Change Representation For OWL 2 Ontologies

Raúl Palma[1,2], Peter Haase[3], Oscar Corcho[1], and Asunción Gómez-Pérez[1]

[1]Ontology Engineering Group, Laboratorio de Inteligencia Artificial
Facultad de Informática, Universidad Politécnica de Madrid, Spain
[ocorcho,asun]@fi.upm.es
[2]Current affiliation: Poznan Supercomputing and Networking Center, Poznań, Poland
rpalma@man.poznan.pl
[3]Institute AIFB, University of Karlsruhe, Germany
Current affiliation: fluid Operations, Walldorf, Germany
peter.haase@fluidops.com

Abstract. Ontologies are entities that evolve over time, therefore it is essential to represent and manage changes to ontologies along with the ontologies themselves. In this paper we propose a change ontology for the OWL 2 ontology language. This change ontology comprises a fine-grained taxonomy of ontology changes that considers the lowest-level atomic operations that can be performed in an ontology, but in addition also on other abstraction levels (ontology entity, composite). It thus allows to describe on a fine grained level how an ontology has changed from one version to another, and it also provides the vocabulary to talk *about* the changes that enables, for instance, to associate provenance or other rich metadata, such as argumentation structures. Additionally, we discuss some useful applications of our change ontology and its technological support.

1 Introduction

Ontologies are dynamic entities, i.e. they evolve over time. An ontology, defined as a formal, explicit specification of a shared conceptualization [15], 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.

Dealing with ontology changes involves the execution of many related tasks identified in the context of the ontology evolution process. Although this process has been modeled in the past in different ways (using different names and number of steps), existing proposals identify similar activities. A fundamental activity common to all approaches is the representation of ontology changes.

The representation of ontology changes has a major role in supporting the management of ontology evolution and related activities. For instance, in order to keep the track of ontology changes, they have to be captured and stored in an appropriate format. Moreover, 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 instances, mappings and metadata). Similarly, in a typical collaborative scenario, ontology editors may propose changes, while authoritative users approve/reject them following a well-defined process. Hence, the formal representation of changes provides the basis for storing changes in a machine-understandable format, facilitating their propagation and maintenance during the collaborative process.

Most of the works addressing the representation of ontology changes have proposed different change ontologies (e.g., [14], [6] and [9]). 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). Moreover, the works addressing the OWL ontology model have been developed based on the original OWL specification; none of them deals with the recent OWL 2 specification. As it has been pointed out in [14], the formal, explicit representation of changes makes them machine-understandable, usable by different ontology evolution systems, exploitable for supplementary functionality (e.g., learnability) and for reusing existing ontology technologies (e.g., reasoners).

In this paper we present a change ontology for the OWL 2 ontology language. This ontology has been implemented following a layered-approach for the representation of changes. The core of this approach consists of a generic change ontology that can be specialized for different ontology languages. For the development of the OWL 2 extension, a metamodel is used to refer to the OWL 2 elements. The metamodel facilitates the task of defining the relationship between the elements of the generic change ontology with the elements of the ontology model. The main contributions of this paper are: (i) a change ontology for the OWL 2 ontology model; (ii) a layered-model for the representation of changes that can be re-used for different ontology languages; (iii) a more fine-grained taxonomy of ontology changes, compared to the state of the art.

The remainder of this paper is organized as follows: Section 2 presents our change ontology for OWL 2. The first part of this section introduces our layered-approach for the representation of ontology changes, then the generic change ontology is described and, finally, the OWL 2 extension is described in detail. Section 3 introduces some useful applications of our model with pointers to the existing technological support. Section 4 provides an overview of related work regarding the change representation approaches, in particular for OWL. We conclude in Section 5.

2 Ontology Change Representation

Our proposal for the representation of changes in OWL 2 ontologies has been implemented following a layered-approach. In particular, we propose a *layered model for the representation of ontology changes* that integrates many of the features of existing change ontologies. The core of this model consists of a generic change ontology[1], *independent of the underlying ontology language*, 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 [3]). This generic change ontology defines some of its properties with unconstrained ranges *to avoid dependencies on*

[1] Available at `http://omv.ontoware.org/changes`.

a specific ontology language. The idea is to provide a common, independent change ontology that can be reused and specialized for specific ontology languages (Figure 1 illustrates the layered model specialized for OWL 2 - elements of the generic change ontology are in white whereas elements of the OWL 2 specialization are in grey).

Compared to existing approaches, we propose a more *fine-grained taxonomy of ontology changes*, including atomic, entity and composite changes. 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 (e.g., OWL axioms). This lower level provides 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. Our entity change corresponds to the elementary (atomic) change in the literature (e.g., [14] and [6]) 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., [14] and [7]), *composite changes* represent a group of changes applied together that constitute a logical entity, such as group a set of classes or 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 the provenance of changes, that is, *when the change was made, who made it, and how it was made*. 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).

The generic change ontology has been implemented as an extension of the OMV ontology metadata model [5]. OMV defines a vocabulary for describing ontologies and related entities. In our context, we decided to use OMV due to two reasons: first, 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.

2.1 Generic Change Ontology

The main classes and properties of the generic change ontology are illustrated in Figure 1 using white nodes (the prefix *omv* is used to refer to OMV metadata model elements). 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 classified into additive changes (`Addition`) and removal changes (`Removal`), which represent the minimal operations that can be performed in an ontology. Update operations are not considered as atomic changes, as they can be represented by combining a removal and an additive change.

Following the approach proposed by [6], 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 atomic changes, we associate both classes using the property `consistsOfAtomicOperation`, and to express that a composite change is a combination of other changes we define the property `consistsOf`. Additionally, to group all the changes made to a particular ontology version (see [13]) 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 a generic class `Agent`. An agent can be, for instance, a person (represented with the `Person` class from the OMV core) or some software artifact. 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[2]. Finally, similar to [14] 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 change) and the relevance describes whether and how the change can fulfil the requirements.

[2] In a multi-user editing environment, concurrent changes from different users may be intertwined for different operations because the object property `hasPreviousChange` is meant to keep changes (atomic/entity/composite) in chronological order and, therefore, it points to the previous change globally. However, thanks to the `consistsOfAtomicOperation` and `consistsOf` object properties, it is always possible to know the atomic changes for a high-level operation (e.g., an entity/composite change). See [10] for additional information.

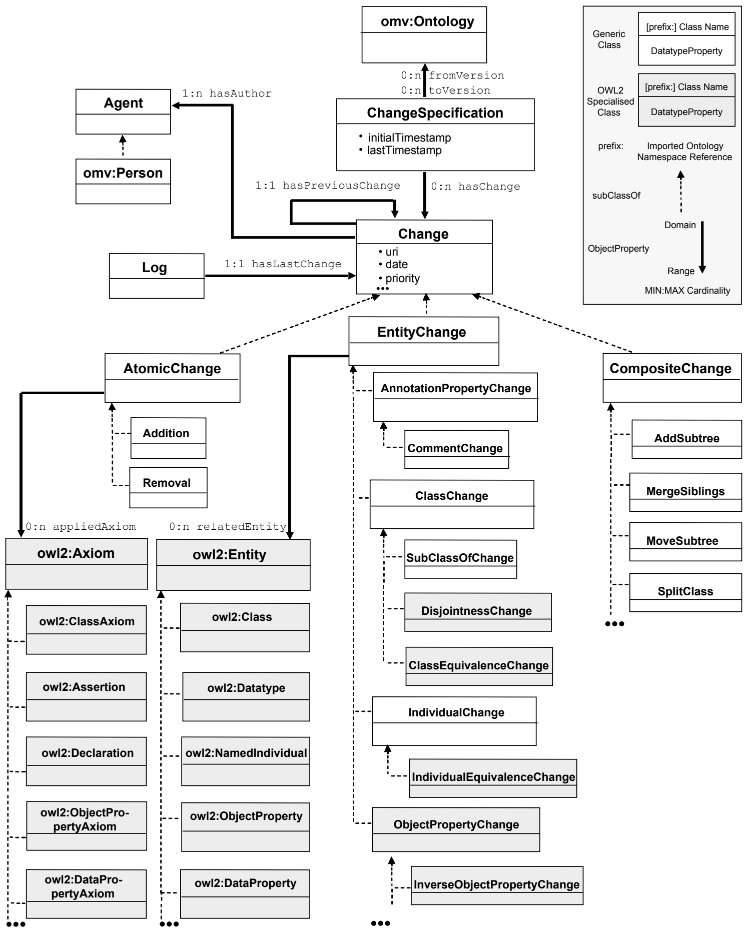

Fig. 1. Main Classes and Properties of OWL 2 Change Ontology

2.2 OWL 2 Change Ontology Extension

To specialize the generic change ontology to a specific ontology language, our approach requires a metaontology about the ontology language, i.e., a vocabulary to *talk about* the elements of an ontology. For this purpose, we have created a metamodel that captures the structure of OWL 2 ontologies. The OWL 2 metamodel has a direct correspondence to the structural specification of OWL 2 and is defined using a metamodeling approach based on MOF (Metaobject Facility[3]). We have made the metamodel available in various formats, including in the form of a metaontology in OWL[4].

The main purpose of the metaontology is to describe elements of OWL 2 ontologies, i.e., its axioms, entities, etc. For example, a class assertion is described in the meta-

[3] http://www.omg.org/mof
[4] http://owlodm.ontoware.org/OWL2

ontology using the following axiom, which states that a class assertion is a kind of assertion and that it has exactly one class description and one individual. For a full specification of the OWL 2 metamodel, we refer the reader to [4].

```
SubClassOf( owl2:ClassAssertion
  ObjectIntersectionOf( owl2:Assertion
    ObjectExactCardinality( 1 owl2:classExpression owl2:ClassExpression )
    ObjectExactCardinality( 1 owl2:individual owl2:Individual )))
```

To extend the generic change ontology to build the OWL 2 change ontology[5], we had to perform two further 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 *Axiom* class and the *Entity* class from the OWL 2 metaontology. Figure 1 illustrates the main classes and properties of the OWL 2 change ontology using grayed nodes (the prefix *owl2* is used to refer to OWL 2 metaontology elements). For reasons of space, the taxonomy of OWL 2 axioms in the figure only shows the top level elements of the OWL 2 metaontology. The set of axioms at the leaf level of this taxonomy represents the possible atomic operations that can be performed over an OWL 2 ontology, i.e. the, `Addition` or `Removal` of specific axioms.

Second, the taxonomy of entity-level changes has been extended to model the particular changes for OWL 2. 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 split class into multiple siblings) and, therefore, they are modeled in the generic change ontology. Figure 1 illustrates part of the taxonomy of entity changes specialized for OWL 2.

Illustrative example To illustrate the usage of our change ontology, imagine an ontology editor (Steve) responsible for the maintenance of a products ontology in company ACME. On the arrival of a new product (iPhone3GS), Steve will have to add a new individual of the class *Product*. In this example, the entity change *Add Individual* consists of two atomic changes, i.e., *Declaration* and *Class Assertion*. The entity change is associated to the involved ontology elements (Product and iPhone3GS), and the atomic changes are associated to the corresponding OWL 2 axioms. An extract of the change ontology individuals created is as follows[6]:

```
Prefix(change=http://omv.ontoware.org/changes#)
Prefix(owl2=http://owl2.ontoware.org/OWL2#)
Prefix(acme=http://www.acme.org/productsACME.owl#)

ClassAssertion(change:AddIndividual _:a1)
DataPropertyAssertion(change:date _:a1 06/08/09 11:41:42 AM)
ObjectPropertyAssertion(change:hasAuthor _:a1 :Steve)
```

[5] Available at `http://omv.ontoware.org/OWLChanges`.

[6] In [10] we have evaluated the completeness of the change ontology in a simulated scenario and its adequacy in a real scenario. In these scenarios, one atomic ontology change, typically, generated around 10 facts in the change ontology.

```
ObjectPropertyAssertion(change:consistsOfAtomicOperation _:a1 _:b1)
ObjectPropertyAssertion(change:consistsOfAtomicOperation _:a1 _:b2)
ObjectPropertyAssertion(change:hasPreviousChange _:a1 _:a0)
ObjectPropertyAssertion(change:hasRelatedEntity _:a1 acme:Product)
ObjectPropertyAssertion(change:hasRelatedEntity _:a1 acme:iPhone3GS)

ObjectPropertyAssertion(change:appliedAxiom _:b1 _:c1)
ObjectPropertyAssertion(change:hasPreviousChange _:b1 _:ax)
ObjectPropertyAssertion(change:appliedAxiom _:b2 _:c2)
ObjectPropertyAssertion(change:hasPreviousChange _:b2 _:ay)

ClassAssertion(owl2:Declaration _:c1)
ObjectPropertyAssertion(owl2:entity _:c1 acme:iPhone3GS)
ClassAssertion(owl2:ClassAssertion _:c2)
ObjectPropertyAssertion(owl2:classExpression _:c2 acme:Product)
ObjectPropertyAssertion(owl2:individual _:c2 acme:iPhone3GS)
```

3 Applications and Technological Support

The change representation model is a core component in the management of ontology evolution. The possible applications from such a model range from the capturing of ontology changes, over the argumentation of changes, the analysis of change history, the propagation/synchronization of changes in a distributed scenario, to the collaborative ontology development process support (to mention a few). In the remainder of this section we discuss some of these useful applications which have been implemented within the NeOn Toolkit[7], an extensible ontology engineering environment based on Eclipse, by means of a set of plugins and extensions[8]. The NeOn Toolkit relies on the distributed registry Oyster[9] [11] for the storage and maintenance of ontology changes, i.e., individuals of our change ontology. All the following applications rely-on/use the OWL 2 change ontology described in this paper.

Ontology change capturing. In order to keep track of the evolution of an ontology, changes need to be captured and stored. Thus, for this task, the change ontology allows storing changes about ontologies 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[10]. Consequently, 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

[7] `http://www.neon-toolkit.org`

[8] Available at `http://www.neon-toolkit.org/wiki/Neon_Plugins`.

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

[10] In a different (non-controlled) scenario where the history of changes is not available, different ontology versions should be compared to compute the *diff* between them. We refer the reader to [12] for a discussion on how to use our model for this task.

change is represented according to the change ontology by creating the appropriate individual(s). In our model, depending on the type of change, it will represented with an individual of the generic change ontology or an individual of the specialized change ontology. Finally, the change individual(s) is stored in an appropriate place (e.g., registry) for future processing and propagation by the ontology change logger (step 5). Note that the task of storing a change individual involves additional subtasks, such as updating the log history keeping track of the chronological order of changes. In our model, the chronological order can be maintained by means of two elements: the object property `hasPreviousChange` to keep the link between different changes, and the `Log` class to point to the last change in the ontology history.

In the NeOn Toolkit, a change capturing plugin implements the previous process, whereas the additional subtasks are responsibility of Oyster.

Ontology change argumentation. The change representation model can also be used 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. Using our model, changes (at different level of granularity) can be associated to the discussion related to the reasoning (e.g., issues addressed) of implementing the changes.

In the NeOn Toolkit, the Cicero tool has been integrated with the ontology editor for the argumentation of ontology changes. Cicero facilitates efficient discussions and accelerates the convergence to decision [1].

Ontology change propagation/synchronization in a distributed scenario during collaborative ontology development. One of the goals of propagating ontology changes in a distributed scenario is to keep distributed copies of the same ontology synchronized, thus allowing the notification of new changes to ontology editors. For this task, changes should be propagated to each node in the distributed network that maintains a copy of the ontology (and wants to receive those changes). The propagation/synchronization of changes supports different collaborative scenarios, typical in an organizational setting, where the management of the ontology and its related changes can be centralized, distributed or be a hybrid between both of them. However, in order to propagate changes, first we need to have those changes stored in an appropriate format (e.g., change ontology individuals). Additionally, for the synchronization, the change histories in different nodes have to be compared. Our model support this task by means of the `hasPreviousChange` object property, the `Log` class, and the set of changes for a particular ontology (version) kept by the `changeSpecification` class.

The NeOn Toolkit relies on Oyster, that implements a combination of a push and pull mechanisms, for the synchronization of changes. This process can be triggered from the change capturing plugin introduced above.

Collaborative ontology development process support. 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. This collaborative process can be formalized by means of a collaborative workflow model. A collaborative workflow allows modeling relations among designers, ontology elements and collaborative tasks (according to [2]).

The collaborative ontology development process highly depends upon the operations performed to the ontology itself (the changes performed by ontology editors). Hence, the change representation model facilitates the representation of the tight relationship that exists between ontology changes and the collaborative process elements. Using our model, changes (at the entity level) can be associated to the corresponding action of the collaborative process. Moreover, for each of those changes, the corresponding state (e.g., Draft, To Be Approved, Approved) can be associated to support the collaborative process. Besides, our model allows to keep the track of the users involved in the modification of the ontology by means of the `hasAuthor` object property.

In the NeOn Toolkit, the workflow feature supports the collaborative development of ontologies following a typical workflow.

4 Related Work

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[11] and complete[12], and, typically, the outcome of this activity is an ontology for representing ontology changes, i.e., change ontology.

We can identify two main works for the representation of ontology changes: [14] and [6]. 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 or extend the previous models. In particular, for the OWL ontology model, the change ontology proposed in [6] is based on previous efforts by the same authors and other colleagues (e.g., [7] and [9]) and it has been extended in later works (e.g., [8]).

To summarize, 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 and in some cases these elementary changes are not even minimal, for example, they include *modify* operations. Furthermore, *existing approaches are dependent on the underlying ontology model*, that is, they are based on proprietary models (e.g., KAON [14]) or they are developed for specific languages (e.g., OWL [6]), consequently, they have different sets of elementary (atomic) changes. Finally, the works addressing the OWL ontology language are still based on the original OWL specification.

5 Conclusion

In this paper we have presented a proposal for the representation of changes in OWL 2 ontologies. Our contribution has been implemented following a layered-approach that

[11] The manipulation operators should only generate valid ontologies.
[12] The set should subsume every possible type of ontology access and manipulation.

consists of a generic change ontology that has been specialized for the OWL 2 ontology language. Our generic change ontology models operations that are expected to be supported by any ontology language and also includes information for the tracking of changes and for the support of decisions by ontology editors. We also showed how the OWL 2 extension was developed using a metamodel to refer to the OWL 2 elements. Our proposed ontology considers a more fine-grained taxonomy of ontology changes compared to existing change ontologies. This supports a more efficient processing of changes (e.g., to implement redo/undo operations and comparison between ontology versions) and provides more detailed information of how an ontology changed as well as the specific reasons/consequences of operations at a higher level (e.g., to keep the argumentation or state of a composite/entity change). Furthermore, we discussed some useful applications of our model and its technological support. In particular, we introduced the ontology registry Oyster that is capable of storing and maintaining individuals of the change ontology, and we introduced software components, implemented within the NeOn Toolkit, that rely or support our proposed model.

Acknowledgments This work has been supported by the European Commission under the NeOn Project (FP6-027595).

References

1. K. Dellschaft, H. Engelbrecht, J. M. Barreto, S. Rutenbeck, and S. Staab. Cicero: Tracking design rationale in collaborative ontology engineering. In *European Semantic Web Conference*, pages 782–786. Springer, 2008.
2. A. Gangemi, J. Lehmann, V. Presutti, M. Nissim, and C. Catenacci. C-ODO: an OWL metamodel for collaborative ontology design. In *Workshop on Social and Collaborative Construction of Structured Knowledge (CKC 2007) at WWW 2007*, Banff, Canada, 2007.
3. T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.
4. P. Haase, R. Palma, and M. d'Aquin. Updated version of the networked ontology model. Technical Report D1.1.5, NeOn Project Deliverable, February 2009.
5. 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.
6. M. Klein. *Change Management for Distributed Ontologies*. PhD thesis, Vrije Universiteit, Amsterdam), 2004.
7. 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.
8. 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.
9. N. F. Noy and M. C. A. Klein. Tracking complex changes during ontology evolution. In *ISWC-2003 Poster Proceedings*, Sanibel Island, Florida, 2003.
10. R. Palma. *Ontology Metadata Management in Distributed Environments*. PhD thesis, Universidad Politécnica de Madrid, Spain, 2009.
11. 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.
12. 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/`.
13. 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/`.
14. L. Stojanovic. *Methods and Tools for Ontology Evolution*. PhD thesis, University of Karlsruhe (TH), Germany, August 2004.
15. R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *Data Knowledge Engineering*, 25(1-2):161–197, 1998.