>
     </omv:Person>
    </owlchange:hasAuthor>
    ...
  </owlchange:EntityChange>
 </relatedChange>
 <performedBy rdf:resource="#John"/>
 ...
</Insert>
```

Once John is sure of his change, he sends it to be approved (creates a new individual of the action *send to be approved*). Eventually, the validator "Jane" will analyze the proposed change and she will approve it (which creates an individual of action *send to approved*) or reject it (which creates an individual of action *reject to draft*). The following is an extract of the action *send to approved*[7]:

```
<SendToApproved rdf:ID="approve1">
 <relatedChange rdf:resource="#addClass1"/>
 <performedBy>
  <omv:Person rdf:ID="Jane">
    <hasRole rdf:resource="#Validator"/>
  </omv:Person>
 </performedBy>
 <relatedChange> <owlchange:EntityChange rdf:ID="addClass1"> </relatedChange>
 ...
</SendToApproved>
```

6.4 Summary

In this chapter we have presented our solution to support collaborative ontology development. Based on the requirements identified in section 6.1.2, we address some of the limitations identified in the state of art in this area.

First, **our solution supports different collaborative scenarios**, typical in an organizational setting, where the management of the ontology and its related

[7]Note that the corresponding change will have now *approved* state and since there are no other changes in this scenario to O1_V2, its state will also change to *approved*.

changes can be centralized, distributed or be a hybrid between both of them (section 6.1). So, unlike the existing approaches that support only a centralized management, we also address scenarios in which distributed ontology editors can work with a local copy of the ontology, while our solution takes care of synchronizing the different copies by means of the change propagation strategies presented in 5.3.1. As a consequence, we provide a more flexible solution when compared to the existing approaches that can address different organizational needs and limitations. For example, in many cases, ontology editors may not be able to have a permanent or reliable connection to a central server to work with an ontology. Similarly, other well known problems of centralized approaches (e.g., performance, maintenance and resources) can be avoided by using a distributed control.

Second, unlike most existing approaches, we focus on the process followed by organizations for the coordination of the change proposals. We propose to **formalize this process by means of a collaborative editorial workflow model** (section 6.2). This kind of workflow allows specifying relations among designers, ontology elements, and collaborative tasks ([GLP+07]). Further, we **implement this workflow model as an ontology itself, called workflow ontology** (as shown with our running example). Having a workflow ontology allows the formal and explicit representation of the workflow knowledge in a machine-understandable format, which can be easily integrated with other models (e.g., our change representation model). Besides, having the history of the workflow as individuals of an ontology allows reusing existing ontology-driven technologies for the processing of the workflow information (e.g., propagation mechanisms and reasoning). Additionally, we presented in section 6.3 the **strategies that we propose for the management of the collaborative process during the ontology development**. We identify the set of tasks that have to be carried out, including those required to enforce the constraints of the collaborative process, and propose strategies to deal with them. This work covers contribution **C6**, which addresses objectives **O2.3** and **O2.4** (see Chapter 3).

Third, since our solution is based on the models, methods and strategies presented in Chapter 5, **we provide a formal management of ontology changes**, as mentioned previously.

In Chapter 7 we present, among others, the technological support for collaborative ontology development that implements in an integrated infrastructure the solutions proposed in this chapter and Chapters 4 and 5.

Chapter 7

EVALUATION

The models, methods and strategies proposed in this thesis have been implemented as part of an infrastructure that supports collaborative ontology development. The basis of this infrastructure is a distributed ontology registry that is responsible for the management of ontology metadata. The infrastructure shows the practicability of our contributions. Moreover, based on this infrastructure, we have conducted an experiment in a real scenario for evaluating the usability and performance of our contributions. Besides, our conceptual models have been evaluated using specific criteria such as applicability, completeness or adequacy.

Table 7.1 describes the general overview of the criteria used for evaluating the contributions of this thesis.

Table 7.1: Evaluation overview

Contribution	Criteria
Ontology Metadata	• Applicability of OMV.
Change Management	• Completeness of OWL 2 change representation model. • Adequacy of change representation model. • Usability and Performance of the System.
Collaborative Ontology Development	• Adequacy of workflow model. • Usability and Performance of the System.

Hence, in section 7.1, we start by introducing the implementation support of our work, and then we present the evaluations performed: in section 7.2, we verify the applicability of OMV by analyzing its acceptance in the community. In section 7.3 we evaluate the completeness of the change representation model by analyzing the full set of possible changes for the OWL 2 ontology language and by verifying that every possible change can be represented in our model. Finally, in section 7.4

we describe and analyze the experiment we conducted in FAO headquarters with a set of representative users to evaluate the complete infrastructure.

Regarding the contributions introduced in section 3.2, which address the corresponding objectives in section 3.1, we present here the following: section 7.1.1 presents Oyster (part of contribution **C5**, which addresses objective **O3.1**) and section 7.1.2 presents the infrastructure to support collaborative ontology development (contribution **C7**, which addresses objectives **O3.2**, **O3.3** and **O3.4**).

7.1 Technological Support

Our technological contribution consists in the implementation of (i) a distributed ontology registry and (ii) a framework for the collaborative ontology development in distributed environments. Our registry is responsible for the maintenance and management of ontology (and related entities) metadata; it is also a core component for the collaborative framework. In this section we start by briefly presenting our registry (7.1.1) followed by the description of our framework (7.1.2).

7.1.1 Distributed Ontology Registry - Oyster

Oyster is a distributed registry that exploits semantic web techniques in order to provide a solution for the management of metadata about ontologies and related entities[1] in distributed environments. As an ontology registry, it provides services for storage, cataloging, discovery, management, and retrieval of ontology (and related entities) metadata definitions. To achieve these goals, Oyster implements OMV as a way to describe ontologies and related entities, supporting the exchange and re-use of ontologies and related entities (e.g., advanced semantic searches of the registered objects) and providing services to support the management and evolution of ontologies in distributed environments.

Oyster offers a user driven approach where each peer has its own local registry of ontology related metadata and access to the information of other registries, thus creating a virtual decentralized ontology metadata registry. The goal is a decentralized knowledge sharing environment using Semantic Web technologies that allows developers to easily share and maintain ontologies and related metadata.

The Oyster system [2] was designed using a service-oriented approach, and it provides a well defined API. The registry functionalities can be accessed by (i) using directly the API within any application, (ii) invoking the web service provided, (iii) using the included java-based GUI as a client for the distributed registry, (iv) using the plug-ins provided for the NeOn Toolkit or (v) starting the registry in server mode from the command prompt. In Oyster, ontologies are used extensively to provide its core registry services as follows:

[1] In the rest of this section we refer to the metadata about ontologies and related entities as ontology related metadata

[2] `http://ontoware.org/projects/oyster2/`

- **Creating and importing metadata**. Oyster enables users to create metadata about ontologies explicitly (submitting the metadata entry), as well as to import ontology files and to automatically extract the ontology metadata available letting the user to complete it later. The ontology metadata entries are aligned and formally represented according to two ontologies (depending on the object described): (1) the OMV ontology (core or appropriate extension depending on the type of object described) and (2) a topic hierarchy (e.g., the DMOZ topic hierarchy) for describing specific categories of subjects for some types of objects (e.g., to define the domain of the ontology and the expertise of a peer).

- **Formulating queries**. A user can search for ontologies or related entities using simple keyword searches, or using more advanced, semantic searches. Here, queries are formulated in terms of these two ontologies. This means queries can refer to fields like name, acronym and ontology language, or queries may refer to specific topic terms.

- **Routing queries**. Users may query a single specific peer (e.g., their own computer because they can have many ontologies stored locally and finding the right one for a specific task can be time consuming or another peer in particular because this peer is a known big provider of information), or a specific set of peers (e.g., all the member of a specific organization), or the entire network of peers (e.g., when the user has no idea where to search), in which case queries are routed automatically in the network.

- **Processing results**. Finally, results matching a query are presented in a result list. The answer of a query might be very large and contain many duplicates due to the distributed nature and potentially large size of the P2P network. Such duplicates might not be exactly copies because the semi structured nature of the metadata, so the ontologies are used again to measure the semantic similarity between different answers and to remove apparent duplicates.

In addition to the core registry services, Oyster provides advanced services to support the evolution of ontologies, including: the management of ontology changes, the management of collaborative editorial workflows that coordinate the proposal of ontology changes, and the synchronization of their related information (see next section for a detailed description).

Oyster Architecture

The high-level design of the architecture of a single Oyster node is shown in Figure 7.1.

Next, we briefly discuss the individual components of the system architecture. For a complete description we refer the reader to [PH05] and [WHP07].

- The Local Repository of a node contains the metadata about ontologies and related entities (OMV individuals) that it provides to the network. It supports

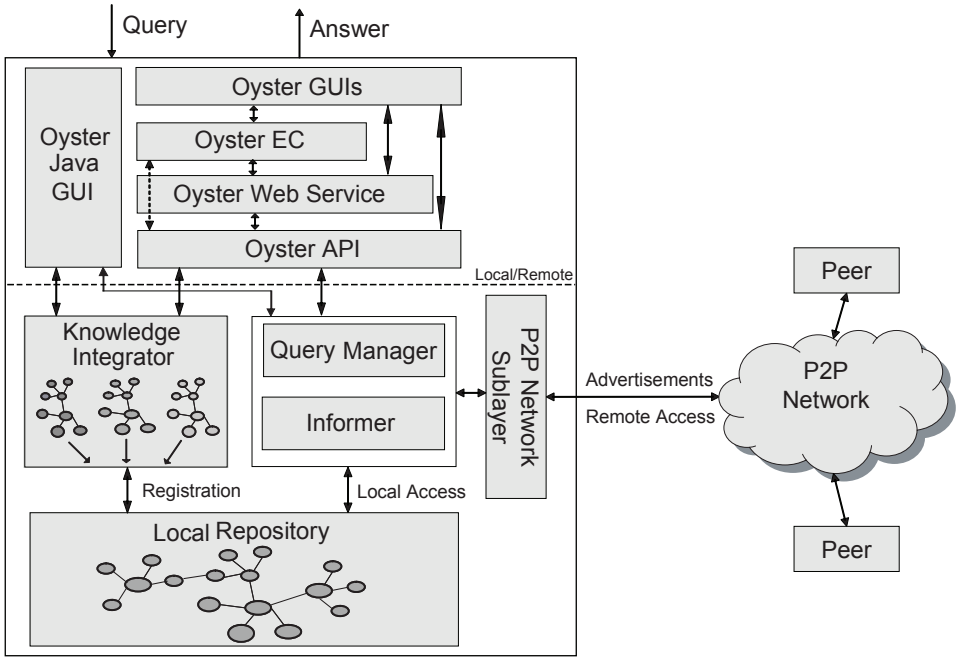


Figure 7.1: Oyster Architecture

query formulation and processing and provides the information for peer selection. In Oyster, the Local Repository is based on KAON2, and it supports SPARQL as its query language.

- The Knowledge Integrator component is responsible for the extraction and integration of knowledge sources (e.g., ontologies) into the Local Repository. This component is also in charge of detecting and merging duplicated query results.

- The Query Manager is the component responsible for the coordination of the process of distributing queries. It receives queries from the user interface, API or from other peers. Either way it tries to answer the query or to distribute it further according to the content of the query.

- The Informer component is in charge of proactively advertising the available knowledge of a Peer in the distributed network and of discovering peers along with their expertise. The expertise contains a set of topics (ontology domains) that the peer is an expert in.

- The Peer-to-Peer network sub-layer is the component responsible for the network communication between peers. In Oyster, we rely on an RMI based implementation; however, other communication protocols would be also possible.

- The Oyster API provides a well defined interface in Java with a set of methods that expose all the registry functionalities.

- The Oyster Web Service encapsulates the Oyster API and provides a free realization of the NeOn registry service compliant with the eBXml standard.

- The Oyster Engineering Components (EC) provide specialized services for the management of the ontology metadata. The aforementioned services, i.e., change management and workflow management, are examples of those components.

- The Oyster GUI's provide ready-to-use clients for the registry. The GUI's should access the registry via the web service or the API. However, Oyster also includes a former java-based GUI that accesses directly the registry.

7.1.2 Framework For Collaborative Ontology Development In Distributed Environments

The models, methods and strategies proposed in this thesis for change management in distributed environments and collaborative ontology development have been implemented within the NeOn Toolkit[3], an extensible ontology engineering environment based on Eclipse, by means of a set of plugins and extensions. A high level conceptual architectural diagram of the components involved is shown in Figure 7.2.

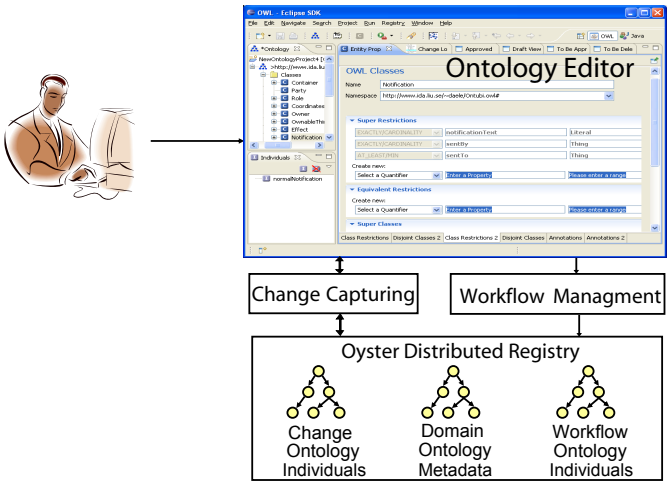


Figure 7.2: Conceptual architecture for the collaborative ontology development support

[3] http://www.neon-toolkit.org/

Next, we present the change capturing related components (left side of the figure), then, the workflow management related components (right side of the figure), next, the user related components for visualizing and editing ontologies (and related information) following a collaborative editorial workflow (upper part of the figure) and, finally, the role of our distributed registry within this framework (bottom part of the figure). Unlike similar existing tools (see 2.3.2), our solution provides a flexible mechanism to support different collaborative scenarios, as we describe in detail at the end of this section. For example, in addition to the typical scenario where a team of ontology editors are working collaboratively with a centralized copy of an ontology, we also support scenarios in which ontology editors are working collaboratively with distributed copies of the same ontology. Note that our infrastructure to support the collaborative ontology development is supported by the management of changes in distributed environments, thus verifying hypothesis H1 (cf. 3.4).

7.1.2.1 Change Capturing Components

Once the ontology editor specifies that he wants to monitor an ontology, changes are automatically captured from the ontology editor by a change capturing plugin. This plugin implements the methods and strategies presented in section 5.2 for the capturing of ontology changes and the maintenance of different ontology versions (c.f. *versioning and change management requirements* in section 6.1.2). In a nutshell, this plugin is notified about events that consist of ontology changes performed by the user in the ontology editor. For each of these events, the change is represented according to the change ontology by creating the appropriate individual. For example, adding a class individual in the ontology editor creates the entity change "Add Individual" and the two corresponding atomic changes (OWL 2 axioms): "Add Declaration" and "Add ClassMember". As described by the change ontology, each individual includes relevant information such as the author, the time, the related ontology and others. The individuals are stored in the Oyster distributed registry [4].

This plugin is also in charge of applying changes received from other clients to the same ontology after Oyster synchronizes the changes in the distributed environment (see last subsection). Finally, this plugin extends the NeOn Toolkit with a view (see Figure 7.3) to display the history of ontology changes (c.f. *visualization requirements* in section 6.1.2).

7.1.2.2 Workflow Management Component

In our implementation, the workflow management component implements the strategy described in section 6.3 as an engineering component in Oyster (c.f. *life-*

[4] In a different scenario, if the system is presented with two versions of the same ontology without their change history, ontology changes could be derived according to the change ontology, similar to the PROMPT tool[NM02]

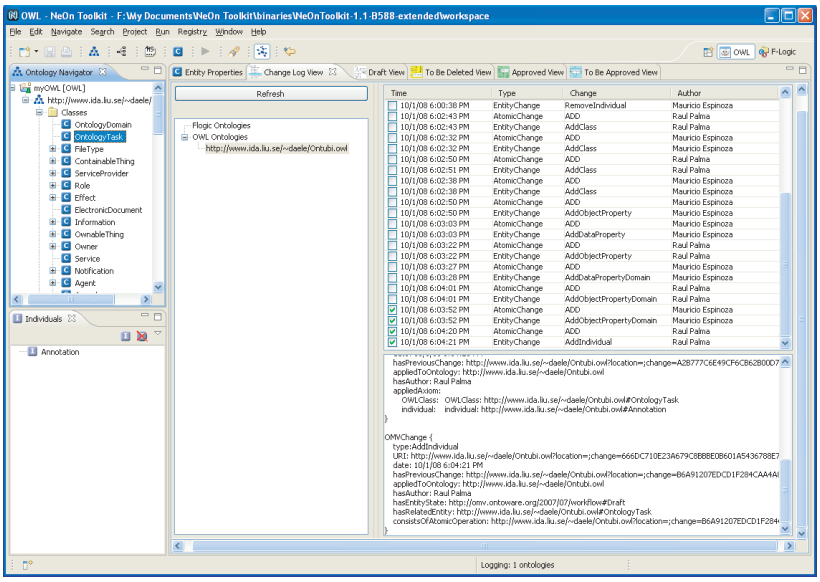


Figure 7.3: Change log view

cycle requirements in section 6.1.2). Hence, it (i) takes care of enforcing the constraints imposed by the collaborative editorial workflow, (ii) creates the appropriate action individuals of the workflow ontology and (iii) registers them into the distributed registry ensuring a transactional support. In detail, whenever a new workflow action is performed, the component performs the following tasks:

- It gets the identity and role of the user performing the action (if it is an explicit action such as "send to approve"), or the associated change (if it is an implicit action). For instance, adding a new class implicitly creates an insert action.
- It gets the state of the ontology element associated to the action/change.
- It verifies that the role associated to the user can perform the requested action when the ontology element is in that particular state.
- If the verification succeeds, it creates the workflow action and registers it.
- If the verification fails, it undoes the associated change(s) for the implicit actions because the complete operation (e.g., adding a new class) failed.

7.1.2.3 Ontology Editing and Visualization Components

To support the workflow activities (c.f. *activities requirements* in section 6.1.2) we rely on the NeOn Toolkit, which comes with an ontology editor that allows the

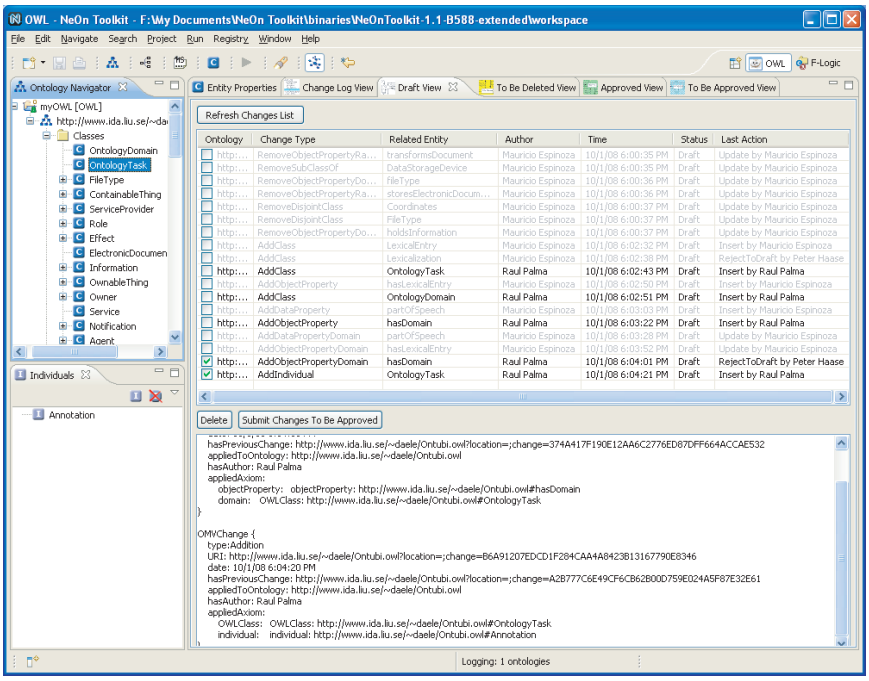


Figure 7.4: Draft View in the NeOn Toolkit

editing of ontology elements. Additionally, according to the *visualization and activities requirements* (c.f. section 6.1.2) the NeOn Toolkit is extended with a set of views that allow editors (i) to visualize the appropriate information of ontologies developed following a collaborative editorial workflow and (ii) to perform (as described in 6.2) the applicable workflow actions (*approve, reject*, etc.), depending on their role. There are four views[5]:

- *Draft view*: Shows all proposed changes (from all editors) to the monitored ontologies. In accordance to our workflow specification, the changes of the current editor are editable while changes from other editors are non editable (see Figure 7.4).
- *Approved view*: Shows the approved changes (from all editors).
- *To Be Approved view*: Shows all changes (from all editors) pending to be approved.
- *To Be Deleted view*: Shows all proposed deletions (from all editors).

[5]Subject experts can see the first two views while validators can see the last three

7.1.2.4 Distributed Registry

Ontologies are stored within a repository and as we described in 7.1.1, their metadata is managed by Oyster distributed registry (c.f. *change management requirements* in section 6.1.2). The metadata includes information about ontologies and users (represented using OMV core), the changes to the ontology (represented using the generic change ontology and/or its specializations) and about the actions performed (represented using the workflow ontology). For each change, the state is also kept to support the collaborative editorial workflow. Oyster implements the strategies described in section 5.2 as an engineering component. So, when a new change is registered into an Oyster node, Oyster automatically updates the log history keeping track of the chronological order of changes. In particular, it performs the following actions:

- gets the last registered change (using the "Log" class)
- adds it as the previous change of the current one
- updates the "Log" class to point to the current change

Additionally, Oyster implements the strategies for the propagation of changes to distributed copies of the ontology described in section 5.3.1, thus allowing the notification of new changes to ontology editors (c.f. *change management requirements* in section 6.1.2). That is, once we have the required changes stored in a machine-understandable format, Oyster is responsible for propagating them to the distributed copies of the ontology. Note that changes are propagated to each node in the distributed network that maintains a copy of the ontology (and wants to receive those changes). Hence, for this task Oyster implements the synchronization process described in 5.3.1, i.e., combination of a push and pull mechanisms, as follows: nodes periodically[6] contact other nodes in the network to exchange updated information (pull changes) and, optionally, they can push their changes to a specific node (called the push node) so that if a node goes offline before all other nodes pull the new changes, the node changes are not lost. In this way, Oyster minimizes conflicts or inconsistencies due to concurrent editing, as it automatically synchronizes changes periodically (and it allows forcing the synchronization immediately) in the distributed environment; therefore, every editor will have an up-to-date copy of the ontology with the changes proposed (c.f. *concurrency control and conflict resolution requirement* in section 6.1.2). Nevertheless, conflicts in the collaborative ontology development could still occur as logical conflicts in the form of inconsistencies or conflicts due to concurrent editing of an ontology. Explicit mechanisms for the identification and resolution of those conflicts are out of the scope of this thesis (see restriction **R2** in section 3.4). However, we refer the reader to [PHWd07] for a discussion on how to deal with those potential problems.

[6] By default every two minutes but it is configurable

The previous description of our registry and framework show how the management of changes is driven by ontologies in our implementation (verifying hypothesis H2 cf. 3.4). That is, (i) based on OMV, we can use Oyster to access the ontology metadata and determine *whether or not an ontology has changed along with a general overview of the changes*; (ii) using our change ontology, we can know *which* were the specific changes applied to the ontology; (iii) using our workflow ontology we can know *how* the changes were applied.

7.1.2.5 Configurations of the Framework to Support Identified Scenarios

Next, we describe how our framework can be configured to support the scenarios identified in section 6.1. Additionally, we discuss the benefits and drawbacks of each scenario and provide some recommendations on where to use each of them. Note that the strategies for the propagation of changes to distributed copies of an ontology supports a distributed control during the collaborative ontology development and maintenance (e.g., scenario B and C), verifying hypothesis H11 (cf. 3.4).

Configuration for Scenario A The configuration of the framework components for scenario A is depicted in Figure 7.5. Below we explain this configuration in detail.

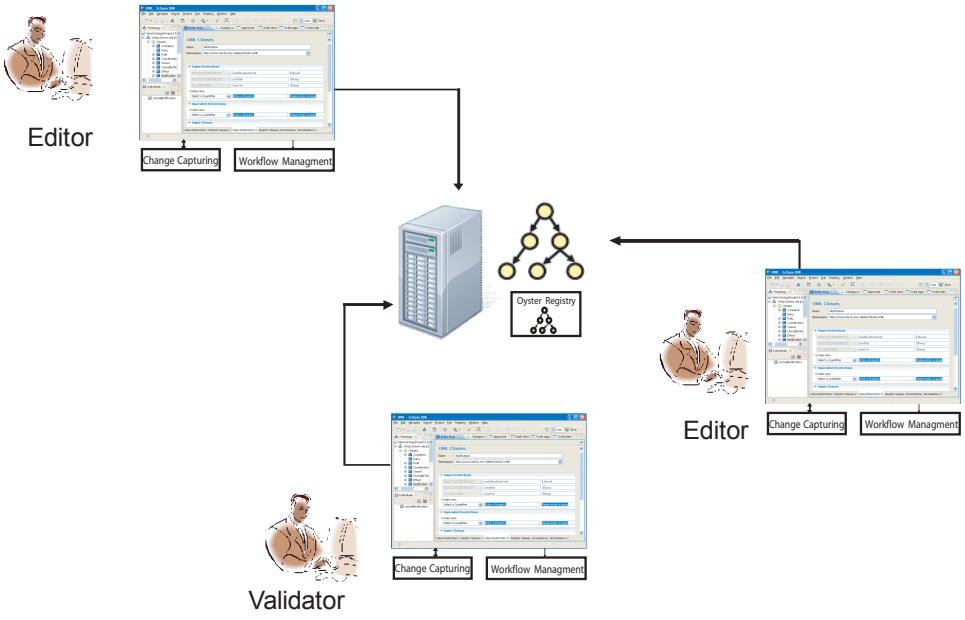


Figure 7.5: Framework Configuration for Collaborative Scenario A

Configuration of machines

- One PC is configured as the server. This PC has to run the NeOn collaboration server.
- One PC (probably the same as above) is configured as the metadata provider (also known as the "super-node"). This PC has to run Oyster (it is an Oyster node). In this scenario, Oyster is running in server mode, although it can be running in any of the possible ways, e.g., within a NeOn Toolkit installation, as the Oyster java GUI, or within any other application via its API. For additional information about the push-node see Section 7.1.2.4.
- The PCs used by the ontology editors only run a NeOn Toolkit installation configured as described above.

Key benefits of configuration A

- Ontology editors are able to work concurrently. The NeOn collaboration server allows multiple clients for the same ontology at the same time.
- Conflicts are automatically handled by the NeOn collaboration server.

Drawbacks of configuration A

- The collaboration server has to be online at every moment or ontology editors will not be able to work.
- If the collaboration server (or the metadata provider) becomes permanently unavailable, the ontology (or the metadata) is lost.
- The network connection between the clients (PCs running NeOn Toolkit) and the collaboration server has to be fast and reliable because clients communicate with the server at every moment they maintain the ontology.
- The collaboration server does not provide (at this moment) relevant information regarding the provenance of ontology changes or identity of clients.

Recommended use of configuration A

This configuration is the most suitable for those cases in which the members of the team of ontology editors are closely located (e.g., organization LAN) and there are many ontology editors usually working at similar times.

Configuration for Scenario B The configuration of the framework components for scenario B is depicted in Figure 7.6. Below we explain this configuration in detail.

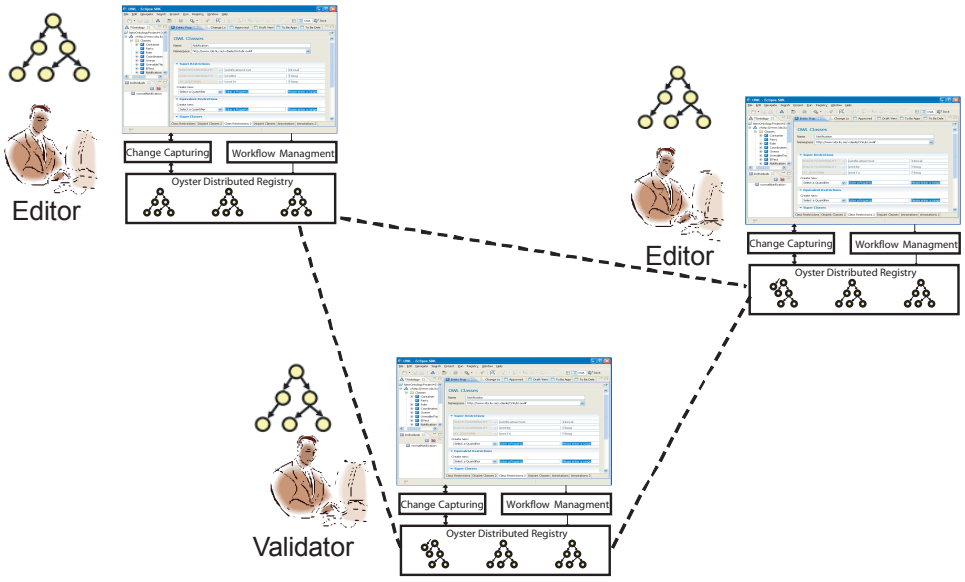


Figure 7.6: Framework Configuration for Collaborative Scenario B

Machines configuration

- The PCs used by the ontology editors run a NeOn Toolkit installation.
- Each PC runs the ontology registry Oyster, i.e., each PC is an Oyster node.
- Optionally, one PC is configured as the ”push-node”. This PC runs Oyster. In this scenario, Oyster is running in server mode, although it can be running in any of the possible ways, e.g., within a NeOn Toolkit installation, as the Oyster java GUI, or within any other application via its API. The push-node guaranties that no changes are lost even if the clients are going frequently offline (see section 7.1.2.4 for additional information about the push-node).

Key benefits of configuration B

- Ontology editors do not need a permanent network connection. They are working locally.
- The network connection does not have to be fast. It is used only during the synchronization process.
- There is no one single point of failure. The ontology (and metadata) is available in many different locations, i.e., each PC.
- Ontology editors can work independently. They do not need to have another machine online (e.g., server) to work.

Drawbacks of configuration B

- There is no support for conflict resolution (at this moment); therefore, ontology editors may encounter those problems when working concurrently.

Recommended use of configuration B

This configuration is the most suitable for those cases in which the members of the team of ontology editors are geographically distributed (e.g., different organizations and different countries), without a reliable and fast network connection, and they do not work usually at similar times (e.g., when people work in different countries with different time zones, or when they coordinate to work at different times).

Configuration for Scenario C The configuration of the framework components for scenario C is depicted in Figure 7.7. Below we explain this configuration in detail.

Machines configuration

As we described in section 6.1, this scenario is a hybrid between scenarios ”A” and ”B”, where the team of ontology editors can be divided in at least two different groups: ”X” and ”Y”, each of them having the characteristics of scenario ”A”. Additionally, the groups have between them an environment with the characteristics of scenario ”B”. Hence, each of these groups has the machine configuration of scenario ”A”, and the groups have between them the machine configuration of scenario ”B” described above.

Key benefits of configuration C

- The groups do not need a permanent network connection between each other.
- The network connection does not have to be fast between the groups. It is used only during the synchronization process.
- There is no one single point of failure. The ontology (and metadata) is available in many different locations, i.e., each collaboration server (and metadata provider).
- Ontology editors of each group can work independently.
- Ontology editors are able to work concurrently within each group. The NeOn collaboration server allows multiple clients of the same ontology at the same time.
- Conflicts are automatically handled by the NeOn collaboration server within each group.

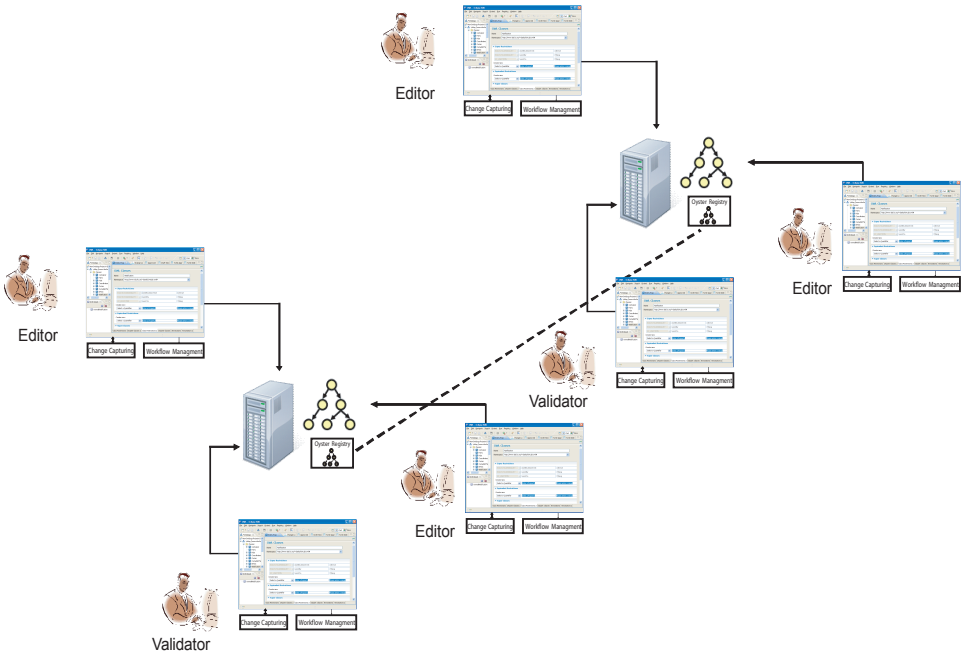


Figure 7.7: Framework Configuration for Collaborative Scenario C

Drawbacks of configuration C

- There is no support for conflict resolution between groups (at this moment); therefore, groups may encounter those problems when working concurrently.
- The collaboration server has to be online at each group every moment or ontology editors will not be able to work.
- The network connection between the clients (PCs running NeOn Toolkit) and the collaboration server in each group has to be fast and reliable because clients communicate with the server at every moment they maintain the ontology.
- The collaboration server in each group does not provide (at this moment) relevant information regarding the provenance of ontology changes or identity of clients.

Recommended use of configuration C

This configuration is the most suitable for those cases in which the members of the team of ontology editors are affiliated to different parties. At each party, the ontology editors are closely located (e.g., organization LAN) and there are many ontology editors usually working at similar times. Additionally, the parties are geographically distributed (e.g., different organizations and different countries),

without a reliable and fast network connection, and they do not work usually at similar times (e.g., when people work in different countries with different time zones, or when they coordinate to work at different times). A typical example is a team composed of ontology editors from different organizations, e.g., two universities. Within each university the editors work at similar times but each university has a different schedule.

7.2 Applicability of OMV

Since the main goal of OMV is to become a community-accepted vocabulary for describing ontologies and related entities, we decided to evaluate its applicability. According to the Merriam-Webster Online Dictionary [7], applicability refers to being capable of or suitable for being applied. Hence in our context, applicability refers to the suitability of OMV for describing ontologies and related entities. As a consequence, the best indicator is its acceptance and usage in the semantic web community.

OMV is listed as the second top downloaded ontology from the ontology engineering platform OntoWare[8]. Only the OMV core ontology has been downloaded around 34 thousand times, including all versions, since it was first made public (May-2005). Table 7.2 shows the total amount of downloads for each version of the OMV core, and the release date for each of them.

Table 7.2: OMV Core ontology downloads

Version	Release Date	Total
0.6	13-May-2005	727
0.7	25-July-2005	921
0.9.1	15-November-2005	765
0.9.2	21-November-2005	744
1.0	02-February-2006	1072
2.0	06-June-2006	1061
2.1	01-June-2007	581
2.2	10-August-2007	750
2.3	19-September-2007	26674
2.4	08-February-2008	614

Note that there is a huge difference between the downloads of version 2.3 and the rest of them. This is because that version was stable during a long period before the new update 2.4 was released, and different applications were developed and tested based on that version. In any case, this data is an indicator that OMV is a popular ontology, specially if we analyze the usage summary for the OMV site

[7] http://www.merriam-webster.com/dictionary/applicability
[8] http://ontoware.org/

generated by OntoWare with Webalizer Version 2.01[9] for the last year (see Table 7.3).

Table 7.3: Usage summary for omv.ontoware.org (April-08 to Mar-2009)

Measure	Avg per Month	Total
Hits	8882.25	106587
Sites	402.5	4830
Visits	833.3	10006
Files	2395.75	28749

According to Webalizer, the measures are defined as follows:

- Hits refer to any request made to the server which is logged. The requests can be for anything (e.g., html pages, graphic images and files).
- Sites show how many unique IP addresses made requests to the server.
- Visits refer to how many requests are made to the server from a given site after a timeout value (default 30 min) plus visits from new sites.
- Files refer to how many times the server sent something back to the requesting client, such as a html page or graphic image.

From the previous data, we see that the OMV site is in general a very active one with around 300 hits per day. Additionally, when we went deeper in the analysis of the hits, we found out that in average there are hits from more than 50 different countries around the world per month.

To be sure that people visiting our omv site are actually interested in the OMV metadata model, we analyzed how many downloads were of the documentation, i.e., the technical report (see Figure 7.8). The total number of downloads is 3262 with an average of 466 per version. This data confirms that many people have been seriously interested in OMV.

Also, another indicator of the acceptance of OMV is the number of institutions that have provided feedback, suggestions or contributions to the refinement of the metadata model (in the current version we have included comments from 6 different institutions (see OMV Technical Report)), and some other organizations have expressed interest in using it. For instance, Stanford BMIR is planning to use OMV in Protégé[10] and their Bioportal ontology repository[11]. Also the Ontology PSIG[12] of the Object Management Group (OMG) together with other OMG subgroups are evaluating on using OMV in their repository. As another example, the Ontology Summit 2008 Communiqué: Towards an Open Ontology Repository, reports that

[9] http://www.mrunix.net/webalizer/
[10] http://protege.stanford.edu/
[11] http://bioportal.bioontology.org/
[12] http://www.omg.org/ontology/

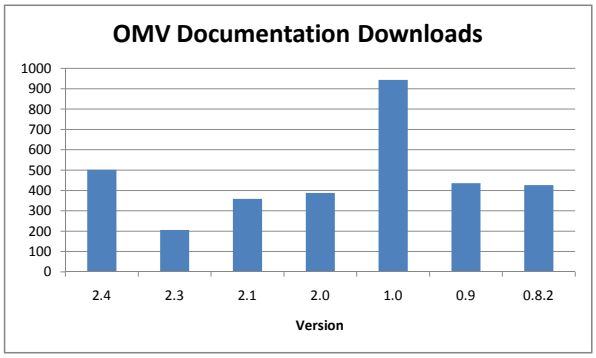


Figure 7.8: Downloads per Version

"*The Ontology Metadata Vocabulary (OMV) http://omv.ontoware.org/ is a strong candidate for describing ontologies in the OOR*"[13]

On the other hand, there exist different implementations and tools support for OMV. For example, the Semantic Web Gateway, Watson[14], provides OMV annotations for the ontologies discovered. The ontology repository Cupboard[15] uses OMV to provide advanced semantic search capabilities of ontologies. The commercial registry Centrasite[16] provides an OMV specialized web service to support the management of ontology metadata. The distributed registry Oyster that we described in section 7.1.1 is based on OMV.

Hence the previous discussion shows that **OMV has become a community-accepted vocabulary for describing ontologies and related entities**, verifying hypothesis H4 (cf. 3.4).

Finally, **OMV has also been extended for different applications/scenarios**, which verifies hypothesis H5 (cf. 3.4). As described in section 4.2.6, currently we have five extensions (in stable or testing stage)[17]:

- Mappings extension
- Changes extension
- Peer extension
- Multilinguality extension
- Modules extension

[13] `http://ontolog.cim3.net/cgi-bin/wiki.pl?OntologySummit2008_Communique/Draft`

[14] `http://watson.kmi.open.ac.uk/WatsonWUI/`

[15] `http://kmi-web06.open.ac.uk:8081/cupboard`

[16] `http://www.infoq.com/zones/centrasite/`

[17] OMV Extensions (in stable stage) are also available at `http://omv.ontoware.org`

7.3 Completeness of the change representation model with respect to the OWL 2 ontology language

To measure the completeness of the change representation model, we programmatically provided the system[18] with a set of changes consisting of at least one for each possible axiom and at least one associated for every possible type of entity/description. In particular, we simulated programmatically the changes in a test ontology[19] and let our system capture and represent each change, i.e., to generate the change ontology individuals. However, it was not possible to simulate all of them because a few are not yet supported by the NeOn Toolkit. For those few changes, we had to register them directly into the registry (programmatically) to simulate their execution to the ontology. In detail, we could simulate the following set of 52 changes as if they had happened in the ontology:

- 20 class axioms
  - (4) one for each possible axiom
  - (16) one for each possible description, except for owlClass because it was used in the previous 4 and objectExistsSelf because KAON2 does not serialize it correctly and the OWL file becomes corrupt).
- 5 declaration axioms
  - (4) one for each possible entity, except for datatype declaration that is not supported.
  - (1) one for the special type of entity called AnnotationProperty. It is mentioned in the specification but it was not still officially a type of entity.
- 5 dataProperty axioms (one for each possible axiom, except disjointDataProperties which is not supported).
- 12 objectProperty axioms (one for each possible axiom, except disjointObjectProperties which is not supported).
- 5 fact axioms (one for each possible axiom, except for negativeObjectPropertyAssertion and negativeDataPropertyAssertion which are not supported).
- 5 entityAnnotation axioms (one for each possible entity)

Therefore, the set of 6 changes that we had to register directly into the registry (programmatically) was:

[18] The code is available as a JUNIT test (AllOWLChangesTest.java) included in the Change Logging plugin release

[19] For the test we used the pizza ontology `http://www.co-ode.org/ontologies/pizza/pizza_20041007.owl`

- one class axiom associated to a objectExistsSelf description.
- Declaration of datatype.
- disjointDataProperties.
- disjointObjectProperties.
- negativeObjectPropertyAssertion.
- negativeDataPropertyAssertion.

After the test was completed, we verified that each of the 58 changes could be correctly represented in our representation model. For this task, we visualized the contents of the registry (190 KB) (i.e., the captured changes) from the Change Log View (c.f. 7.3) and the Draft View (c.f. 7.4) of our infrastructure. **The conclusion is that every change was successfully represented in our model**, verifying hypothesis H8 (cf. 3.4). That is, for each of the 58 changes, we had 58 correct individuals of the `AtomicChange` concept of our model and 57 correct individuals of the `EntityChange` concept of our model. Note that the relationship between `AtomicChange` and `EntityChange` is almost 1 to 1 in this scenario. This is due to the way changes are captured and notified within the NeOn toolkit. The only `AtomicChange` that did not have a corresponding `EntityChange` was the declaration of individual, because within the NeOn toolkit a declaration of an individual is followed always by the assertion of this individual as a member of a class.

Finally, to verify if the changes that could be made programmatically to the ontology were successfully executed, we opened the resulting ontology[20] within the NeOn toolkit. The conclusion was that all changes made (and that could be displayed by the GUI of the NeOn toolkit) to the ontology were successfully executed. Some changes such as adding the axiom asymetricObjectProperty are correctly captured in the OWL file but it is not possible to visualize them in the GUI at this moment.

7.4 Experimentation

In order to evaluate the conceptual and technological contributions of this thesis, we conducted an experiment in a real case scenario at the Food and Agricultural Organization[21] (FAO).

Following the phases considered in most software experimentation approaches ([BSH86], [Pfl95], [KPP+02]), the experiment was performed in the following three consecutive phases:

[20] Available at `http://torresq.dia.fi.upm.es/neon/results/pizzaAll.owl`.
[21] `http://www.fao.org`

- Plan phase: describes the definition and design of the experiment (see 7.4.1)
- Experiment phase: describes the experiment execution (see 7.4.2)
- Analysis phase: describes the analysis of the experiment results (see 7.4.3)

7.4.1 Plan Phase

We present in section 7.4.1.1 the description of the motivations, constraints and goals of the experiment, and identify its subject and beneficiaries. Then, in section 7.4.1.2, we describe the relevant characteristics of experiment, followed by the metrics and criteria used to evaluate those characteristics. Then, we analyze the variables involved during the experiment, and introduce its requirements and the processes for the collection and analysis of the data.

7.4.1.1 Experiment Definition

Motivation

The main motivation of this experiment was to evaluate the models, methods and strategies proposed in this thesis for the management of ontology changes to support the development and maintenance of ontologies in a collaborative scenario. Moreover, the experiment allowed us to asses hypotheses **H3**, **H9**, **H10** and **H13** of this thesis (see section 3.4).

Constraints

The framework for collaborative ontology development is constrained to be used by organizations with a well-defined process that coordinates the development of ontologies collaboratively. The users, which are usually geographically distributed, are required to belong to the same organization, i.e., they know each other, and have associated permissions to perform some tasks based on their expertise and responsibilities. A representative organization for this scenario is FAO.

Goals

The goal of the experiment was to evaluate the benefits of using the infrastructure that implements the models, methods and strategies described in this thesis. Based on the management of ontology changes, this infrastructure supports the development and maintenance of ontologies collaboratively by a team of ontology editors with different roles and following a well-defined process for the proposal of ontology changes.

Beneficiaries of the experiment

Ontology editors at any organization following a well-defined process in the development and maintenance of ontologies.

Experiment subject

In particular, we evaluated the following items:

- The change representation model
- The workflow model
- The system implementation

7.4.1.2 Experiment Design

Relevant characteristics

We studied, on the one hand, the conceptual models that provide the foundations to represent the required information and, on the other hand, the implementation support. The following attributes were studied:

1. Conceptual Models
    - Change representation model:
        - The adequacy with respect to the users' requirements
    - Workflow model
        - The adequacy of the model with respect to the users' actions
2. System Implementation
    - The overall usability and performance of the system. According to the ISO standard 9241-11[ISO98], usability[22] refers to:
        - Effectiveness
        - Efficiency
        - User satisfaction

Metrics and criteria

To evaluate the aforementioned characteristics, we used the following metrics and criteria (summarized in Table 7.4):

- To measure the adequacy of the change representation model we analyzed the set of changes that ontology editors are usually required to perform. For every change proposed, we verified that it could be represented by our model and that it captured all the information required by the ontology editors.
- To measure the adequacy of the workflow model we analyzed if ontology editors were able to perform all of the required workflow actions. For each possible action, we verified that it could be represented with our model and that it captured all the information required by the ontology editors.

[22]" The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use"

- To measure the effectiveness of the system we requested ontology editors to perform a set of tasks and we analyzed:
  - Percentage of tasks completed
  - Ratio of successes to failures
  - Number of features or commands used
- To measure the efficiency of the system we used the same set of tasks as above and we determined:
  - Time to complete a task:
    - Change the ontology: Time required to perform all requested changes to the ontology taking into account the number of proposed changes.
    - Complete process: All the required actions (and proposed changes) to change one ontology from one stable version to another stable version, i.e., the time to complete all tasks of the experiment, taking into account the number of proposed changes.
  - Time to learn how to use the system.
  - Percentage or number of errors.
  - Frequency of help or documentation use.
- To measure the user satisfaction, we used as basis the Software Usability Measurement Inventory[23] (SUMI) which is a rigorously tested and proven method of measuring software quality from the end user's point of view. SUMI[vV98] studies the following 5 dimensions:
  - Efficiency: it measures the degree to which users feel that the software assists them in their work and is related to the concept of transparency.
  - Affect: it measures the user's general emotional reaction to the software (also known as Likeability).
  - Helpfulness: it measures the degree to which the software is self-explanatory, as well as more specific things like the adequacy of help facilities and documentation.
  - Control: it measures the extent to which the user feels in control of the software, as opposed to being controlled by the software, when carrying out the task.
  - Learnability: it measures the speed and facility with which the user feels that they have been able to master the system or to learn how to use new features when necessary.

[23] `http://sumi.ucc.ie/index.html`

Hence, for this task we requested ontology editors to fulfill an online survey (at `http://www.surveymonkey.com/`) which included the de facto industry standard evaluation SUMI questionnaire consisting of 50 statements (10 for each dimension) to which the user had to reply either *Agree, Don't know, or Disagree*. The second option (*Don't know*) is selected if the user is undecided, cannot make up his mind, or if the statement has no relevance to the software or to the situation. Furthermore, in addition to the standard SUMI questions, the survey included 10 questions specific to the collaborative ontology development process to assess the user's satisfaction when using the infrastructure for this purpose.

Table 7.4: Metrics and criteria for characteristics evaluated in experiment

Metric	Criteria
Adequacy of the change representation model	From a set of typical changes performed by ontology editors, to verify that each change could be represented by our model and that it captured all the information required by the ontology editors.
Adequacy of the workflow model	To analyze if ontology editors were able to perform all of the required workflow actions and that each action could be represented with our model capturing all the information required by the ontology editors.
Effectiveness of the system	From a set of tasks performed by ontology editors, to analyze the percentage of tasks completed, the ratio of successes to failures and the number of features or commands used.
Efficiency of the system	From a set of tasks performed by ontology editors, to analyze the time to complete a task, the time to learn how to use the system, the percentage or number of errors and the frequency of help or documentation use.
User satisfaction	Measure software quality from the end users point of view, including the five SUMI dimensions (efficiency, affect, helpfulness, control and learnability) and the collaborative ontology development process perception.

Variables

We identified the following controlled and not controlled variables:

1. Controlled

   - The experience and the background level of the ontology editors performing the experiments. Ontology editors were chosen from two major groups, Subject Experts and Validators. Subject experts know about specific aspects of the ontology domain and are in charge of adding or modifying ontology content. They usually know little or nothing about

ontology software or design issues. Validators revise, approve or reject changes made by subject experts, and they are the only ones who can copy changes into the production environment for external availability. They have a broader knowledge of the ontology domain and have at least some knowledge about design issues.

2. Not controlled
    - The learning capabilities of the ontology editors could affect the result, but due to time and the ontology editors' availability restrictions, it was not used as a factor of the experiment.
    - Similarly, the motivation that ontology editors had to learn a new system could affect the result. In this case the effect was at leat reduced by an appropriate introduction of the system and the expected improvements.

Data collection process

To collect the required data regarding the adequacy of our models and the overall usability and performance of the system, we conducted an experiment with a set of appropriate users. In particular, the experiment was performed by a team of representative users from FAO who carried out a series of representative tasks as follows:

- The team was provided with a stable version of a fishery ontology (v1).
- They were asked to perform collaboratively a set of typical changes and actions to the ontology in order to reach a new stable version (v2). The tasks included:
    - Adding new ontology elements;
    - Modifying ontology elements;
    - Deleting ontology elements;
    - Approving proposed changes (including the previous required actions);
    - Rejecting proposed changes (including the previous required actions);
- Users were asked to visualize all the information recorded by the system for each of the performed changes and actions and to confirm that all the required information is available.
- One person (the tester) recorded and documented the time users took to accomplish the different tasks, the possible problems/errors and the users' comments during the process.
- At the end of the experiment, the editors were presented with a survey to evaluate their experience in performing the requested tasks.
- All the results were stored and documented by the tester person.

Requirements

At leat one complete execution of the experiment with a team of ontology editors working collaboratively on the maintenance of an ontology. The team was required to include at least one ontology editor for each possible role in the workflow, i.e., one Subject Expert and one Validator. Additionally, the ontology used for the experiment was required to be one of the most frequently used fishery ontologies at FAO.

Analysis procedure

The collected data was validated to ensure that the ontology editors performed the required actions to reach the new stable version of the ontology (v2). The analysis of the data was performed with a combination of manual and automatic techniques. We analyzed if our models were adequate to represent the corresponding changes and actions based on the user's feedback, and we computed the required ratios, percentages and times (e.g., time to complete a specific task) summarizing the individual results. We also used some of the automatic reporting tools (e.g., `http://www.surveymonkey.com/`) to analyze the results of the surveys about the user's satisfaction. We analyzed the possible reasons that affected those results and we compared them with the current time and efforts that take to FAO ontology editors to accomplish the same task without the proposed infrastructure.

7.4.2 Experiment Phase

The experiment was conducted at FAO headquarters during the last week of October, 2008. We needed two days, one for the set-up of the collaborative infrastructure in FAO computers and one for running the experiment. During the first day, following the typical behavior of the ontology editors in FAO and because of time constraints, the tester configured the collaborative infrastructure as the configuration A described in 7.1.2.5. The particular configuration was:

- One server running:
  - NeOn Collaboration Server
  - Oyster (server mode)
- Three clients, each running:
  - NeOn Toolkit extended with:
    - Registry Plugins
    - Change Management Plugin
    - Collaboration Plugins

Table 7.5: Software versions of framework components used for the experiment

Software	Version
NeOn Collaboration Server	OntoBroker-Enterprise-5.2-B719
NeOn Toolkit	NeOnToolkit-1.2.0-B766-extended
Oyster	Oyster2APIv2.3.1
Registry Plugins	org.neontoolkit.oyster.plugin.menu-1.8.0.jar org.neontoolkit.registry.api-2.3.0.jar
Change Management Plugins	org.neontoolkit.changelogging-1.8.0.jar
Collaboration Plugins	org.neontoolkit.collab.preference-1.8.0.jar org.neontoolkit.collab-1.8.0.jar

The software versions used for the experiment are summarized in table 7.5.

It is important to note that the version of the NeOn collaboration server used for the experiment has one critical limitation: when a change is performed on an ontology managed by the server, no provenance information is available when the event is fired, i.e., it is not possible to know from which client the change was originally performed. Nevertheless, our infrastructure provides a temporary solution to overcome this problem. The only limitation is that two (or more) editors should not work with the same ontology element at the same time; otherwise, a change on that element will be logged once for each of them.

The ontology used for the experiment was species-v1.0-model.owl[24]. This ontology is the schema of one of the most important ontologies of FAO for the fishery domain. The tester uploaded this ontology to the server as part of the configuration.

On the second day, before running the actual experiment, the tester gave a brief introduction (30 min) to the system and the goal of the experiment to the FAO team composed of three ontology editors (Subject Expert A, Subject Expert B and Validator A). Each of the editors were in charge of maintaining the ontologies of the fishery domain, and they had a different background profile:

- Subject Expert A had a great knowledge about the fishery domain, but he had never used the NeOn toolkit and in general he had a small knowledge of computer systems or modeling design issues.
- Subject Expert B had a fair knowledge about the fishery domain and the NeOn toolkit and some knowledge about modeling design issues.
- Validator A also had a fair knowledge about the fishery domain, the NeOn toolkit, and also about modeling design issues.

All ontology editors were in the same room and each was provided with a detailed and personalized guide of the tasks he had to perform including the initialization of his NeOn Toolkit installation, i.e., each of them had to configure his client as in a real situation. The three complete guides are available in Appendix

[24] Available at `http://www.fao.org/aims/neon.jsp`

A. In a nutshell, each subject expert (SE) had to perform 6 main tasks while the validator (V) had to perform 3 main tasks, as follows:

- Every ontology editor was requested to configure and start the collaboration support within his NeOn toolkit (T1).
- Next, each subject expert was requested to make several changes to the ontology concurrently (SE's-T2). The chosen changes were 34 (17 changes for each SE) realistic modifications to the ontology including real information according to FAO experts. Examples of those changes are:
  - To add Individual 31005-10001 (Species).
  - To add Individual 31005-10000 DataProperty hasCodeAlpha3 value: DCR. Type: string.
  - To add Individual 31005-10001 DataProperty hasNameScientific value: Pterodroma wrong macroptera. Type: string.
  - To add Root Class Speciation.
  - To add ObjectProperty hasScientificNameAuthor.
- Then, subject experts had to visualize the results of their changes and analyze the information provided by the system (SE's-T3).
- Next, subject experts were requested to submit their changes to be approved (SE's-T4).
- Then, the validator was requested to analyze the changes performed and to approve/reject them (V-T2).
- The subject experts were then requested to perform some additional actions according to the workflow to test the possible subject expert actions (e.g., delete a rejected change and modify an approved change) (SE's-T5 and T6).
- Finally, the validator was also requested to perform some additional actions in order to test the possible validator actions (e.g., reject to be approved a change and delete an approved change) (V-T3).

As we explained in the previous section (see section 7.4.1.2), during the experiment the tester was taking note of the behavior of the editors, their questions and problems, and at the end of the experiment, each editor fulfilled an online survey (see Figure 7.9) consisting of 60 questions (50 of the standard SUMI questionnaire and 10 specific for the collaborative ontology development). The complete survey is available in Appendix B. Here are some example statements:

- This software responds too slowly to inputs (SUMI).
- The way that system information is presented is clear and understandable (SUMI).

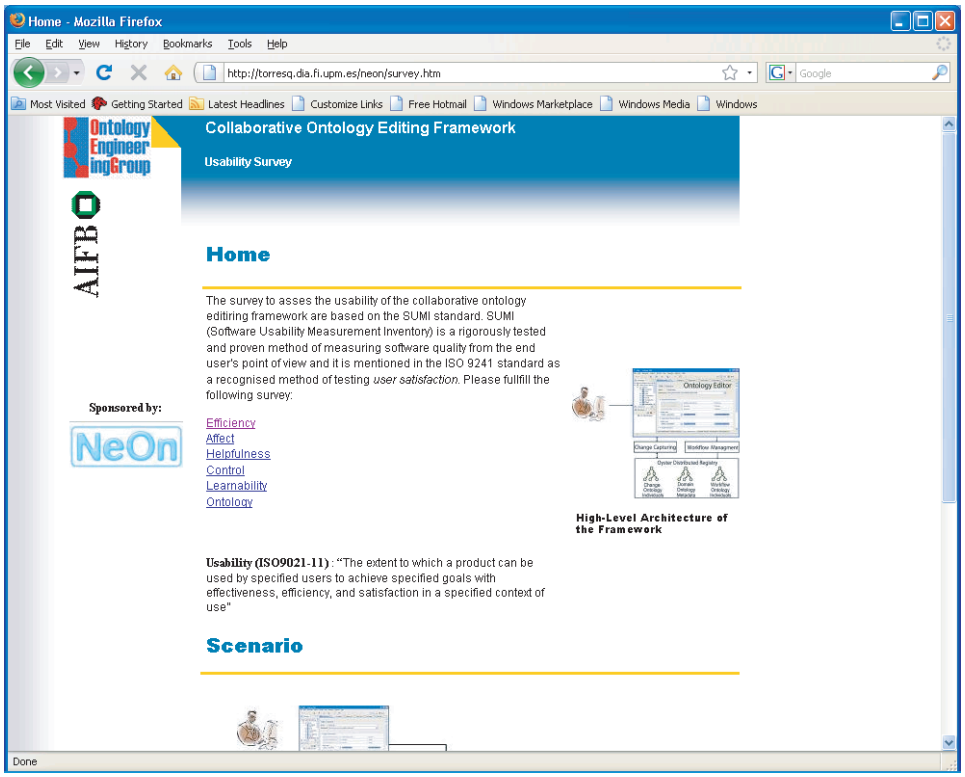


Figure 7.9: Online Survey

- I can understand and act on the information provided by this software (SUMI).
- The time to build an ontology collaboratively decreases with this software (Collaborative Ontology Development).
- The information captured for the ontology changes is enough (Collaborative Ontology Development).

The total time taken to complete the experiment was about two hours.

7.4.3 Analysis Phase

Next, we present in sections 7.4.3.1 and 7.4.3.2 the analysis of the adequacy of the change representation model and the workflow model (respectively), with respect to the ontology editors' needs. Then, in sections 7.4.3.3 and 7.4.3.4, we analyze the effectiveness and efficiency of the system (respectively). Finally, in section 7.4.3.5 we present the analysis of the users' satisfaction regarding the usability and performance of the system.

7.4.3.1 Adequacy of the change representation model with respect to the ontology editors' needs

To measure the adequacy of the change representation model, we requested the ontology editors of our case study at FAO to perform a set of representative changes and asked them to analyze the captured information for the change. The chosen set of changes was carefully selected with the cooperation of FAO experts to reflect valid modifications and using real data.

Based on those changes, we verified during the experiment that none of the ontology editors had problems to perform the changes to the ontology, and consequently that **all changes proposed could be captured by our representation model**.

Additionally, the survey that ontology editors had to fulfil after the experiment (see previous section) included two questions to verify the adequacy of the model from the ontology editors' perspective. From the three possible answers in the survey (*Agree, Don't know, Disagree*), we are mainly interested in the first and the third one. According to the three possible meanings of the option *Don't know* explained above (see 7.4.1.2), in this particular situation it could only mean that either the user is undecided or that he cannot make up his mind, because the statement is relevant to the software and the situation. Consequently, if the user did not provide additional feedback or comments, we ignored it. Next, we analyze the results of the two statements:

- *Some changes were not captured correctly*. The result of this statement shows (see Figure 7.10) that nobody agreed with it, one person did not know and the other two disagreed. Since, there was no additional feedback from the undecided user, we can affirm that **all changes proposed could be captured by our representation model** because nobody disagreed.

- *The information captured for the ontology changes is enough*. In this case, the result of the statement shows (see Figure 7.11) that one person agreed, one did not know and one disagreed. Similar to the previous statement, the undecided user did not give any additional feedback and so, we concentrate in the negative answer. For this survey item, we explicitly requested users to provide some feedback in the case they disagreed. The comments from the user were the following (textually):

  - "It would be nice to have some formatting, it's rather hard to read and takes a bit to figure out what the change actually was".
  - "It would be nice if you could click on a change and go to the relevant part of the ontology".
  - "The Change log view lacks a column present in the other views that tells you the concept or property associated to the change".

  As we can see from the comments, the user was providing feedback regarding the visualization of the changes rather than the information captured for

the changes, e.g., the author, the date and time, the related entity and others. Therefore, as we do not have any request for additional information that should be captured for the changes, we can argue that **the information captured for the ontology changes is enough**.

The previous analysis provides positive evidence for hypothesis H10 (cf. 3.4), with respect to the adequacy of the model for the representation of changes.

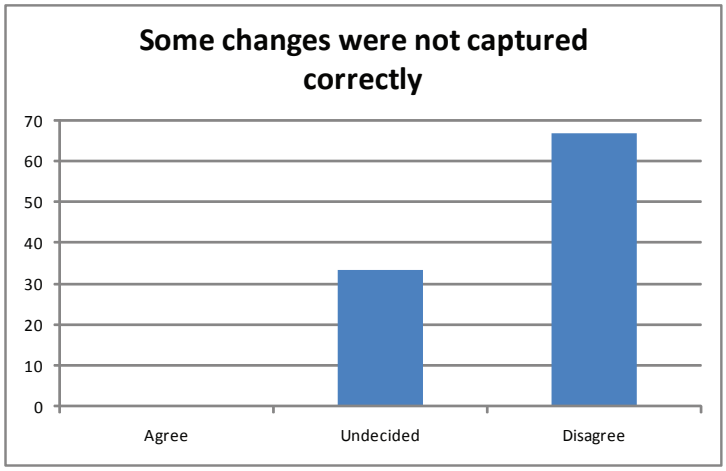


Figure 7.10: Survey Question 6-4

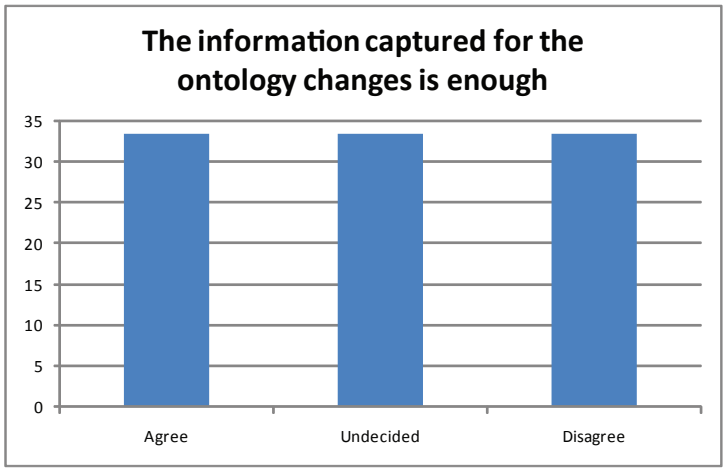


Figure 7.11: Survey Question 6-5

7.4.3.2 Adequacy of the workflow model with respect to the ontology editors' needs

Similarly to the previous metric, to measure the adequacy of the workflow model, we requested that the ontology editors of our case study at FAO perform every possible action according to their role, and asked them to verify that their actions were correctly executed, i.e., to see the consequences of their actions (e.g., when a change in draft state is sent to be approved, it should no longer appear in the list of draft changes) which in turn would confirm that the action captured all the required information.

During the experiment, we verified that none of the ontology editors had problems performing the set of possible actions according to his role and, consequently, that **all workflow actions could be captured and represented by our workflow model**.

Also like in the previous metric, the survey that ontology editors had to fulfil after the experiment included two questions to verify the adequacy of the model from the ontology editors' perspective. For the same reasons explained above, we focus our attention to the *Agree* or *Disagree* answers, and ignore the *Don't know* answer if the user did not provide additional feedback or comments. In the following we analyze the results of the two statements:

- *The software allows to perform all the required actions of our workflow*. The result for this statement shows (see Figure 7.12) that everyone agreed with it. Hence, we can affirm that **all workflow actions could be captured and represented by our workflow model**. Furthermore, if all ontology editors could perform all the required actions of the workflow, this means that they could also verify the expected consequences of the actions. So, we can affirm that **actions captured all the required information**.

- *Users are able to perform all the actions they could perform according to their role and only those*. In this case, the result for the statement shows (see Figure 7.13) that nobody disagreed with it, one person did not know and the other two agreed. Since there was no additional feedback from the undecided user, we can affirm that **all workflow actions could be captured and represented by our representation model** because nobody disagreed.

The previous analysis provides positive evidence for hypothesis H9 (cf. 3.4), supporting that the collaborative process, usually followed by organizations for the management of ontology change proposals, can be modeled by means of a collaborative editorial workflow. Additionally, it also provides positive evidence for hypothesis H10 (cf. 3.4), with respect to the adequacy of the model for the collaborative process.

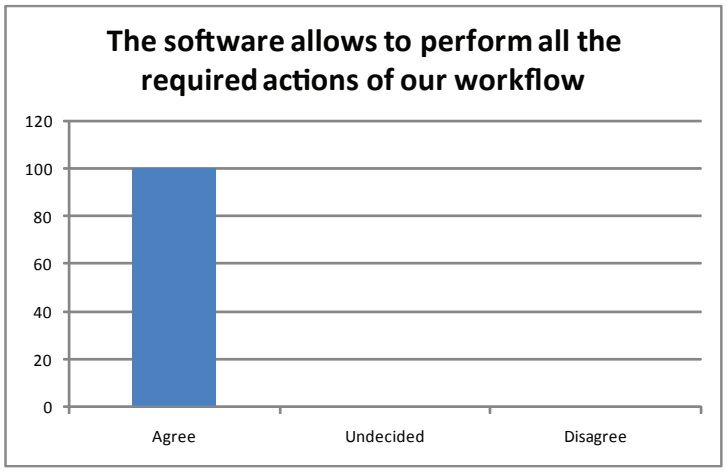


Figure 7.12: Survey Question 6-2

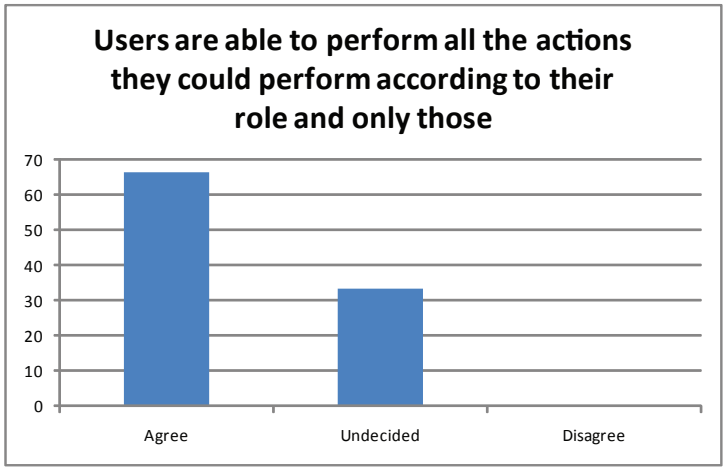


Figure 7.13: Survey Question 6-3

7.4.3.3 Effectiveness of the system

The experiment gave the following results for the effectiveness of the system:

Percentage of tasks completed

Each ontology editor completed 100% of his assigned tasks (six tasks for each subject expert and four for the validator). However, after analyzing the log of changes of the experiment, it reveals that for task T2 of subject experts (that consisted in making 17 changes to the ontology), one of the subject experts was author of only 13 of the required changes (76% individually or 92% globally). The other four changes were not present in the log, so either the user forgot to made them or there was a failure in the system. Since there was no report or complain during the

experiment that a change was not created, we assume there was no system failure. Furthermore, we found 13 changes that were not part of the guide. After analyzing each of them, we concluded that:

- Five changes were performed to provide additional information or to test other possible changes.
- Five changes were consequence of a human error (e.g., one SE created a subClass instead of a root class and some individuals were removed and then added again).
- Three changes were incorrectly generated as a consequence of the limitation of the collaboration server described in the previous section.

Ratio of successes to failures

For this measure we analyzed the following two functionalities:

- Change logging: We consider it a success if a change performed was correctly captured, represented and registered, and a failure otherwise. We analyzed the 30 logged changes (from the guide) (17 from SE1 and 13 from SE2) and we verified that all of them were successfully logged. However, three of them were incorrectly logged twice due to the NeOn collaboration server limitation. Therefore, we have a 90% rate of success.
- Workflow management: We consider it a success if a workflow action (i) was correctly created, represented and registered and (ii) had the expected consequences (e.g., adding an ontology element, should create an *insert* workflow action and set to *draft* the element state). As we explained in section 6.2, actions are performed either implicitly (e.g., add element) or explicitly (e.g., submit change to be approved). During the experiment, SE1 performed 17 implicit actions and 20 explicit ones (from the guide), SE2 performed 13 implicit actions and 16 explicit ones (from the guide) and V1 performed 7 explicit actions. The total number of actions was 73. We analyzed each of them and verified that they were a success, giving a 100% rate of success.

Number of features or commands used

During the experiment ontology editors used the following 18 features or commands (classified according to the infrastructure components):

- Registry
  - Registry support
  - Remote storage location
- Change logging

  - Log changes

- Workflow management
  - Workflow actions (9): Insert, Update, Delete, Submit to Be Approved, Submit to Approved, Submit to Be Deleted, Reject to Approve, Reject to Be Approved, Reject to Draft.
  - Enforcement of workflow constraints

- Visualization components
  - Change log view
  - Draft view
  - To Be Approved view
  - Approved view
  - To Be Deleted view

Only the following two features of the infrastructure were not tested during the experiment (2 out of 20, or 10%) due to the configuration of our scenario (configuration A described in 7.1.2.5):

- Registry
  - Synchronization of changes

- Change logging
  - Propagation of changes to local ontology

7.4.3.4 Efficiency of the system

Next, we present the results of the times analyzed based on the log of changes, the information in the registry, and the notes taken during the experiment.

Time to complete a task

As we explained in section 7.4.1.2, we analyzed two tasks, i.e., changing the ontology and the complete process. The results are:

- Change the ontology: The time required by both SEs to perform 40 changes concurrently (30 from the guide plus 10 they wished to test or had to repeat after they made a mistake) took 47 minutes and 47 seconds (47:47).

- Complete process: The time required by all ontology editors to complete the experiment, i.e., change the ontology, send changes to be approved, approve/reject changes and all additional workflow actions from the guide, was 1 hour and 50 minutes (1:50:03). During the experiment, the NeOn collaboration server crashed once, and it took around 5 minutes to restart it since it was in a remote location.

Time to learn how to use the system

As we described in the previous section, before starting the experiment we gave a brief introduction (30 minutes) to the system and the goal of the experiment to the FAO team. Then, during the experiment, users asked a few times for assistance when they had a problem or wanted to know additional information.

Percent or number of errors

During the experiment, we did not get any errors from the infrastructure. However, we got the following problems:

- The NeOn Collaboration server crashed once. After analyzing the log of the server, we found out that the problem was that the server was out of memory.
- After the server crashed, the SEs had to repeat the last change they were performing when it crashed.
- One of the SEs had to repeat his first five changes because they were not logged at all. In this case, the SE did not configure correctly his NeOn Toolkit and had to restart it.

Frequency of help or documentation use

Throughout the experiment, ontology editors asked regularly for assistance of the tester. However, in general they wanted either to ask how to do something on the ontology editor (which is provided by the NeOn toolkit not by us), or to provide some feedback. In general, they did not have problems using the features of the infrastructure. In any case, taking into account that users had only a brief introduction to the collaborative infrastructure (and the experiment), in addition to the fact that they did not use the NeOn toolkit regularly, it is totally understandable that they needed some help during the experiment.

7.4.3.5 User satisfaction

In this section we analyze the results of the 60 questions of the survey filled by the ontology editors at the end of the experiment. First, we focus on the 10 questions specific to the collaborative ontology development and then we present the results for the 5 SUMI dimensions, consisting of 10 questions each.

Collaborative Ontology Development

Out of the 10 questions of the survey specific for this matter, eight required the user to reply either *Agree, Don't know, or Disagree*. The last two questions requested written information from the user: the best and worst part of the infrastructure and any final feedback/comments.

The general impression of the infrastructure to support the collaborative ontology development process was calculated as the average of the first eight questions and it is shown in Figure 7.14.

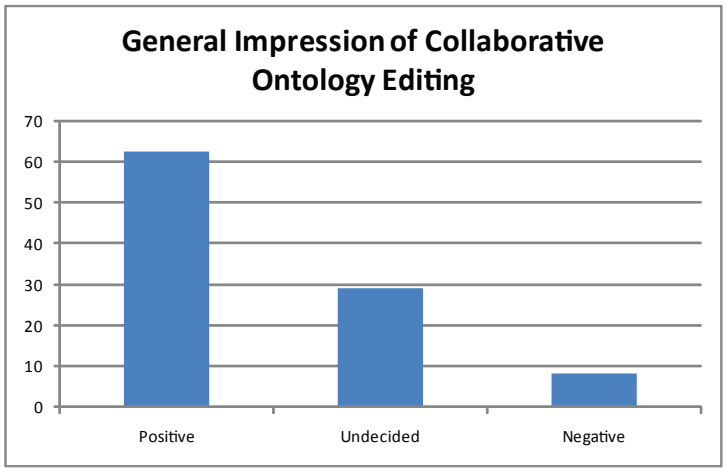


Figure 7.14: Collaborative Ontology Development Survey

As we can see from the figure, the impression of the users was around 63% positive, around 29% was undecided, and only around 8% was negative. The 29% reflects the cases when the user is undecided or when he cannot make up his mind and, as we explained above, if the user did not provide additional feedback or comments we ignored it. After analyzing the feedback received from the users on the negative answers and undecided answers when available (see the detailed analysis of each question below), we found that in general they refer only to desired improvements in the GUI to facilitate or make some tasks more intuitive. Nevertheless, feedback also showed that users were in general highly satisfied with the infrastructure and that they agreed on its usefulness and correctness. Hence, taking into account that this is the first implementation (at the best of our knowledge) of a complete infrastructure that addresses the collaborative ontology development in organizations with a well-defined edition process, **we claim that the results are highly encouraging and motivational. In particular, the results provide an indication of the real value and practical usability of the models and methods proposed in this work. Nevertheless, we need additional experiments and more users to draw full conclusions**.

In the rest of this section, we analyze the individual questions grouped by their relationship. Note that questions 2 to 5 were already analyzed in the previous sections.

Statements related to the usability and helpfulness of the infrastructure

- *The time to build an ontology collaboratively decreases with this software.* The result for this statement shows (see Figure 7.15) that everybody agreed with it. Therefore, we can affirm in a straightforward manner that **users agreed that the infrastructure supports and improves the time required to develop ontologies collaboratively**.

- *I prefer to use this software when developing an ontology collaboratively rather than the previous approach.* In this case, the result for the statement shows (see Figure 7.16) that nobody disagreed with it, one person did not know and the other two agreed. Since, there was no additional feedback from the undecided user, we can conclude that **users preferred to use the presented infrastructure to develop ontologies collaboratively rather than any previous approach**.

In FAO, the previous approach for developing ontologies collaboratively consisted in ontology editors using a local ontology engineering tool to work with an ontology and either to send the modified ontology to other editors via email, or to share it using the intranet facilities.

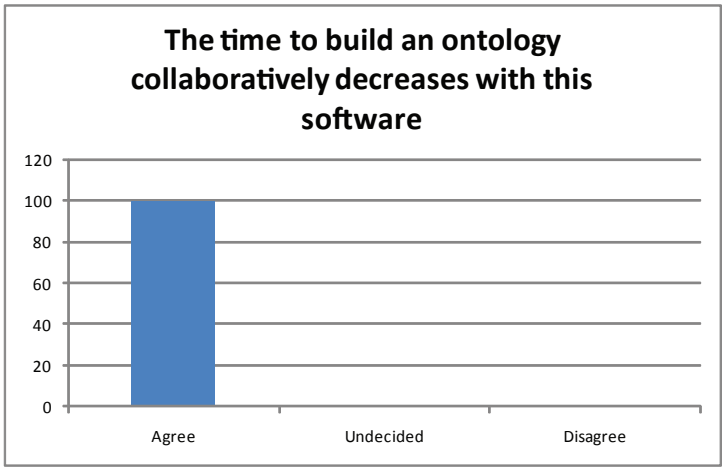


Figure 7.15: Survey Question 6-1

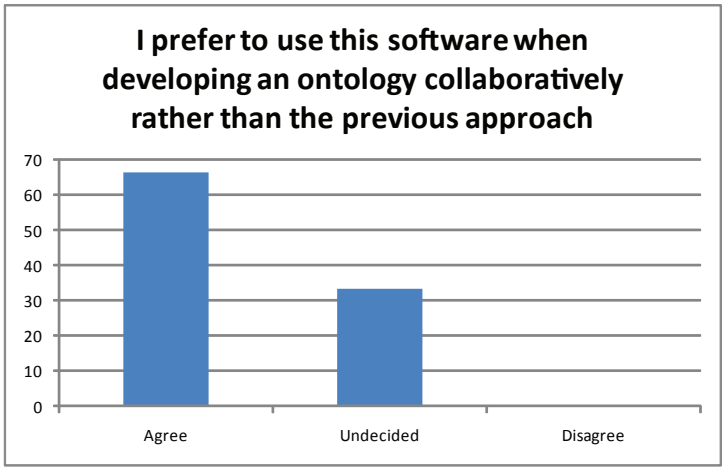


Figure 7.16: Survey Question 6-8

The previous results provide positive evidence that our proposed infrastructure facilitates the work of ontology editors when developing or maintaining an ontology collaboratively, supporting hypothesis H3 (cf. 3.4).

Statements related to the user interfaces of the infrastructure

- *The information shown in the workflow interfaces is what I expected.* The result for this statement shows (see Figure 7.17) that one person disagreed with it and the other two did not know. As we can see, this was the only case when the answer was mainly negative. However, since users were requested explicitly to provide feedback on this question, we found out that in general they did not think that something was wrong, but that they desire additional features to make the interface more intuitive. The comments from the users were the following (textually):

  – "For validators, it would be useful a more compact way to summarize changes made by subject experts"

  – "The GUI is not enough intuitive; there should be the possibility of sorting by author and by time"

  – "It would be nice if the view could be grouped around concepts to help understand the changes more easily, maybe a tree view"

  Hence, we can learn from the previous comments that the workflow interfaces can be improved still in order to satisfy users. Nevertheless, the feedback was critical as it is the exact kind of information we needed from the actual users that could not be anticipated in our first implementation. Moreover, most of these desired improvements are simple modifications easy to implement. So, we can conclude that **although the workflow interfaces can be improved they provide a good and correct starting point.**

- *The software takes the user role correctly into account when displaying information.* In this case, the result for the statement shows (see Figure 7.18) that nobody disagreed with it, one person did not know and the other two agreed. Since there was no additional feedback from the undecided user, we can conclude that **the workflow interfaces take correctly into account the user role when displaying information**.

The best and worst thing of the software

The comments from the users were the following (textually):

- Best

  – "Seeing changes of others"

  – "A unified view of everything happening in the workflow"

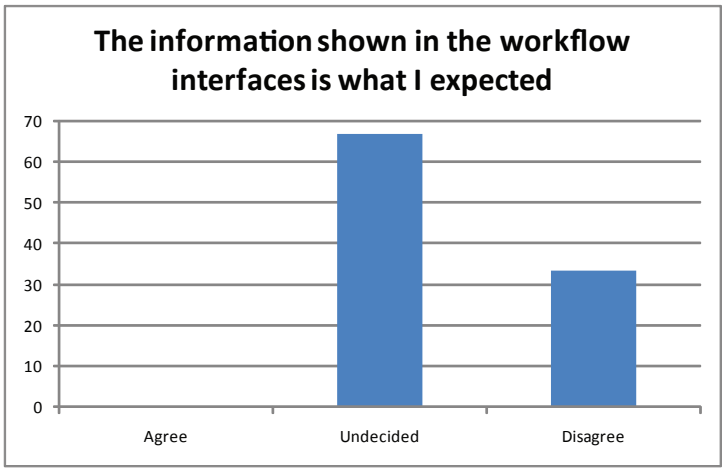


Figure 7.17: Survey Question 6-6

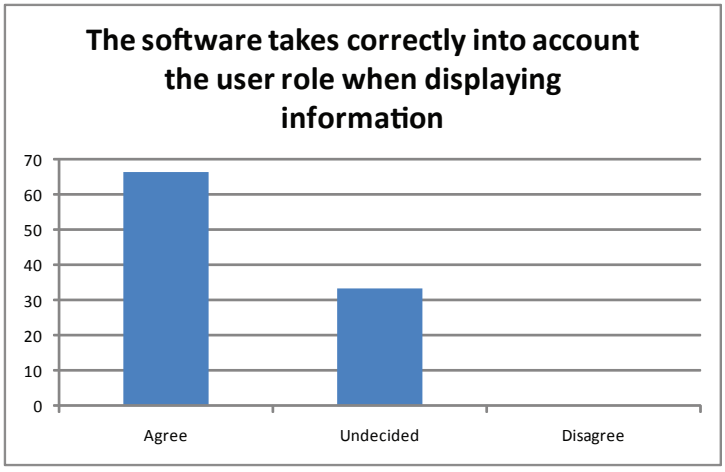


Figure 7.18: Survey Question 6-7

  - ”Great capability and real time update”

- Worst
  - ”Log messages hard to read”
  - ”Presentation of list of changes can be improved. No connection between graphical visualization of the ontology and changes made/proposed.”
  - ”Slow, difficult to add classes and instances”

Note that the comments regarding the best things refer to more fundamental matters than the worst things. Indeed, our main goal is to provide a practical and

usable infrastructure that offers a unified view of everything that is happening in the workflow, supporting users in the collaborative ontology development. This infrastructure relies on the management of changes in distributed environments that allows users to access and see changes from every editor. Furthermore, the worst things are not criticizing or in disagreement with the goal of the infrastructure. Therefore, from the previous comments, we can conclude that **although some aspects can be improved (in particular in the visualization), in general users are satisfied and find useful the whole infrastructure for the development of ontologies collaboratively in a real scenario.**

Additional feedback

Some of the final comments received from the users were (textually):

- "Grouping of changes (by change type, user, related entity etc.) would be useful".
- "An hour glass should be shown when the application is taking time to update a data submission or whatever action...".
- "It should be possible to sort (changes) by author and time, and by default they should sorted by descending time".
- "A button saying "submit all" may help".
- "Really useful would be to be able to view the ontology and have the changed areas highlighted in some way together with their current change state".
- "When first opening a view, it would be nice if it automatically refreshed itself".
- "...when going into the workflow, a wizard could lead you through the various steps that involve accessing several different functional areas of the toolkit".
- "When adding an instance, since there is a lag time as the ontology is being accessed remotely, the instance label shows a strange string until updated which is disconcerting".

Similar to the previous question, all these comments (except the last two) are desired improvements to the interfaces to make it more user friendly. The penultimate item, which suggests the creation of a wizard to guide editors when using the NeOn toolkit functionalities, would require not only support in our infrastructure but from the toolkit itself. Finally, the last item refers to a small issue with the NeOn collaboration server that we discuss in the analysis of the efficiency dimension.

Efficiency

The global efficiency was calculated as the average of the 10 SUMI questions for this dimension. According to the definition of efficiency in SUMI[vV98], users felt in 50% of the times that the software assisted them in their work (also known as transparency), in around 23% they were undecided, and in around 27% of the times they disagreed (see Figure 7.19). After analyzing each of the 10 questions for this dimension, we found out that the following two contributed in particular to the 27% of disagreement:

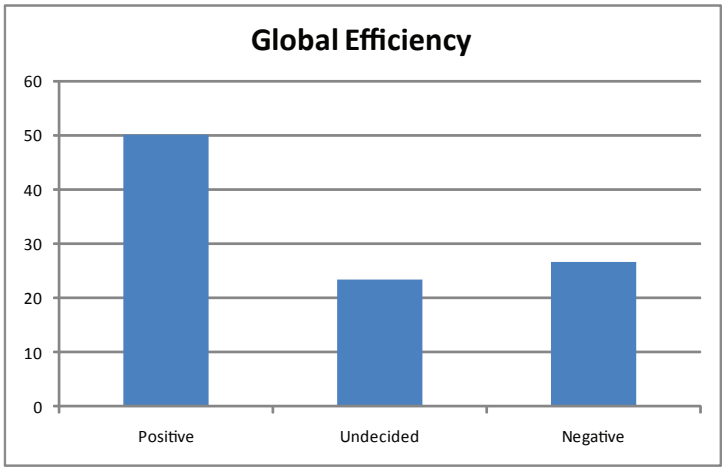


Figure 7.19: Global Efficiency Results

- *"The software has at some time stopped unexpectedly"*. As was explained above, the NeOn collaboration server crashed once during the experiment. However, after analyzing the server log, we learned that the reason was that the process ran out of memory. Additionally, although we decided to use the NeOn collaboration server for this scenario, it is not part of the implementation of our work in this thesis, and hence we consider it an external component for this evaluation.

- *"This software responds too slowly to inputs"*. In this case, it is true that sometimes the interaction with the NeOn collaboration server was slow. In fact, the last comment of the previous section refers to this issue. However, as we explained in the previous paragraph, the server was not running with the required memory (that was the reason it crashed). Consequently, it is very likely that the interaction with the server can be improved by assigning additional memory to the server process. In any case, this is still an issue related to an external component for this evaluation.

From the previous analysis, we can conclude that most of the 27% of disagreement was caused by an external component of the infrastructure. Moreover, the

result consists of 50% of positive answers and only about one fourth of the total (27%) were negative answers. Therefore, we consider the evaluation on efficiency satisfactory, as **users did not feel that the software does not help them in their work**.

Affect

The global Affect was calculated as the average of the 10 SUMI questions for this dimension. As we can see from Figure 7.20, users had a positive emotional reaction to the software (they liked it) in around 63% of the cases, they did not know in around 27% and only in 10% of the cases they had a negative reaction. After analyzing each of the 10 questions for this dimension, we found out that one of the few questions with negative answers was the following:

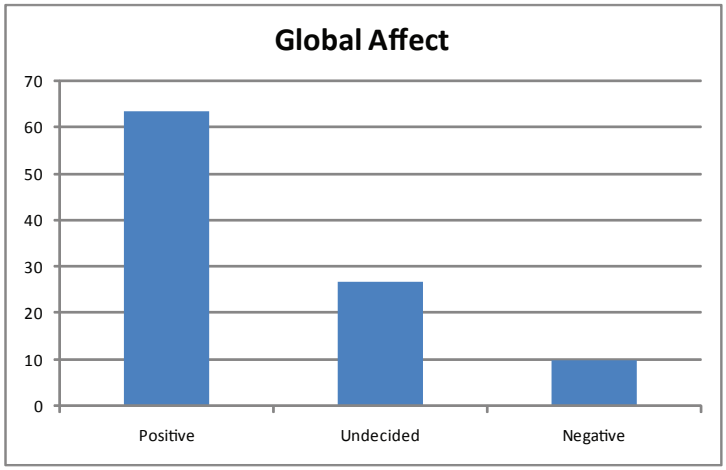


Figure 7.20: Global Affect Results

- *The way that system information is presented is clear and understandable.* As we can see this is a question related to the visualization of the information and, as we already analyzed, this was one of the aspects where users gave some comments on how to improve it.

Moreover, from the others questions for this dimension, the results show that users found the software *satisfying* and *stimulating*. Therefore, we can conclude that **in general users liked the infrastructure**.

Helpfulness

The global Helpfulness was calculated as the average of the 10 SUMI questions for this dimension. In this case, as we can see in Figure 7.21, users agreed 50% of the time that the software is self-explanatory (including things like adequate help facilities and documentation), 30% of the time they were not sure, and only 20% of

the time they disagreed. We analyzed each of the 10 questions for this dimension, and we found that the only question where most of the answers were negative (2 out of 3) was the following:

- *The speed of this software is fast enough.* In this case, as it was explained above, it is true that sometimes the interaction with the NeOn collaboration server was slow, but this is an issue external to our implementation and therefore it is not part of this evaluation.

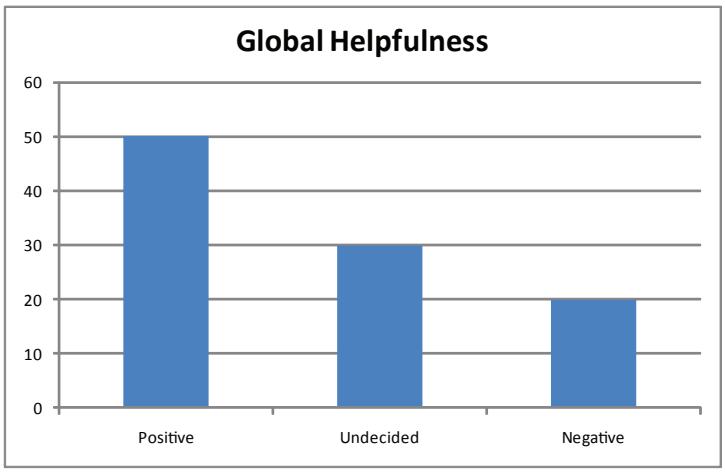


Figure 7.21: Global Helpfulness Results

Furthermore, from the other questions for this dimension, we found out that all of the users considered that the *software was consistent* and that they could *understand and act on the information provided by the software*. So, from the previous analysis and having 80% of non-negative answers, we can conclude that **in general users find the software self-explanatory**.

Control

The global Control was calculated as the average of the 10 SUMI questions for this dimension. Figure 7.22 shows that users felt in control of the software in 50% of the times, in around 27% they were not sure, and in around 23% of the times they disagreed. After analyzing each of the 10 questions for this dimension, we found out that the only questions where most of the answers were negative (2 out of 3) are the following:

- *There have been times in using this software when I have felt quite tense.* In this case, taking into account that it was the first time that the users used the infrastructure after a brief introduction of 30 minutes and that they knew they were being monitored to performed several tasks, it is understandable they felt tense.

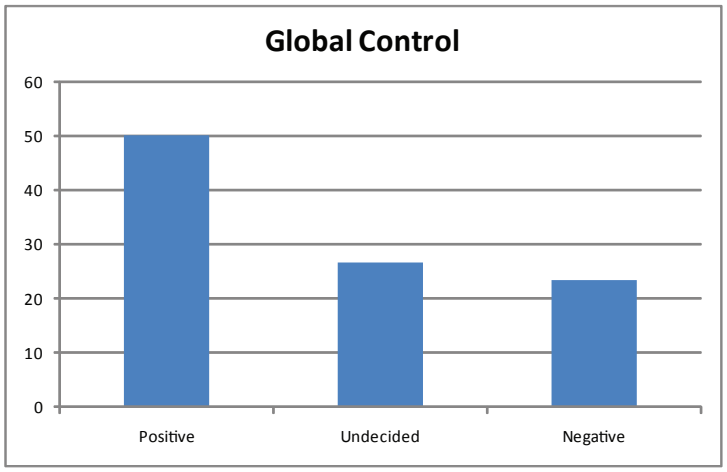


Figure 7.22: Global Control Results

- *Error prevention messages are not adequate.* This issue is something that can be improved in the interfaces. For instance, when the NeOn collaboration server crashed, users did not get any message preventing them from continuing their work because the server was down. So, similar to our previous analysis, this means that some aspects of the visualization can be improved.

Nevertheless, none of the users thought they would never *learn to use all that is offered in this software*. In fact most of them thought that *the organization of the menus or information lists seemed quite logical* and that *the software allows the user to be economic of keystrokes*. So, similar to the previous dimension, based on the previous analysis and having more than 75% of non-negative answers, we can conclude that in general **users did not feel as being controlled by the software, when carrying out the task**.

Learnability

The global Learnability was calculated as the average of the 10 SUMI questions for this dimension. As we can see from Figure 7.23, 50% of the time, the users felt positive with the speed and facility needed to master the system or to learn new features, around 27% of the time they were not sure, and around 23% of the time they disagreed. After analyzing each of the 10 questions for this dimension, we found out that none of the questions had a majority of negative answers (at least 2 out of 3). In contrast, most of the users considered that *most times when they used this software, they did not have to look for assistance*, that *this software behaved in an understandable way* and that *the software behaved as expected* (to mention a few).

Based on the previous analysis, and taking into account the more than 75% of non-negative answers, we can conclude that in general **users did not think that it is difficult to master the system or to learn new features**.

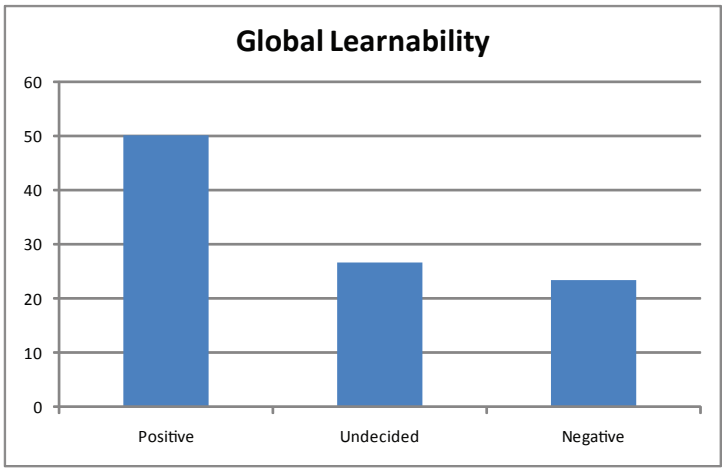


Figure 7.23: Global Learnability Results

7.5 Evaluation Summary and Recommendations

As a general conclusion we can say that the results of the evaluation are very positive. First, the results of the applicability evaluation of OMV indicate that it successfully represents the requirements of the majority of users and that it can be extended and specialized for different applications/scenarios.

Second, we have shown that our layered approach for the representation of changes can be easily instantiated for a specific ontology language (e.g., OWL and RDFS) and we have proved the completeness of the change ontology extension with respect to the OWL 2 ontology language, i.e., every change was successfully represented in our model.

Third, the analysis of the results of the experiment that we conducted at FAO shows several good points of the infrastructure as well as some issues that could be improved as part of our future work. In particular, the results showed that:

- Our models (change representation, workflow model) are adequate with respect to the ontology editors needs. That is, representative changes and workflow operations from our use case could be captured and represented correctly by our models along with their required information.

- The overall system effectiveness was positive (90% or above) which demonstrates the good capability of our infrastructure to produce the overall goal, i.e., collaborative ontology development, providing positive evidence for hypothesis H13 with respect to the effectiveness of the infrastructure (cf. 3.4).

- The efficiency of the system was in general satisfactory. A very positive point is that the time users required to complete their tasks was better than with their previous approach (see results of question 1 of the part of the survey specific to collaborative ontology development), providing positive

evidence for hypothesis H13 with respect to the efficiency of the infrastructure (cf. 3.4). Regarding the frequency of help use, it is understandable that users asked frequently for assistance, taking into account that they had only a brief introduction to the collaborative infrastructure (and the experiment) (30 min), in addition to the fact that they did not use the NeOn toolkit regularly. Finally, as most of the problems we found during the experiment were related to the NeOn Collaboration server, which is not part of this work, we feel satisfied with the results. Note that the problem related to the server crash was just a lack of memory of the process which can be easily fixed.

Finally, the results of the survey to measure the user satisfaction showed that users were in general highly satisfied with the infrastructure and that they agreed on its usefulness and correctness, providing positive evidence for hypothesis H13 with respect to the usability of the infrastructure (cf. 3.4). For instance, the questions of the survey that evaluated the editors's satisfaction regarding the collaborative ontology development, show that editors think our infrastructure is better than the previous approach, i.e., it is faster and they prefer it. Moreover, ontology editors actually liked the main features of the system (e.g., the integrated view of the workflow and the management of changes in a collaborative environment) as we can see from the feedback received in the textual answers. Nevertheless, the feedback received also shows some aspects that can be improved. In general those aspects are related to improvements in the user interfaces (which are very useful for our future work) such as being able to select multiple changes in one click, being able to sort the changes according to different criteria, or improving the readability of the change information. We also received a few comments regarding the NeOn collaboration server such as speed issues when performing some operations (e.g., adding individuals), or the problem when the server crashed. However, although we used the NeOn collaboration server in this scenario, it is not part of the work we are evaluating.

The overall results for each of the five SUMI dimensions have a similar pattern. In all cases, only around one fourth (25%) of the total answers were negative, another 25% (approximately) were undecided and at least half of the answers (50%) were always positive. So, in general we can conclude that the results were fairly satisfying, specially if we take into account that users had only a brief introduction to the system before the experiment and none of them had much experience with the NeOn toolkit. The results show that in general users liked the infrastructure and that they found it self-explanatory. Furthermore, users did not feel that the software did not help them in their work, or that they were being controlled by the software when carrying out a task, or that it was difficult to master the system (e.g., learn new features).

During the experiment and as part of the analysis of the results, we learned important lessons from which we can get some recommendations. For instance, we found out that sometimes users were interested to see only specific changes (e.g., from specific users and from a specific type), in specific order, or grouped by some

criteria, instead of having the complete history of changes in chronological order (as it is at this moment). Another interesting observation is that users wanted to have a quick view of the changes related to a specific ontology element instead of having again a complete list of changes. Also, we could observe that users got easily doubtful (e.g., they tried to repeat the action) whenever there was a small delay in the communication with the server. From these (and other) observations we got some recommendations to improve our infrastructure, specifically at the GUI level. First we should improve our views with additional features such as sorting, grouping and filtering. Second, we should also add additional user-friendly features to our interfaces, such as the ability to select several changes in one click or refreshing automatically the views when opening them. Third, we should provide a tighter link between the ontology navigator and the information displayed in our views. Fourth, in the case of the NeOn collaboration server, as it is an external component of our infrastructure, we learned that we should provide additional resources (memory) to avoid any crash and to improve the speed of the communications.

Finally, although the results of the experiment provide an indication of the real value and practical usability of the models, methods and strategies proposed in this thesis, we will need additional experiments and more users to draw full conclusions.

Chapter 8

CONCLUSIONS AND FUTURE WORK

Ontologies are increasingly being developed by reusing existing ontological and non-ontological resources, and in many cases in a collaborative and distributed manner. This change in ontological engineering practices has raised new challenges to traditional ontology engineering methodologies, methods and tools.

The need for a systematic approach to ontology development in a distributed environment has been emphasized many times in the past in the ontology engineering community. As a result, different solutions have been proposed ranging from informal or lightweight strategies (e.g., [HBS06]) to semi-formal approaches like the Gene Ontology project to formal methodologies (e.g., [Tem06]).

Moreover, as the reuse of ontologies is progressively better supported by the growing availability of ontologies for different domains, complex networks of ontologies are being created where each ontology may depend on several others and may also be related to other artifacts (e.g., individuals, mappings, applications and metadata). In this situation, the management of ontology changes becomes crucial, as ontologies are dynamic entities that change over the time. As a consequence, different approaches for the management of changes have been proposed in the past, ranging from general frameworks (e.g., [Sto04] and [Kle04]), to specific methods and models (e.g., [NCLM06], [Oli00] and [MPSd06]), that resulted in methodological and technological contributions, which are not necessarily aligned.

Furthermore, in this context, ontology metadata plays a central role, as it allows providing systematic descriptions of ontologies, helps to improve their accessibility and gives other useful ontology information to support their maintenance, such as information about changes in ontologies. Hence, ontology metadata allows determining whether an ontology has changed and it can provide a high-level overview of how an ontology has changed. For instance, additional classes have been defined or the domain has been specialized.

As we showed in the analysis of the state of the art, most of the previous aspects have been addressed to some extent in the past; however, there are still

some limitations.

The overall objective of this thesis is to support collaborative ontology development activities based on the management of changes in distributed environments. Next, we summarize for each of the main topics addressed in this thesis the open research problems, the objectives formulated, and the contributions provided. Then, we discuss the verification of our hypotheses, and finally we provide an outlook for the future work in those topics.

Ontology Metadata

Open Research Problems

Previous efforts towards the definition of an ontology metadata model resulted in general in a list of property-value pairs for describing ontologies, limiting the processing capabilities and the related relevant information that can be described, or they were implicitly defined within existing systems and repositories, which made them difficult to integrate or use. As a consequence, none of them became a standard or community-accepted ontology metadata vocabulary for documenting and annotating ontologies with metadata information, which is demonstrated by the fact that currently most ontologies exist in pure form without any additional information, such as domain of interest, authorship information and statistic information. Besides, as we discussed, general purpose standards, such as Dublin Core, are not appropriate for capturing information about ontologies due to the differences between arbitrary information sources and ontologies.

Objectives

To address the previous limitations, we formulated the following conceptual objectives:

O1. The definition of **conceptual models that provide the foundations to support the management of ontology changes in distributed environments**. This goal includes the definition of a common and community-accepted vocabulary for describing ontologies — ontology metadata model — (Objective **O1.1**).

Contributions

We have developed **OMV (Ontology Metadata Vocabulary)**, a metadata model for the description of ontologies (contribution **C1**). It was designed modularly, distinguishing between the OMV Core and various OMV Extensions, to allow reflecting the needs of the majority of ontology users, but at the same time to allow proprietary extensions and refinements in particular application scenarios. OMV

captures the key aspects of the ontology metadata information, as reported by existing case studies in ontology reuse, and it is implemented in OWL, which makes it accessible and processable by both humans and machines, and which facilitates metadata exchange among applications.

OMV provides many benefits. In the context of this thesis, it allows us to determine if an ontology has changed (the OMV description changed) and provides a high-level overview of how it has changed, but OMV does not provide more detailed information about the specific changes (specific classes or properties that have changed). However in a more general context, OMV plays an important role in the ontology reuse task by facilitating the discovery and exchange of ontologies, fostering the widespread dissemination of ontology-driven technologies and the development of fully-fledged ontology repositories on the Web.

The novelty of OMV lies mainly in the modular and open approach that facilitates the participation of different organizations and experts in the revision and refinement of the model and in extending the vocabulary for particular applications or scenarios, which have made OMV become a community-accepted vocabulary, as we have shown in the evaluation chapter.

Change Management In Distributed Environments

Open Research Problems

With respect to the management of changes, we have focused on two main activities: the representation and propagation of changes. Although we can find several approaches for the representation of changes, they are dependent on the underlying ontology model, which makes them difficult to integrate or reuse. Besides, even when they classify changes at different granularity levels, their lower level considers changes at the entity level (e.g., concepts, properties and individuals), instead of considering the real low level operations that can be performed in an ontology, which makes them less flexible, less detailed and more difficult to process. On the other hand, the propagation of changes has only been considered in the past to related ontologies, individuals, and to some extent to related applications. Furthermore, for the propagation of changes to related ontologies, existing approaches consider only a central (main) copy of the ontology that is either replicated or divided into several component ontologies and where in general, changes are propagated only in one direction: from the main copy to its replicas. The only exception is when the ontology is divided into several component ontologies, but in this case the management of changes is centralized. As a consequence, the propagation of changes to distributed (editable) copies of the same ontology with a distributed control, are not support yet. Finally, the evolution of the ontology related metadata as consequences of changes in the ontology has not even been considered.

Objectives

To address the previous limitations, we formulated the following conceptual objectives:

O1. The definition of **conceptual models that provide the foundations to support the management of ontology changes in distributed environments**. This goal includes the definition of a language-independent model for the representation of ontology changes (Objective **O1.2**).

O2. The development of **models, methods and strategies for the management of ontology changes in distributed environments to support collaborative ontology development**. This objective includes the definition of strategies for the propagation of ontology changes in distributed environment to distributed copies of the same ontology and to ontology related metadata (Objective **O2.1**); and methods and strategies for the manipulation of ontology changes and the identification of ontology versions (Objective **O2.2**).

Additionally, we formulated the following technological objectives:

O3. The development of **an infrastructure that implements the models, methods and strategies proposed for the management of changes in order to support collaborative ontology development in distributed environments**. This infrastructure should include the implementation of a distributed ontology registry (Objective **O3.1**).

Contributions

With respect to the management of ontology changes in distributed environments we address several issues.

First, for the representation of ontology changes, we provide a layered model that consists of a **generic change ontology** independent of the underlying ontology language that can be reused and specialized for different ontology languages (contribution **C2**). Our model considers ontology changes at the lowest level of granularity (the real atomic operations) instead of at the entity level like in existing approaches. Since, information about changes in ontologies is a type of ontology metadata, we implemented our change ontology as one extension of OMV. Additionally, our model integrates many of the relevant features of existing approaches for describing ontology changes, but its novelty lies in its language-independent approach and finer granularity level of changes. Our contribution also included the development of two extensions for two different ontology languages (OWL 2 and RDFS).

A language-independent model that can be easily extended fosters its reusability and interoperability between different applications and scenarios, and a finer granularity level for the classification of ontology changes supports a more efficient processing of changes (e.g., to implement redo/undo operations and comparison between ontology versions) and provides a more detailed information of how

an ontology changed as well as the specific consequences of operations at a higher level.

Second, we address two open research problems in the literature regarding the propagation of changes, making them novel contributions: On the one hand, **the propagation of changes to distributed copies of the same ontology** (contribution **C3**), and on the other hand, **the propagation of changes to ontology related metadata** (contribution **C4**).

The benefits of the former is that it addresses current unsupported scenarios for collaborative ontology development where distributed editors can work with a local copy of the same ontology version, having a distributed control of the ontology and its related changes. While the benefit of having updated ontology metadata is that it supports an accurate and efficient identification of changed ontologies as well as the more general task of ontology reuse.

Our solution also addresses complementary activities required for the management of ontology changes. We propose **methods and strategies for the identification of ontology versions and the manipulation of changes**, including their capturing (e.g., monitoring, processing and logging), storage and maintenance in a distributed ontology registry. This work constitutes part of contribution **C5**.

Our contributions also include the implementation of the distributed ontology registry Oyster (contribution **C5**) for the management of ontology changes in distributed environments.

This implementation shows the practicability of our contributions.

Collaborative Ontology Development

Open Research Problems

Existing solutions for collaborative ontology development support only a centralized management of the ontology and its related changes. In some cases, a shared/main copy of the ontology can be adapted/specialized by distributed users; in other cases, a main copy of the ontology is divided in sub-ontologies each of them modified by distributed users; and in other cases there is only a central copy of the ontology that all distributed users can modify. In every case, changes are applied and managed in the central copy of the ontology. Moreover, they do not support the processes typically followed by organizations for the coordination of the change proposals, which specifies the type of actions/operations ontology editors can perform depending on their role and the state of the ontology. Even more, in many of them there is not even a formal management of ontology changes (e.g., explicit representation and propagation). Finally, there is no integrated approach (and technological support) that addresses all the previous issues.

Objectives

To address the previous limitations, we formulated the following conceptual objectives:

O2. The development of **models, methods and strategies for the management of ontology changes in distributed environments to support collaborative ontology development**. This objective includes the formalization of the process of collaborative ontology development usually followed by organizations to coordinate, on the one hand, who can change the ontology, and on the other hand, when and how an ontology can be changed (Objective **O2.3**); and strategies for the management of the collaborative process (Objective **O2.4**).

Additionally, we formulated the following technological objectives:

O3. The development of **an infrastructure that implements the models, methods and strategies proposed for the management of changes in order to support collaborative ontology development in distributed environments**. This infrastructure should include the implementation of a changes manipulation component (Objective **O3.2**), a collaborative workflow management component (Objective **O3.3**), and user related components for editing and visualizing ontologies (and related information) during collaborative ontology development (Objective **O3.4**).

Contributions

For our contribution to the collaborative ontology development, first, we address different collaborative scenarios, typical in an organizational setting, where the management of the ontology and its related changes can be centralized, distributed or a hybrid between both of them. So, unlike existing approaches that support only a centralized management, we provide novel strategies to address scenarios in which distributed ontology editors can work with a local copy of the ontology, while our solution takes care of synchronizing the different copies by means of the change propagation strategies we developed.

The benefit is that we provide a more flexible solution, compared to existing approaches, that can address different organizational needs and limitations. For example, in many cases, ontology editors may not be able to have a permanent or reliable connection to a central server to work with an ontology. Similarly, other well known problems of centralized approaches (e.g., performance, maintenance and resources) can be avoided by using a distributed control.

Second, unlike most existing approaches, we focus on the process followed by organizations for the coordination of the change proposals. We propose to **formalize this process by means of a collaborative editorial workflow model**. Further, this model can then be implemented as an ontology itself. So, we also provide a **workflow ontology** (as we showed with a running example) that was implemented by reusing knowledge modeled by OMV and our change ontology. This work is part of contribution **C6**.

The benefits of having a workflow ontology is that it allows the formal and explicit representation of the workflow knowledge in a machine-understandable format, that can be easily integrated with other models (e.g., our change representation model). In particular, it allows to represent the tight relationship that exists between the workflow elements and the ontology changes. Besides, having the history of the workflow as individuals of an ontology allows reusing existing ontology-driven technologies for the processing of the workflow information (e.g., propagation mechanisms and reasoning).

Additionally, we propose **strategies for the management of the collaborative process** during the ontology development. We identify the set of tasks that have to be carried out, including those to enforce the constraints specified by the collaborative process, and propose strategies to deal with them (contribution **C6**).

Our contributions also include an integrated infrastructure implemented within the NeOn Toolkit by means of a set of plugins and extensions that support the collaborative ontology development (contribution **C7**). It relies on the distributed ontology registry Oyster for the management of ontology changes in distributed environments. In contrast to existing solutions, it supports a formal collaborative process that coordinates the proposal of ontology changes and a completely distributed control for the management of changes.

This implementation shows the practicability of our contributions.

Hypotheses verification

We have verified the hypotheses of this thesis by different means:

First, we have shown the applicability of OMV, by demonstrating that it has become a community-accepted vocabulary (hypothesis **H4**), and by showing how OMV has been extended for different applications/scenarios (hypothesis **H5**). Then, we have shown the completeness of the change representation model with respect to the OWL 2 ontology language by conducting a simulation of all possible changes in OWL 2 (hypothesis **H8**).

We have also shown how the layered approach for the representation of changes can be reused and specialized for OWL 2 and RDFS (to verify hypothesis **H6**). Similarly, we have discussed how the change representation model unifies many of the features of different existing approaches and extends them by considering the atomic change operations that can be performed in an ontology language (hypothesis **H7**).

Next, the results of the experiment we conducted at FAO provided positive evidence supporting the adequacy of the change representation model and the workflow model (hypothesis **H10**), and the usability and performance of the infrastructure (hypothesis **H13**). In this experiment a team of ontology editors were collaboratively maintaining one of the ontologies of the organization. They were requested to perform a set of representative changes and activities, and at the end the users fulfilled a survey to evaluate the overall user satisfaction. This survey pro-

vided, among others, positive evidence that the infrastructure facilitates the work of ontology editors when developing or maintaining an ontology collaboratively (hypothesis **H3**). Additionally, this experiment provided positive evidence that the collaborative process, usually followed by organizations for the management of ontology change proposals, can be successfully modeled by means of a collaborative editorial workflow (hypothesis **H9**).

A complete and detailed description of the definition and analysis of the experiment was presented in the evaluation chapter, but as a general conclusion, we can say that the results of the evaluation are positive and motivational. We have also learned important lessons on how to improve our infrastructure, but in general at the GUI level. Furthermore, we have illustrated with different scenarios how the strategies for the propagation of changes to distributed copies of an ontology support a distributed control during the collaborative ontology development and maintenance (hypothesis **H11**). Additionally, we have discussed how the formal representation of the lifecycle of ontology metadata annotations supports the propagation of ontology changes to their related metadata (hypothesis **H12**).

Finally, the implementation of our infrastructure shows that the management of ontology changes in distributed environments supports the collaborative ontology development and maintenance (hypothesis **H1**), and that the management of ontology changes can be ontology-driven (hypothesis **H2**).

Future Work

Although we address in this thesis many open research problems in the context of change management in collaborative and distributed environments, there are still open issues that can be resolved or extensions that can be implemented in the future. We would like to mention the most important topics from our perspective:

- *Configurable workflows*: An important feature to elaborate in the future is to extend our approach for the formalization of the collaborative process to support any kind of collaborative editorial workflow. This would require the definition of a more flexible workflow ontology and the implementation of a customizable workflow management component that could be reconfigured in run-time, making sure at every moment that the workflow constraints are not violated.

- *Conflict Resolution*: In our current work, the propagation of changes to distributed ontology copies can only minimize conflicts by running periodically a synchronization process in an automatic manner. However, we do not provide explicit mechanisms for the identification and resolution of conflicts. This is a very interesting issue since the conflicts should be detected at the semantic level, and in some cases it could be possible to resolve it automatically or at least to provide some suggestions to the users.

- *Argumentation support*: Another interesting aspect that can complement our current work is to support argumentation lines during the collaborative ontology development. In particular, this would be useful for the proposal of changes. However, instead of being just comments annotations to the proposed changes, it would be more interesting to support discussions that can be represented in a machine-understandable format. This would support a better processing of the information, keeping the track of the users reasoning, and even use some knowledge-elicitation techniques.

- *Algebra of changes*: Another interesting future work is to define an algebra of changes that would be useful to represent the semantics of changes on ontologies expressed in different ontology languages. For this task, it will be necessary to identify and formalize the different operations that can be performed over changes. Having such an algebra will support, for instance, semantic comparisons between changes in ontologies represented in different ontology languages.

The general goal of this thesis, the management of ontology changes in distributed environments to support collaborative ontology development, is a core requirement to support the new generation of ontology development activities and scenarios. This thesis provides a step forward to achieve this goal.

Appendix A

EXPERIMENT GUIDES

A.1 Subject Expert 1

T1. Setting-up the environment.

1. Start the NTK.
2. Configure the Registry Properties: Open Preferences → Oyster Storage Preference and configure it with the following properties:
   - Super Node IP: ServerIP (provided at the experiment)
   - Push Node IP: blank (default)
   - Read Ontologies Locally: checked
3. Start the Registry: Either from the Registry menu or by clicking the Registry icon on the toolbar. Note that the icon shape and the message on the Registry menu changes when the registry is running
4. Identify to the system: Open Preferences → Collaborative Development Preference, go to the Register section and provide:
   - Your first name
   - Your last name
   - Choose your role
   - Click Register
5. Create a new Ontology Development Project with the following properties:
   - Ontology Language: OWL
   - Datamodel Type: CollaborationServer
     - Host: ServerIP (provided at the experiment)
     - Port: 8267 (default)
6. Open existing ontology species_v1.0_model.owl into the project by selecting "Add Ontology To Project" in the project context menu.

T2. Change Proposals.

1. Start logging the ontology: Right-click the ontology and select ”Log Changes”
2. Add Individual 31005_10000 (Species)
3. Add Individual 31005_10001 (Species)
4. Add Individual 31005_10000 DataProperty hasCodeAlpha3 value: DCR. Type: string
5. Add Individual 31005_10000 DataProperty hasID value: 10000. Type: string
6. Add Individual 31005_10000 DataProperty hasMeta value: 31005. Type: string
7. Add Individual 31005_10000 DataProperty hasNameEN value: Yellow-nosed albat. Type: string
8. Add Individual 31005_10000 DataProperty hasNameScientific value: Diomedea chlororhynchos. Type: string
9. Add Individual 31005_10001 DataProperty hasCodeAlpha3 value: PDM. Type: string
10. Add Individual 31005_10001 DataProperty hasID value: 10001. Type: string
11. Add Individual 31005_10000 DataProperty hasMeta value: 31005. Type: string
12. Add Individual 31005_10001 DataProperty hasNameEN value: Great-winged petre. Type: string
13. Add Individual 31005_10001 DataProperty hasNameScientific value: Pterodroma wrong macroptera. Type: string
14. Add Root Class Speciation
15. Add Individual Allopatric (Speciation)
16. Add Individual Peripatric (Speciation)
17. Add Individual ParapatricWrong (Speciation)
18. Add DataProperty hasSpeciation (Species)

T3. Analysis of the changes/actions. To see the changes information, open the view ”Change Log View”. Read and comment. To see the workflow actions, open the view ”Draft View”. Read and comment.

T4. Submit your changes to be approved. From the ”Draft View”, select them and click submit to be approved.

Wait for Validator to finish task T2.

T5. Delete the rejected changes. From the "Draft View", select them and click delete button.

T6. Submit change to be deleted. Open "Approved View" and submit the following change to be deleted:

- DataProperty hasSpeciation (Species)
  - Change Type: AddDataProperty
  - Related Entity: hasSpeciation
  - Author: SE1

A.2 Subject Expert 2

T1. Setting-up the environment.

1. Start the NTK.
2. Configure the Registry Properties: Open Preferences → Oyster Storage Preference and configure it with the following properties:
   - Super Node IP: ServerIP (provided at the experiment)
   - Push Node IP: blank (default)
   - Read Ontologies Locally: checked
3. Start the Registry: Either from the Registry menu or by clicking the Registry icon on the toolbar. Note that the icon shape and the message on the Registry menu changes when the registry is running
4. Identify to the system: Open Preferences → Collaborative Development Preference, go to the Register section and provide:
   - Your first name
   - Your last name
   - Choose your role
   - Click Register
5. Create a new Ontology Development Project with the following properties:
   - Ontology Language: OWL
   - Datamodel Type: CollaborationServer
     - Host: ServerIP (provided at the experiment)
     - Port: 8267 (default)
6. Open existing ontology species_v1.0_model.owl into the project by selecting "Add Ontology To Project" in the project context menu.

T2. Change Proposals.

1. Start logging the ontology: Right-click the ontology and select ”Log Changes”
2. Add SubClass genus of biological_entity. Right Click biological_entity and add Class genus.
3. Add Root Class Person
4. Add DataProperty name (Person)
5. Add ObjectProperty hasScientificNameAuthor
6. Add ObjectProperty domain (hasScientificNameAuthor,Species)
7. Add ObjectProperty range (hasScientificNameAuthor,Person)
8. Add Root Class Category
9. Add DataProperty description (Category)
10. Add Individual Extinct (Category)
11. Add Individual Endangered (Category)
12. Add Individual Extint DataPropertyValue (description, the last remaining member of the species has died, string,-)
13. Add ObjectProperty hasCategory.
14. Add ObjectPropertyRange (hasCategory,Category)
15. Add ObjectPropertyDomain (hasCategory,Species)

    Wait for SE1 to finish task T2.

16. Add Species Class Super Restriction (AT_LEAST/MIN, 1, hasScientificNameAuthor, -)
17. Add Species Class Super Restriction (EXACTLY/CARD, 1, hasCategory, Thing)
18. Add Species Class Super Restriction (HAS_VALUE, hasMeta, ”31005”)

T3. Analysis of the changes/actions. To see the changes information, open the view ”Change Log View”. Read and comment. To see the workflow actions, open the view ”Draft View”. Read and comment.

T4. Submit your changes to be approved. From the ”Draft View”, select them and click submit to be approved.

Wait for Validator to finish task T2.

T5. Delete the rejected changes. From the ”Draft View”, select them and click delete button.

T6. Submit change to be deleted. Open "Approved View" and submit the following change to be deleted:

- Individual Extint DataPropertyValue (description, the last remaining member of the species has died, string, -)
  - Change Type: AddIndividualDataProperty
  - Related Entity: Extint
  - Author: SE2

A.3 Validator 1

T1. Setting-up the environment.

1. Start the NTK.
2. Configure the Registry Properties: Open Preferences → Oyster Storage Preference and configure it with the following properties:
   - Super Node IP: ServerIP (provided at the experiment)
   - Push Node IP: blank (default)
   - Read Ontologies Locally: checked
3. Start the Registry: Either from the Registry menu or by clicking the Registry icon on the toolbar. Note that the icon shape and the message on the Registry menu changes when the registry is running
4. Identify to the system: Open Preferences → Collaborative Development Preference, go to the Register section and provide:
   - Your first name
   - Your last name
   - Choose your role
   - Click Register
5. Create a new Ontology Development Project with the following properties:
   - Ontology Language: OWL
   - Datamodel Type: CollaborationServer
     - Host: ServerIP (provided at the experiment)
     - Port: 8267 (default)
6. Open existing ontology species_v1.0_model.owl into the project by selecting "Add Ontology To Project" in the project context menu.

T2. Approve/Reject Changes

1. Start logging the ontology: Right-click the ontology and select "Log Changes"
   Wait for SEs finish task T4.

2. Open "To Be Approved view" and analyze changes from SubjectExperts. Approve all except the following (reject them):
   - Individual 31005_10001 DataProperty hasNameScientific value: Pterodroma wrong macroptera.
     - Change Type: AddIndividualDataProperty.
     - Related Entity: 31005_10001
     - Author: SE1
   - Individual ParapatricWrong
     - Change Type: AddIndividual
     - Related Entity: Speciation
     - Author: SE1
   - Species Class Super Restriction (EXACTLY/CARD, 1, hasCategory, -)
     - Change Type: AddSubClassOf
     - Related Entity: Species
     - Author: SE2
   - SubClass genus of biological_entity.
     - Change Type: AddSubClassOf
     - Related Entity: genus
     - Author: SE2

   Wait for SEs finish task T6.

T3. Other Activities of the Collaborative Editorial Workflow

1. Open "To Be Deleted View" and reject back to Approved the change:
   - DataProperty hasSpeciation (Species):
     - Change Type: AddDataProperty
     - Related Entity: hasSpeciation
     - Author: SE1
2. From "To Be Deleted View", delete permanently the change:
   - Individual Extint DataPropertyValue (description, the last remaining member of the species has died, string,-)
     - Change Type: AddIndividualDataProperty
     - Related Entity: Extint
     - Author: SE2

Imagine there is another validator, then do:

3. Open ”Approved View” and (i.e. Validator1) and reject back to be approved change:
   - Species Class Super Restriction (HAS_VALUE, hasMeta, ”31005”).
     - Change Type: AddSubClassOf
     - Related Entity: Species.
     - Author: SE2
4. Open ”ToBeApproved View” and (i.e. Validator2) approve the pending change.

Appendix B

COLLABORATIVE ONTOLOGY DEVELOPMENT FRAMEWORK EVALUATION SURVEY

B.1 Collaborative Ontology Development

1. The time to build an ontology collaboratively decreases with this software.
    - Agree
    - Undecided
    - Disagree
2. The software allows to perform all the required actions of our workflow.
    - Agree
    - Undecided
    - Disagree
3. Users are able to perform all the actions they could perform according to their role and only those.
    - Agree
    - Undecided
    - Disagree
4. The information captured for the ontology changes is enough.
    - Agree

- Undecided
- Disagree

If something is missing please specify:

5. Some changes were not captured correctly.

   - Agree
   - Undecided
   - Disagree

6. The information shown in the workflow interfaces is what I expected.

   - Agree
   - Undecided
   - Disagree

   If something is missing please specify:

7. The software takes correctly into account the user role when displaying information.

   - Agree
   - Undecided
   - Disagree

8. I prefer to use this software when developing an ontology collaboratively rather than the previous approach.

   - Agree
   - Undecided
   - Disagree

9. The best and worst things of the software are:
   Best:
   Worst:

10. Please provide any additional feedback/comments:

B.2 Efficiency

1. This software responds too slowly to inputs.
    - Agree
    - Undecided
    - Disagree
2. I would recommend this software to my colleagues.
    - Agree
    - Undecided
    - Disagree
3. The instructions and prompts are helpful.
    - Agree
    - Undecided
    - Disagree
4. The software has at some time stopped unexpectedly.
    - Agree
    - Undecided
    - Disagree
5. Learning to operate this software initially is full of problems.
    - Agree
    - Undecided
    - Disagree
6. I sometimes don't know what to do next with this software.
    - Agree
    - Undecided
    - Disagree
7. I enjoy my sessions with this software.
    - Agree
    - Undecided
    - Disagree
8. I find that the help information given by this software is not very useful.

- Agree
- Undecided
- Disagree

9. If this software stops, it is not easy to restart it.
   - Agree
   - Undecided
   - Disagree
10. It takes too long to learn the software commands.
    - Agree
    - Undecided
    - Disagree

B.3 Affect

1. I sometimes wonder if I'm using the right command.
   - Agree
   - Undecided
   - Disagree
2. Working with this software is satisfying.
   - Agree
   - Undecided
   - Disagree
3. The way that system information is presented is clear and understandable.
   - Agree
   - Undecided
   - Disagree
4. I feel safer if I use only a few familiar commands or operations.
   - Agree
   - Undecided
   - Disagree
5. The software documentation is very informative.

- Agree
- Undecided
- Disagree

6. This software seems to disrupt the way I normally like to arrange my work.
   - Agree
   - Undecided
   - Disagree
7. Working with this software is mentally stimulating.
   - Agree
   - Undecided
   - Disagree
8. There is never enough information on the screen when its needed.
   - Agree
   - Undecided
   - Disagree
9. I feel in command of this software when I am using it.
   - Agree
   - Undecided
   - Disagree
10. I prefer to stick to the facilities that I know best.
    - Agree
    - Undecided
    - Disagree

B.4 Helpfulness

1. I think this software is inconsistent.
   - Agree
   - Undecided
   - Disagree
2. I would not like to use this software every day.

- Agree
- Undecided
- Disagree

3. I can understand and act on the information provided by this software.
   - Agree
   - Undecided
   - Disagree
4. This software is awkward when I want to do something which is not standard.
   - Agree
   - Undecided
   - Disagree
5. There is too much to read before you can use the software.
   - Agree
   - Undecided
   - Disagree
6. Tasks can be performed in a straightforward manner using this software.
   - Agree
   - Undecided
   - Disagree
7. Using this software is frustrating.
   - Agree
   - Undecided
   - Disagree
8. The software has helped me overcome any problems I have had in using it.
   - Agree
   - Undecided
   - Disagree
9. The speed of this software is fast enough.
   - Agree
   - Undecided

- Disagree

10. I keep having to go back to look at the guides.
    - Agree
    - Undecided
    - Disagree

B.5 Control

1. It is obvious that user needs have been fully taken into consideration.
    - Agree
    - Undecided
    - Disagree
2. There have been times in using this software when I have felt quite tense.
    - Agree
    - Undecided
    - Disagree
3. The organization of the menus or information lists seems quite logical.
    - Agree
    - Undecided
    - Disagree
4. The software allows the user to be economic of keystrokes.
    - Agree
    - Undecided
    - Disagree
5. Learning how to use new functions is difficult.
    - Agree
    - Undecided
    - Disagree
6. There are too many steps required to get something to work.
    - Agree
    - Undecided

- Disagree

7. I think this software has made me have a headache on occasion.
   - Agree
   - Undecided
   - Disagree
8. Error prevention messages are not adequate.
   - Agree
   - Undecided
   - Disagree
9. It is easy to make the software do exactly what you want.
   - Agree
   - Undecided
   - Disagree
10. I will never learn to use all that is offered in this software.
    - Agree
    - Undecided
    - Disagree

B.6 Learnability

1. The software hasnt always done what I was expecting.
   - Agree
   - Undecided
   - Disagree
2. The software has a very attractive presentation.
   - Agree
   - Undecided
   - Disagree
3. Either the amount or quality of the help information varies across the system.
   - Agree
   - Undecided

- Disagree

4. It is relatively easy to move from one part of a task to another.
   - Agree
   - Undecided
   - Disagree
5. It is easy to forget how to do things with this software.
   - Agree
   - Undecided
   - Disagree
6. This software occasionally behaves in a way which cant be understood.
   - Agree
   - Undecided
   - Disagree
7. This software is really very awkward.
   - Agree
   - Undecided
   - Disagree
8. It is easy to see at a glance what the options are at each stage.
   - Agree
   - Undecided
   - Disagree
9. Getting data files in and out of the system is not easy.
   - Agree
   - Undecided
   - Disagree
10. I have to look for assistance most times when I use this software.
    - Agree
    - Undecided
    - Disagree

Bibliography

[ACFLGP01] J. C. Arpírez, O. Corcho, M. Fernández-López, and A. Gómez-Pérez. WebODE: a scalable workbench for ontological engineering. In *K-CAP '01: Proceedings of the 1st international conference on Knowledge capture*, pages 6–13, New York, NY, USA, 2001. ACM.

[ADR06] S. Auer, S. Dietzold, and T. Riechert. Ontowiki - a tool for social, semantic collaboration. In *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006*, pages 736–749. Springer, 2006.

[AGPLTP00] J. Arpírez, A. Gómez-Pérez, A. Lozano-Tello, and H. Pinto. Reference Ontology and (ONTO)2 Agent: The Ontology Yellow Pages. *Knowledge and Information Systems*, 2:387–412, 2000.

[Arm01] W. Y. Arms. *Digital Libraries*. The MIT Press, 2001.

[Ber90] B. Berliner. Cvs II: Parallelizing software development. In *Proc. of the USENIX Winter 1990 Technical Conf. (Berkeley, CA)*, pages 341–352. USENIX Association, 1990.

[BFLGPGP98] M. Blázquez, M. Fernández-López, J. M. García-Pinar, and A. Gómez-Pérez. Building ontologies at the knowledge level using the ontology design environment. In *Proceedings of the 11th International Workshop on Knowledge Acquisition, Modeling and Management (KAW'98)*, 1998.

[BG04] D. Brickley and R. V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Rec. 10 February 2004, 2004. available at `http://www.w3.org/TR/rdf-schema/`.

[BH04] J. Bao and V. Honavar. Collaborative ontology building with wiki@nt - a multi-agent based ontology building environment. In *Proc. of 3rd International Workshop on Evaluation of Ontology-based Tools, located at the 3rd International Semantic Web Conference ISWC 2004*, Hiroshima, Japan, November 2004.

[BKKK87] J. Banerjee, W. Kim, H.-J. Kim, and H. F. Korth. Semantics and implementation of schema evolution in object-oriented databases. *SIGMOD Rec.*, 16(3):311–322, 1987.

[BKvH02] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF Schema. In *Proceedings of the first Int'l Semantic Web Conference (ISWC 2002)*, pages 54+, Sardinia, Italy, 2002. Springer Verlag.

[BLFM05] T. Berners-Lee, R. Fielding, and L. Masinter. RFC 3986, Uniform Resource Identifier (URI): Generic syntax, 2005. Available at `http://tools.ietf.org/html/rfc3986`.

[BR00] K. H. Bennett and V. Rajlich. Software maintenance and evolution: a roadmap. In *ICSE - Future of SE Track*, pages 73–87, 2000.

[BSH86] V. R. Basili, R. W. Selby, and D. H. Hutchens. Experimentation in software engineering. *IEEE Trans. Software Eng.*, 12(7):733–743, 1986.

[CAK+06] O. Corcho, P. Alper, I. Kotsiopoulos, P. Missier, S. Bechhofer, and C. Goble. An overview of S-OGSA: A reference semantic grid architecture. *Web Semantics: Science, Services and Agents on the World Wide Web*, 4(2):102–115, June 2006.

[CAM+07] O. Corcho, P. Alper, P. Missier, S. Bechhofer, C. A. Goble, and W. Xing. Metadata management in S-OGSA. In Y. Shi, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, editors, *International Conference on Computational Science (2)*, volume 4488 of *Lecture Notes in Computer Science*, pages 712–719. Springer, 2007.

[CAW98] S. S. Chawathe, S. Abiteboul, and J. Widom. Representing and querying changes in semistructured data. In *In Proceedings of the International Conference on Data Engineering*, pages 4–13, 1998.

[CFF+98] V. Chaudhri, A. Farquhar, R. Fikes, P. Karp, and J. Rice. Open knowledge base connectivity 2.0.3. www.ai.sri.com/ okbc/spec/okbc2/okbc2.html, 1998.

[CFLGP03] O. Corcho, M. Fernández-López, and A. Gómez-Pérez. Methodologies, tools and languages for building ontologies: where is their meeting point? *Data Knowl. Eng.*, 46(1):41–64, 2003.

[CFLGPV02] O. Corcho, M. Fernández-López, A. Gómez-Pérez, and O. Vicente. WebODE: An integrated workbench for ontology representation, reasoning, and exchange. In *Proceedings of EKAW 2002. LNCS 2473*, pages 138–153, 2002.

[D+04] L. Ding et al. Swoogle: A search and metadata engine for the semantic web. In *In Proceedings of the Thirteenth ACM Conference on Information and Knowledge Management*, pages 58–61, November 2004.

[dBG+07] M. dÁquin, C. Baldassarre, L. Gridinoc, S. Angeletou, M. Sabou, and E. Motta. Watson: A gateway for the semantic web. In *Poster session of the European Semantic Web Conference, ESWC'07*, 2007.

[DF01] Y. Ding and D. Fensel. Ontology library systems: The key to successful ontology reuse. In *Proceedings of SWWS'01, The first Semantic Web Working Symposium*, pages 93–112, Stanford University, California, USA, August 2001.

[dHR+08] M. d'Aquin, P. Haase, S. Rudolph, J. Euzenat, A. Zimmermann, M. Dzbor, M. Iglesias, Y. Jacques, C. Caracciolo, C. Buil-Aranda, and J. Gómez-Pérez. Neon formalisms for modularization: Syntax, semantics, algebra. Technical Report D1.1.3, Open University, February 2008.

[DKP+01] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. J. Shenoy. Adaptive push-pull: disseminating dynamic web data. In *World Wide Web*, pages 265–274, 2001.

[DMC99] J. Domingue, E. Motta, and O. Corcho. Knowledge modelling in WebOnto and OCML: A user guide, 1999. Available at `http://kmi.open.ac.uk/projects/ocml/`.

[dMLM06] A. de Moor, P. D. Leenheer, and R. Meersman. DOGMA-MESS: A meaning evolution support system for interorganizational ontology engineering. In *Proc. of the International Conference on Conceptual Structures, (ICCS 2006), Aalborg, Denmark*. Springer, 2006.

[Dom98] J. Domingue. Tadzebao and webonto: Discussing, browsing, and editing ontologies on the web. In *In Proceedings of the 11th Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1998.

[DS05] M. Duerst and M. Suignard. Internationalized Resource Identifiers (IRIs). Technical report, Network Working Group, Jan-

uary 2005. Available at `http://www.rfc-archive.org/getrfc.php?rfc=3987`.

[EH03] J. Euzenat and S. H. The 'family of languages' approach to semantic interoperability. *In: Omelayenko B, Klein M (eds) Knowledge Knowledge transformation for the semantic web, IOS press*, pages 49–63, 2003.

[EHS+03] M. Ehrig, P. Haase, R. Siebes, S. Staab, R. Studer, and C. Tempich. The swap data and metadata model for semantics-based peer-to-peer systems. In *In: Proceedings of MATES-2003. First German Conference on Multiagent Technologies. LNAI*, pages 22–25. Springer, 2003.

[Euz95] J. Euzenat. Building consensual knowledge bases: Context and architecture. In *Towards Very Large Knowledge Bases. Proceedings of the KB&KS '95 Conference*, 1995.

[FFR96] A. Farquhar, R. Fikes, and J. Rice. The ontolingua server: a tool for collaborative ontology construction. In *International Journal of Human-Computer Studies*, 1996.

[FL99] M. Fernández-López. Overview of methodologies for building ontologies. In *In IJCAI99 Workshop on Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends, Stockholm*, 1999.

[FLGPPSPS99] M. Fernández-López, A. Gómez-Pérez, J. Pazos-Sierra, and A. Pazos-Sierra. Building a chemical ontology using methontology and the ontology design environment. *IEEE Intelligent Systems*, 14(1):37–46, 1999.

[Fon01] R. L. Fontaine. A delta format for xml: Identifying changes in xml files and representing the changes in xml. In *Internationales Congress Centrum (ICC)*, May 2001.

[FP05] G. Flouris and D. Plexousakis. Handling ontology change: Survey and proposal for a future research direction. Technical Report FORTH-ICS/TR-362, Institute of Computer Science, FORTH, September 2005. Available at `http://www.ics.forth.gr/isl/publications/paperlink/fgeo_TR362.pdf`.

[GF95] M. Gruninger and M. S. Fox. Methodology for the design and evaluation of ontologies. In *In Workshop on Basic Ontological Issues in Knowledge Sharing, Montreal*, 1995.

[GHT06] T. Gardiner, I. Horrocks, and D. Tsarkov. Automated benchmarking of description logic reasoners. In *Proceedings of the 2006 Description Logic Workshop (DL 2006), volume 189 of CEUR*, 2006.

[GLP+07] A. Gangemi, J. Lehmann, V. Presutti, M. Nissim, and C. Catenacci. C-ODO: an OWL meta-model for collaborative ontology design. In *Workshop on Social and Collaborative Construction of Structured Knowledge (CKC 2007) at WWW 2007*, Banff, Canada, 2007.

[GPS99] A. Gangemi, D. M. Pisanelli, and G. Steve. An overview of the ONIONS project: Applying ontologies to the integration of medical terminologies. *Data Knowledge Engineering*, 31(2):183–220, 1999.

[Gru93] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[HA05] R. Heery and S. Anderson. Digital repositories review. Technical report, UKOLN/AHDS, February 2005. Available at `http://ahds.ac.uk/preservation/digital-repositories-review-2005.pdf`.

[Har06] J. Hartmann. Onthology - an ontology metadata repository. In *Poster session of the European Semantic Web Conference, ESWC'06*, 2006.

[HBP+07] P. Haase, S. Brockmans, R. Palma, J. Euzenat, and M. d'Aquin. Updated version of the networked ontology model. Technical Report D1.1.2, University of Karlsruhe, August 2007.

[HBS06] M. Hepp, D. Bachlechner, and K. Siorpaes. Ontowiki: community-driven ontology engineering and ontology usage based on wikis. In *WikiSym '06: Proceedings of the 2006 international symposium on Wikis*, pages 143–144, New York, NY, USA, 2006. ACM.

[Hef01] J. Heflin. *Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment*. PhD thesis, University of Maryland, 2001.

[HH00] J. Heflin and J. A. Hendler. Dynamic ontologies on the web. In *AAAI/IAAI*, pages 443–449, 2000.

[HJSS06] A. Hotho, R. Jschke, C. Schmitz, and G. Stumme. Bibsonomy: A social bookmark and publication sharing system. In *Proceedings*

of the Conceptual Structure Tool Interoperability Workshop at the 14th International Conference on Conceptual Structures, pages 87–102, 2006.

[HPGP09] J. Hartmann, R. Palma, and A. Gómez-Pérez. *Handbook of Ontologies (2nd edition)*, chapter Ontology Repositories. Springer: Berlin, 2009.

[HPS+05a] J. Hartmann, R. Palma, Y. Sure, P. Haase, and M. C. Suárez-Figueroa. OMV – ontology metadata vocabulary. In C. Welty, editor, *ISWC 2005 Workshop on Ontology Patterns for the Semantic Web*, NOV 2005.

[HPS+05b] J. Hartmann, R. Palma, Y. Sure, M. C. Suárez-Figueroa, P. Haase, A. Gómez-Pérez, and R. Studer. Ontology metadata vocabulary and applications. In R. Meersman, Z. Tari, and P. H. et al., editors, *International Conference on Ontologies, Databases and Applications of Semantics. In Workshop on Web Semantics (SWWS)*, LNCS 3762, pages 906–915. Springer, OCT 2005.

[HSV04] P. Haase, Y. Sure, and D. Vrandecic. D3.1.1 Ontology Management and Evolution: Survey, Methods and Prototypes. Technical Report D3.1.1, AIFB, University of Karlsruhe; Sekt Deliverable, December 2004. Available at `http://www.sekt-project.com/rd/deliverables/wp03/sekt-d-3-1-1-IncrementalOntologyEvolution.V1.pdf`.

[HvHH+05] P. Haase, F. van Harmelen, Z. Huang, H. Stuckenschmidt, and Y. Sure:. A framework for handling inconsistency in changing ontologies. In *Proc. of 4th International Semantic Web Conference (ISWC'05)*, pages 353–367. Springer, 2005.

[HW02] R. Heery and H. Wagner. A metadata registry for the semantic web. *D-Lib Magazine*, 8(5), 2002.

[Inm02] W. H. Inmon. *Building the Data Warehouse, 3rd Edition*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[ISO98] ISO. Ergonomic requirements for office work with visual display terminals (VDTs) - Part 11 : Guidance on usability. International Standard, International Organization for Standardization, March 1998. ISO Reference Number: ISO 9241-11:1998(E). Available at `http://www.idemployee.id.tue.nl/g.w.m.rauterberg/lecturenotes/ISO9241part11.pdf`.

[Jar05] M. Jarrar. *Towards Methodological Principles for Ontology Engineering*. PhD thesis, Vrije Universiteit Brussel, Brussels, Belgium, May 2005.

[KF01] M. Klein and D. Fensel. Ontology versioning for the semantic web. In *Proceedings of the International Semantic Web Working Symposium (SWWS'01)*, Stanford University, California, USA, August 2001.

[KFAC07] G. Konstantinidis, G. Flouris, G. Antoniou, and V. Christophides. Ontology evolution: A framework and its application to rdf. In *Proceedings of the SWDB-ODBIS07: Joint ODBIS & SWDB workshop on Semantic Web, Ontologies, Databases Collocated with VLDB2007, 23-24 September 2007, Vienna, Austria*, September 2007.

[KFK+02] M. Klein, D. Fensel, A. Kiryakov, N. F. Noy, and H. Stuckenschmidt. Versioning of distributed ontologies. Technical report, Vrije Universiteit Amsterdam, December 2002. Available at `http://wonderweb.semanticweb.org/deliverables/documents/D20.pdf`.

[Kle02] M. Klein. Ontology versioning and change detection on the web. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02)*, Siguenza, Spain, 2002.

[Kle04] M. Klein. *Change Management for Distributed Ontologies*. PhD thesis, Vrije Universiteit, Amsterdam), 2004.

[KN03] M. Klein and N. Noy. A component-based framework for ontology evolution. In *Proceedings of the IJCAI'03 Workshop: Ontologies and Distributed Systems*, Acapulco, Mexico, 2003.

[KPP+02] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, El, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *Software Engineering, IEEE Transactions on*, 28(8):721–734, 2002.

[KSKM07] K. Kozaki, E. Sunagawa, Y. Kitamura, and R. Mizoguchi. A framework for cooperative ontology construction based on dependency management of modules. In *Proceedings of the International Workshop on Emergent Semantics and Ontology Evolution (ESOE2007) at ISWC/ASWC2007, Busan, South Korea*, November 2007.

[KVH05a] V. Komulainen, A. Valo, and E. Hyvönen. A tool for collaborative ontology development for the semantic web. In *Proceedings of International Conference on Dublin Core and Metadata Applications (DC 2005)*, Nov 2005.

[KVH05b] V. Komulainen, A. Valo, and E. Hyvnen. A collaborative ontology development and service framework onki. In *Proceeding of ESWC 2005, poster papers*, 2005.

[LADS06] Y. Liang, H. Alani, D. Dupplaw, and N. Shadbolt. An approach to cope with ontology changes for ontology-based applications. In *In: Second Advanced Knowledge Technologies DTA Symposium, Aberdeen*, 2006.

[Ler00] B. S. Lerner. A model for compound type changes encountered in schema evolution. *ACM Transactions on Database Systems*, 25(1):83–127, 2000.

[LH90] B. S. Lerner and A. N. Habermann. Beyond schema evolution to database reorganization. In *OOPSLA/ECOOP '90: Proceedings of the European conference on object-oriented programming on Object-oriented programming systems, languages, and applications*, pages 67–76, New York, NY, USA, 1990. ACM Press.

[Lia06] Y. Liang. *Mini-Thesis: Enabling Active Ontology Change Management within Semantic Web-based Applications*. PhD thesis, University of Southampton, 2006.

[LM07] P. D. Leenheer and T. Mens. *Ontology Management. Semantic Web, Semantic Web Services, and Business Applications*, chapter Ontology Evolution. State-of-the-art and Future Directions. Springer, 2007.

[LTGP04] A. Lozano-Tello and A. Gómez-Pérez. ONTOMETRIC: A Method to Choose the Appropriate Ontology. *Journal of Database Management*, 15(2), 2004.

[MAC+06] P. Missier, P. Alper, O. Corcho, I. Kotsiopoulos, I. Dunlop, W. Xing, S. Bechhofer, and C. Goble. Managing semantic grid metadata in S-OGSA. In *Cracow Grid Workshop 2006*, Cracow, Poland, 2006.

[MGGPISK07] O. Muñoz-García, A. Gómez-Pérez, M. Iglesias-Sucasas, and S. Kim. A workflow for the networked ontologies lifecycle. A case study in FAO of the UN. In *Proceedings of the CAEPIA-TTIA 2007*, Spain, 2007. Springer.

[MMS+03a] A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz. An infrastructure for searching, reusing and evolving distributed ontologies. In *Proceedings of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 439–448. ACM, 2003.

[MMS03b] A. Maedche, B. Motik, and L. Stojanovic. Managing multiple and distributed ontologies on the semantic web. *VLDB Journal*, 12(4):286–302, 2003.

[MPdCSF+07] E. Montiel-Ponsoda, G. A. de Cea, M. Suárez-Figueroa, R. Palma, A. Gómez-Pérez, and W. Peters. LexOMV: an OMV extension to capture multilinguality. In *6th International Semantic Web Conference. In Workshop Ontolex07*, NOV 2007.

[MPSd06] D. Maynard, W. Peters, M. Sabou, and M. dÁquin. Change management for metadata evolution. In *International Workshop on Ontology Dynamics (IWOD) ESWC 2007 Workshop - 7 June - Innsbruck,*, June 2007-06.

[NCA08] N. F. Noy, A. Chugh, and H. Alani. The ckc challenge: Exploring tools for collaborative knowledge construction. *IEEE Intelligent Systems*, 23(1), 2008. Available at `http://eprints.ecs.soton.ac.uk/15036/`.

[NCLM06] N. Noy, A. Chugh, W. Liu, and M. Musen. A framework for ontology evolution in collaborative environments. In *International Semantic Web Conference*, pages 544–558, 2006.

[NIS04] NISO. Understanding metadata. NISO Press, National Information Stadards Organization, 2004. Available at `http://www.niso.org/publications/press/UnderstandingMetadata.pdf`.

[NK03] N. F. Noy and M. C. A. Klein. Tracking complex changes during ontology evolution. In *ISWC-2003 Poster Proceedings*, Sanibel Island, Florida, 2003. `http://www.stanford.edu/~natalya/papers/trackingChangesPoster.pdf`.

[NK04] N. F. Noy and M. Klein. Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems*, 6(4):428–440, July 2004.

[NKKM04] N. Noy, S. Kunnatur, M. Klein, and M. Musen. Tracking changes during ontology evolution. In *International Semantic Web Conference*, 2004.

[NM02] N. F. Noy and M. A. Musen. Promptdiff: A fixed-point algorithm for comparing ontology versions. In *National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence*, pages 744–750, Edmonton, Alberta, Canada, July 2002.

[NR89] G. T. Nguyen and D. Rieu. Schema evolution in object-oriented database systems. *Data Knowl. Eng.*, 4(1):43–67, 1989.

[OAS06] OASIS. Web Services Base Notification 1.3. Technical report, OASIS, October 2006.

[OK02] D. Ognyanov and A. Kiryakov. Tracking changes in rdf(s) repositories. In *EKAW '02: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, pages 373–378, London, UK, 2002. Springer-Verlag.

[Oli00] D. Oliver. *Change Management and Synchronization of Local and Shared Versions of a Controlled Vocabulary*. PhD thesis, Stanford University, 2000.

[OMG06] OMG. Meta Object Facility (MOF) Core Specification Version 2.0. OMG Available Specification, Object Management Group, January 2006. OMG Document Number: formal/06-01-01. Available at `http://www.omg.org/cgi-bin/doc?formal/2006-01-01`.

[OMG08] OMG. Ontology Definition Metamodel. OMG Adopted Specification - Draft 3, Object Management Group, September 2008. OMG Document Number: ptc/2008-09-07. Available at `http://www.omg.org/spec/ODM/1.0/Beta3/PDF`.

[OSSM99] D. E. Oliver, Y. Shahar, E. H. Shortliffe, and M. A. Musen. Representation of change in controlled medical terminologies. *Artificial Intelligence in Medicine*, 15:53–76, 1999.

[PBMT05] E. Paslaru Bontas, M. Mochol, and R. Tolksdorf. Case Studies on Ontology Reuse. In *Proceedings of the IKNOW05 International Conference on Knowledge Management*, 2005.

[Pfl95] S. L. Pfleeger. Experimental design and analysis in software engineering: Part 2: how to set up and experiment. *SIGSOFT Softw. Eng. Notes*, 20(1):22–26, 1995.

[PGmW95] Y. Papakonstantinou, H. Garcia-molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proceed-*

ings of the Eleventh International Conference on Data Engineering, pages 251–260, 1995.

[PH05] R. Palma and P. Haase. Oyster - sharing and re-using ontologies in a peer-to-peer community. In *International Semantic Web Conference*, pages 1059–1062, 2005.

[PHH08] R. Palma, J. Hartmann, and P. Haase. OMV - Ontology Metadata Vocabulary for the Semantic Web. Technical report, Universidad Politécnica de Madrid, University of Karlsruhe, 2008. Version 2.4. Available at `http://omv.ontoware.org/`.

[PHWd07] R. Palma, P. Haase, Y. Wang, and M. d'Aquin. D1.3.1 propagation models and strategies. Technical Report D1.3.1, UPM; NeOn Deliverable, November 2007. Available at `http://www.neon-project.org/`.

[PK97] A. Pons and R. K. Keller. Schema evolution in object databases by catalogs. In *IDEAS '97: Proceedings of the 1997 International Symposium on Database Engineering & Applications*, page 368, Washington, DC, USA, 1997. IEEE Computer Society.

[PM01] H. S. Pinto and J. P. Martins. A methodology for ontology integration. In *Proc. of the International Conf. on Knowledge Capture K-CAP01*, 2001.

[PÖ97] R. J. Peters and M. T. Özsu. An axiomatic model of dynamic schema evolution in objectbase systems. *ACM Transactions on Database Systems*, 22(1):75–114, 1997.

[PS87] D. Penney and J. Stein. Class Modification in the GemStone Object-Oriented DBMS. In *Proceedings of the 2nd ACM Conference on Object-Oriented Programming Systems, Langages and Applications*, pages 111–117, Orlando, Florida, October 1987.

[PSST04] S. Pinto, S. Staab, Y. Sure, and C. Tempich. Ontoedit empowering swap: a case study in supporting DIstributed, Loosely-controlled and evolvInG Engineering of oNTologies (DILIGENT). In *Proceedings of the First European Semantic Web Symposium, ESWS 2004*, pages 16–30, Heraklion, Crete, Greece, 2004.

[Raj97] V. Rajlich. A model for change propagation based on graph rewriting. In *In Proc. International Conference Software Maintenance*, pages 84–91. IEEE Computer Society Press, 1997.

[Rod95] J. F. Roddick. A survey of schema versioning issues for database systems. *Information and Software Technology*, 37(7):383–393, 1995.

[RP05] D. Rogozan and G. Paquette. Managing ontology changes on the semantic web. In *WI '05: Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 430–433, Washington, DC, USA, 2005. IEEE Computer Society.

[RR97] Y.-G. Ra and E. A. Rundensteiner. A transparent schema-evolution system based on object-oriented view technology. *IEEE Transactions on Knowledge and Data Engineering*, 9(4):600–624, 1997.

[RVMS99] T. Russ, A. Valente, R. Macgregor, and W. Swartout. Practical experiences in trading off ontology usability and reusability. In *Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW'99*, pages 16–21, 1999.

[S+01] H. Suguri et al. Implementation of fipa ontology service. In *Proc. of the Workshop on Ontologies in Agent Systems, 5th Int. Conf. on Autonomous Agents Montreal, Canada*, 2001.

[SAS03] Y. Sure, J. Angele, and S. Staab. Ontoedit: multifaceted inferencing for ontology engineering. *Journal on Data Semantics*, 2800:2003, 2003.

[SBF98] R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *Data Knowledge Engineering*, 25(1-2):161–197, 1998.

[SFGP08] M. Suárez-Figueroa and A. Gómez-Pérez. Neon methodology: Scenarios for building networks of ontologies. In *Proceedings of the 16th International Conference on Knowledge Engineering and Knowledge Management EKAW'08, poster papers*, 2008.

[SK03] H. Stuckenschmidt and M. C. A. Klein. Integrity and change in modular ontologies. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 900–908, 2003.

[SKKM02] E. Sunagawa, K. Kozaki, Y. Kitamura, and R. Mizoguchi. Management of dependencies between two or more ontologies in the environment for distributed development. *SIG-KBS*, 57:53–58, 2002.

[SKKM03] E. Sunagawa, K. Kozaki, Y. Kitamura, and R. Mizoguchi. An environment for distributed ontology development based on dependency management. In *International Semantic Web Conference*, pages 453–468, 2003.

[SMMS02] L. Stojanovic, A. Maedche, B. Motik, and N. Stojanovic. User-driven ontology evolution management. In *Proceedings of the 13th European Conference on Knowledge Engineering and Knowledge Management EKAW2002*, pages 285–300. Springer, 2002.

[SNTM08] A. Sebastian, N. F. Noy, T. Tudorache, and M. A. Musen. A generic ontology for collaborative ontology-development workflows. In *Proceedings of the 16th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2008)*, pages 318–328, 2008.

[SRNM07] K. Supekar, D. Rubin, N. Noy, and M. Musen. Knowledge zone: A public repository of peer-reviewed biomedical ontologies. In *In: Proceedings of the 12th World Congress on Health (Medical) Informatics; Building Sustainable Health Systems*, 2007.

[SSH02a] L. Stojanovic, N. Stojanovic, and S. Handschuh. Evolution of the metadata in the ontology-based knowledge management systems. In *Proceedings of the 1st German Workshop on on Experience Management*, pages 65–77. GI, 2002.

[SSH02b] N. Stojanovic, L. Stojanovic, and S. Handschuh. Evolution in the ontology-based knowledge management systems. In *Proceedings of the 10th European Conference on Information Systems, Information Systems and the Future of the Digital Economy, ECIS 2002*, Gdansk, Poland, 2002.

[SSSS01] S. Staab, R. Studer, H.-P. Schnurr, and Y. Sure. Knowledge processes and ontologies. *IEEE Intelligent Systems*, 16(1):26–34, 2001.

[STNM08] A. Sebastian, T. Tudorache, N. F. Noy, and M. A. Musen. Customizable Workflow Support for Collaborative Ontology Development. In *4th International Workshop on Semantic Web Enabled Software Engineering (SWESE) at ISWC 2008*, November 2008.

[Sto04] L. Stojanovic. *Methods and Tools for Ontology Evolution*. PhD thesis, University of Karlsruhe (TH), Germany, August 2004.

[Tem06] C. Tempich. *Ontology Engineering and Routing in Distributed Knowledge Management Applications*. PhD thesis, University of Karlsruhe (TH), Germany, 2006.

[TMN06] G. Tummarello, C. Morbidoni, and M. Nucci. Enabling semantic web communities with dbin: An overview. In *International Semantic Web Conference*, pages 943–950, 2006.

[TN07] T. Tudorache and N. Noy. Collaborative protege. In *Workshop on Social and Collaborative Construction of Structured Knowledge (CKC 2007) at WWW 2007*, Banff, Canada, 2007.

[TNTM08] T. Tudorache, N. Noy, S. Tu, and M. Musen. Supporting collaborative ontology development in protg. In *International Semantic Web Conference*, November 2008.

[UHW+98] M. Uschold, M. Healy, K. Williamson, P. Clark, and S. Woods. Ontology Reuse and Application. In *Proc. of the Int. Conf. on Formal Ontology and Information Systems FOIS98*, 1998.

[Völ06] M. Völkel. D2.3.3.v2 SemVersion: Versioning RDF and Ontologies. Technical Report D2.3.3v2, University of Karlsruhe; Knowledge Web Deliverable, January 2006. Available at `http://knowledgeweb.semanticweb.org/semanticportal/deliverables/D2.3.3v2.pdf`.

[vV98] E. P. van Veenendaal. Questionnaire based usability testing. In *In Proceedings of the European Software Quality Week, Brussels*, 1998.

[WHP07] Y. Wang, P. Haase, and R. Palma. D1.4.1 prototypes for managing networked ontologies. Technical Report D1.4.1, University of Karlsruhe; NeOn Deliverable, March 2007. Available at `http://www.neon-project.org/`.

[Yil06] B. Yildiz. Ontology evolution and versioning: The state of the art. Technical Report Asgaard-TR-2006-3, Vienna University of Technology. Institute of Software Technology & Interactive Systems (ISIS), October 2006. Available at `http://publik.tuwien.ac.at/files/pub-inf_4603.pdf`.

[ZB07] V. Zacharias and S. Braun. Soboleo - social bookmarking and lightweight ontology engineering. In *Workshop on Social and Collaborative Construction of Structured Knowledge (CKC), 16th International World Wide Web Conference (WWW 2007)*, May 2007.

[Zic92] R. Zicari. A framework for schema updates in an object-oriented database system. In *Building an object-oriented database system: the story of 02*, pages 146–182. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.