Web Semantics: Science, Services and Agents on the World Wide Web 9 (2011) 299–314

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: http://www.elsevier.com/locate/websem

A holistic approach to collaborative ontology development based on change management

Raúl Palma [a,*], Oscar Corcho [a], Asunción Gómez-Pérez [a], Peter Haase [b,1]

[a] Ontology Engineering Group, Departamento de Inteligencia, Artificial, Facultad de Informática, Universidad Politécnica de, Madrid, Campus de Montegancedo sn, Boadilla del Monte 28660, Spain
[b] Institute AIFB, KIT, D-76128 Karlsruhe, Germany

ARTICLE INFO

Article history:
Available online 30 June 2011

Keywords:
Collaborative ontology development
Ontology changes
Ontology metadata

ABSTRACT

This paper describes our methodological and technological approach for collaborative ontology development in inter-organizational settings. It is based on the formalization of the collaborative ontology development process by means of an explicit editorial workflow, which coordinates proposals for changes among ontology editors in a flexible manner. This approach is supported by new models, methods and strategies for ontology change management in distributed environments: we propose a new form of ontology change representation, organized in layers so as to provide as much independence as possible from the underlying ontology languages, together with methods and strategies for their manipulation, version management, capture, storage and maintenance, some of which are based on existing proposals in the state of the art. Moreover, we propose a set of change propagation strategies that allow keeping distributed copies of the same ontology synchronized. Finally, we illustrate and evaluate our approach with a test case in the fishery domain from the United Nations Food and Agriculture Organisation (FAO). The preliminary results obtained from our evaluation suggest positive indication on the practical value and usability of the work here presented.

1. Introduction

Ontology development is transforming from a process traditionally performed by isolated ontology engineers or domain experts into a process performed collaboratively by mixed teams, who may be geographically distributed and play different roles in the process. For example, some domain experts acting as editors may propose changes in specific parts of the ontologies, while other domain experts acting as authoritative users may approve or reject them following a well-defined editorial process. Similarly, ontology engineers acting as knowledge architects may propose remodeling parts of the ontologies to follow specific ontology design patterns, which have to be approved by authoritative users.

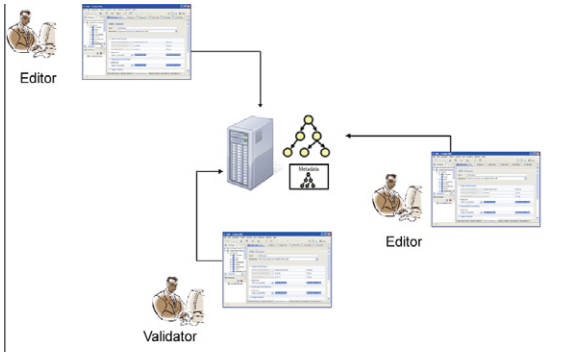
Classical ontology engineering methodologies (e.g., METHONTOLOGY [1], On-To-Knowledge [2]) do not consider this distributed setting and propose most of their ontology development activities and tasks for isolated ontology engineers or domain experts working locally with their ontologies. This is also the case for most of the existing ontology development tools (e.g., Protégé,[2] TopBraid Composer[3] SWOOP[4]), which mainly support the single-developer scenario, where only one user is involved in the development and later modification of the ontologies. Only in some cases they provide additions or plugins to support groups of ontology developers working in collaboration (e.g., Collaborative Protégé[5] and Hozo[6]).

Other more recent methodologies or approaches, such as DILIGENT [3] and DOGMA-MESS [4], consider that ontologies can be developed collaboratively in distributed settings. However, they normally focus on centralized approaches for ontology and change management, where a main copy of the ontology is maintained and updated by groups of ontology developers that may work with local adaptations. More importantly, even though they consider some methodological aspects involved in collaborative ontology development, in general they focus less on the varied approaches followed by different organizations to coordinate this process. For example, they only propose simple conceptual models describing how ontology developers may use a shared (aka. main) copy of ontology, create local adaptations, and then, somehow, other

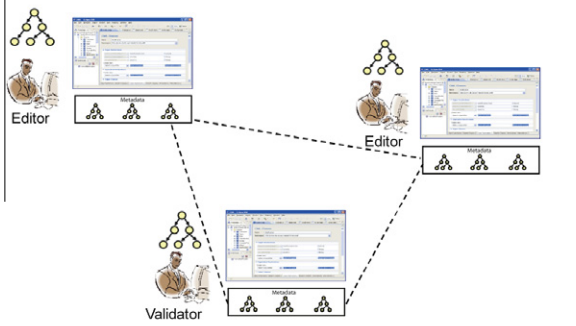
* Corresponding author. Present address: Poznan Supercomputing and Networking Center, ul. Dabrowskiego 79a, 60-529 Poznan, Poland. Tel.: +48 61 858 21 60.
E-mail addresses: rpalma@man.poznan.pl (R. Palma), ocorcho@fi.upm.es (O. Corcho), asun@fi.upm.es (A. Gómez-Pérez), peter.haase@fluidops.com (P. Haase).
[1] Present address: Fluid Operations AG, Altrottstr. 31 69190, Walldorf, Germany.

[2] http://protege.stanford.edu/.
[3] www.topbraidcomposer.com/.
[4] http://code.google.com/p/swoop/.
[5] http://protegewiki.stanford.edu/wiki/Collaborative_Protege.
[6] http://www.hozo.jp/.

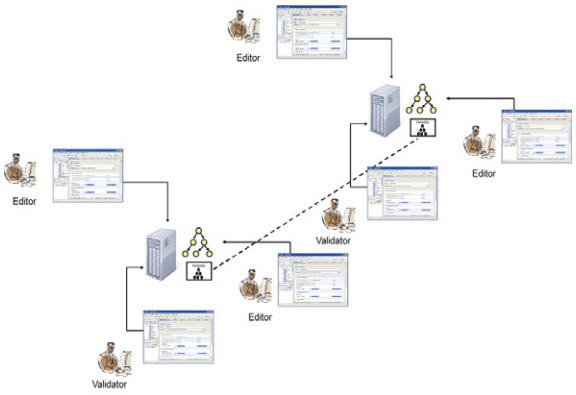
1570-8268/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.websem.2011.06.007



(a) Scenario A: Centralized Ontology Management.



(b) Scenario B: Distributed Ontology Management.



(c) Scenario C: Hybrid Ontology Management.

Fig. 1. Typical scenarios for the management of ontologies and their changes during collaborative ontology development.

authoritative users will collect, select and merge the proposed changes to create a new version of the main copy.

Different organizations may follow different approaches for collaborative ontology development. Examples of such collaborative development processes can be found in international institutions like the United Nations Food and Agriculture Organization (FAO),[7] which is developing and maintaining large ontologies in the fishery domain [5], or the World Health Organization (WHO), which is developing and maintaining large ontologies and classifications in the medical domain, such as ICD[8] (International Classification of Diseases) or ICPS[9] (International Classification for Patient Safety). Another similar example is the Gene Ontology (GO) project,[10] which addresses the need for consistent descriptions of gene products in different databases.

Fig. 1 identifies a number of representative scenarios for the management of ontologies and their changes during collaborative ontology development. The continuous lines represent a permanent (tight) connection between the editors and the centralized copy of the ontology, whereas the dotted lines represent a temporary (loose) connection between the editors and the distributed copies of the ontology. At one end (Scenario A), in a permanent online environment, ontology developers may work collaboratively using a centralized copy of the ontology. This model is supported, for example, by WebProtege[11] and the more recent iCAT platform.[12] At the other end (Scenario B), in a mostly offline environment, they may collaborate using distributed local copies of the ontology that are sporadically synchronized, allowing ontology developers to collaborate even without a permanent and reliable connection, or if not enough centralized resources are available. This is supported, for example, by our collaborative ontology development extensions in NeOn Toolkit, implemented with a set of plugins[13] (described in detail in Section 6). Hybrid approaches (Scenario C), partly centralized and partly distributed, are also possible.

In all these collaborative ontology development scenarios, there is a need to coordinate the actions of all the developers (for example, when do editors want their changes to be reviewed, what kind of actions can they perform according to their roles, etc.). Therefore, we need appropriate models, strategies and infrastructure to support those coordination tasks. This is one of the two major contributions of our work. Moreover, since the whole collaborative ontology development scenario is driven by the ontology changes, there is a need to support the management of those changes in distributed settings, including their formal representation, manipulation and propagation. This is the second major contribution of our work.

The remainder of this document is organized as follows: after a summary of the state of the art (Section 2), we provide a high level overview of our solution (Section 3). Then, we describe our contributions for collaborative ontology development (Section 4) and for ontology change management in distributed environments (Section 5). Then, in Section 6 we present the technological support for the work here presented. In Section 7 we present the evaluation experiments, in Section 8 we summarize and discuss the results. We conclude in Section 9 and discuss directions for future research in Section 10.

2. Overview of the state of the art

In this section we briefly discuss the main limitations and challenges encountered for the core topics of this work, namely, collaborative ontology development and ontology change management. A more comprehensive state of the art of all these activities can be found in [6].

Previous efforts to support collaborative ontology development produced relevant methodological and technological results (e.g., [7,4,8–16]). However, existing solutions support only a centralized management of the ontology and its related changes. In some cases (e.g., [7] and [4]), a shared (aka. main) copy of the ontology can be adapted or specialized by distributed users; in other cases (e.g., [9] and [10]), a main copy of the ontology is divided into sub-ontologies each of them modified by distributed users; and in other cases (e.g., [8,11–13,15]) 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, in general they do not support the processes typically followed by organizations for the coordination of change

[7] http://www.fao.org/.
[8] http://www.who.int/classifications/icd/en/.
[9] http://www.who.int/patientsafety/taxonomy/en/.
[10] http://www.geneontology.org/.
[11] http://protegewiki.stanford.edu/wiki/WebProtege.
[12] http://sites.google.com/site/icd11revision/home/icat.
[13] http://neon-toolkit.org/wiki/1.x/Workflow_Support, http://neon-toolkit.org/wiki/1.x/Change_Capturing and http://neon-toolkit.org/wiki/1.x/Oystermenu.

proposals, which specifies the type of actions or operations ontology editors can perform depending on their role and the state of the ontology. Some early conceptual efforts were presented in [7] and [4]. Besides, a notable exception is the work from [14]—and derivative works—which provided recently a proposal, although without any technological support.

Furthermore, in many of them (e.g., [7,4,10,11]) there is not even a formal management of ontology changes (e.g., explicit representation and propagation). And in summary, there is no integrated approach (and technological support) that addresses all the previous issues.

In contrast to existing approaches, first, our solution is based on a formalization of the collaborative development process followed by organizations for the coordination of change proposals. Second, while there is a poor management of ontology changes in many existing solutions, our solution is supported by models, methods and strategies for change management in distributed environments. Third, unlike existing approaches that support only a centralized management of the ontology and its related changes, our solution supports both a centralized and a distributed management. Finally, we provide an integrated approach (and technological support), currently not available, which addresses all the previous issues.

Regarding the management of changes, we limit our discussion to the main activities that we address: change representation, change propagation and tool support.

For the representation of ontology changes two main works can be identified: Stojanovic's [17] and Klein's [18]. They classify changes in a similar manner (atomic or elementary, composite and complex), where atomic changes refer to operations at the entity level. Some other works in the literature resemble (e.g., [19] and [20]) or extend (e.g., [21] and [12]) the previous models or provide partial solutions (e.g., [22–24]). All of these, however, 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., classes, 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. In our work, we propose a change representation model that is independent of the underlying ontology language, and that includes a more fine-grained taxonomy of ontology changes in comparison to existing models, which identifies the real low-level operations that can be performed in an ontology.

With respect to the propagation of changes, it has only been considered in the past to related ontologies (e.g., [17,25] and [9]), to ontology individuals (e.g., [26,27]), and to some extent to related applications ([28]). For the propagation of changes to related ontologies, existing approaches consider only a central (main) copy of the ontology that is either replicated (e.g., [17]) or divided into several component ontologies (e.g., [9]) 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 supported yet. Our work addresses exactly that issue, enabling a distributed management of the ontology and its changes.

Finally, existing tools (that do not tackle additional collaborative aspects) support different aspects of the change management. Many of them are focused on the management of ontology versions (e.g., [29–31]). Nevertheless, some tools address evolution aspects, such as the effect of changes on dependent applications (e.g., [28]), the tracking of changes along with their formal representation, and the propagation task (e.g., [17,32,31]). 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. The technological support that we provide, integrates our solutions for the management of changes in distributed environments, including the tracking, representation and propagation of changes from any copy of the ontology to other copies.

3. Our approach in a nutshell

This paper presents a holistic approach to support collaborative ontology development in inter-organizational settings. Our work is based on models, methods and strategies for the management of ontology changes in distributed environments.

The diagram in Fig. 2 depicts a general overview of the paper contributions and the relationship between them. In the diagram, the two boxed areas represent the two research areas that we address; the components in the outer box rely on the components of the inner box. The way the components are related to each other is illustrated by the arrows.

As presented in the figure, the first major contributions of this work (described in Section 4) starts with the formalisation of the collaborative development process as an editorial workflow (Sections 4.3 and 4.4), which can be described as a special case of epistemic workflow[14] characterized by the ultimate goal of designing networked ontologies and by specific relations among designers, ontology elements, and collaborative tasks [33]. The need for such workflows has also been acknowledged in the past in other related works (e.g., [13]). Then we focus on the management and enactment of such editorial workflows (Section 4.5), which include the execution of several tasks that have to be carried out to provide a complete solution that supports collaborative ontology development, such as the enforcement of process constraints. Finally, our contribution includes an integrated infrastructure for collaborative ontology development (described in Section 6) that implements all the models, methods and strategies here proposed.

Our second major contribution is the management of ontology changes in distributed settings (described in Section 5), which is central to our approach since the whole collaborative ontology development scenario is driven by them. The contributions here can be divided into: (i) those related to the representation of changes (Section 5.1), including how they can be identified, and at which level of granularity, from high level changes that identify the basic parameters in which an ontology has changed, including its metadata, to low level changes that identify the changes that were produced at the axiom level and which are highly dependent on the ontology implementation language; (ii) those related to their manipulation (Section 5.2), including how changes can be captured or how the consistency of the ontology is ensured after those changes; and finally, (iii) those related to change propagation (Section 5.3) in distributed settings. As a fundamental characteristic of our approach, all these contributions on ontology change management build on OMV[15] [34], a metadata vocabulary that captures relevant information about ontologies such as provenance, availability, statistics, etc.

4. Collaborative ontology development supported by change management

This section presents our solution to support collaborative ontology development in distributed settings. It addresses the main limitations in existing approaches as discussed in Section 2.

[14] Epistemic workflows describe the flow of knowledge from one rational agent to another [16].

[15] http://omv.ontoware.org.

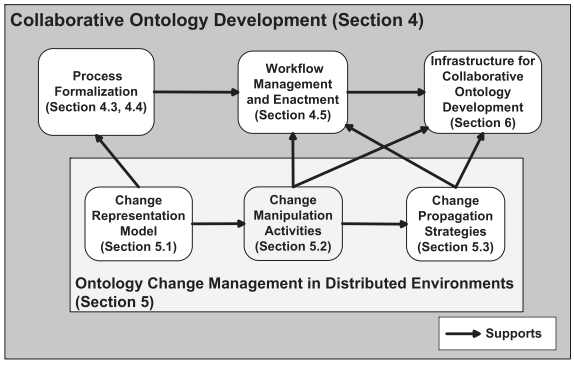


Fig. 2. Overview of our contributions and the relationship between them.

In order to achieve our goal, we first identified the most relevant requirements to support collaborative ontology development in distributed settings, based on the analysis of the processes typically followed by organizations in the development and maintenance of ontologies (Section 4.1). For this analysis, we considered different processes and scenarios for collaborative ontology development and previous efforts in the state of the art. We will illustrate the outcome of this analysis, as well as the models and strategies proposed, with a concrete example (described in Section 4.2) based on the fisheries ontologies lifecycle from FAO [5]. In particular, Section 4.3 describes 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, in Section 4.4 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 4.5 we describe the strategies for the management and enactment of the workflow during ontology development.

4.1. Analysis of collaborative ontology development

As mentioned before, the collaborative development of ontologies usually follows a pre-defined process that specifies who (depending on the user role), when (depending on the ontology state) and how (which actions or operations users can perform) an ontology can change. However, the details of this process, as well as the configuration of the collaborative scenario, can vary from one organization or context to another.

Fig. 1 already presented three representative scenarios for collaborative ontology development. 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. Fig. 1(a)), ontology and change management is centralized. There is only one copy of the ontology, stored in a central server, which can be edited by all (distributed) members of the team from their (remote) node. The change information about the ontology is also stored in a central registry.

(ii) In *Scenario B* (c.f. Fig. 1(b)), ontology and change management is distributed. There are "n" copies of the ontology, where "*n*" is the number of ontology editors working on their own local copy of the ontology. Change information is also stored in a distributed manner (e.g., in a local registry of each member). The local nodes exchange information about ontology metadata and synchronize 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. Fig. 1(a)) is a hybrid between scenarios "A" and "B". The team of ontology editors can be divided in at least two different groups. Each group works 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 nodes in configuration B.

Since 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), part of our work was targeted for supporting them.

From this analysis, as well as from previous efforts in the state of the art (e.g., [12]), we derived a set of requirements [35]. Briefly, *lifecycle* requirements address the coordination of the change proposals enabling ontology editors to consult, modify, validate or publish ontologies depending on their role and the state of the ontology. *Activity* requirements deal with the activities required to be supported, such as edit ontology elements, change state of elements, etc. *Visualization* requirements are those concerned with enabling users to consult the corresponding information generated by those actions, such as view the change history. *Change management* requirements are related to the representation, capturing, propagation and notification of changes. *Versioning* requirement deals with the identification and tracking of different ontology versions. Finally, we identified requirements related to *concurrency control*, such as the coordination of changes applied in distributed ontology copies, necessary to support the collaboration among editors during ontology development.

In the remainder of this section we provide solutions to support the lifecycle and activity requirements. In Section 5 we address the management of changes, the management of different ontology versions and the concurrency control by the synchronization of local ontology copies. Finally, in Section 6, we address the visualization requirements with our technological support.

4.2. Running example

Ontology developers at FAO 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 fishery ontologies and their lifecycle.

Several actors are involved in the ontology development process, such as, experts in ontology modeling who, are in charge of defining the original skeleton of the ontology, and ontology editors, who are in charge of the everyday editing and maintenance of the ontologies. Ontology editors include subject experts who know about the domain to be modeled and validators who, besides being domain experts, can move a change to production state for external availability. Ontology development follows a well-defined 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 as final, they are authorized to release it and make it available to end users and systems.

4.3. Workflow model

In our approach, we consider the collaborative development process at two different levels of abstraction: the (ontology) element level and the ontology level. In each level, we must model

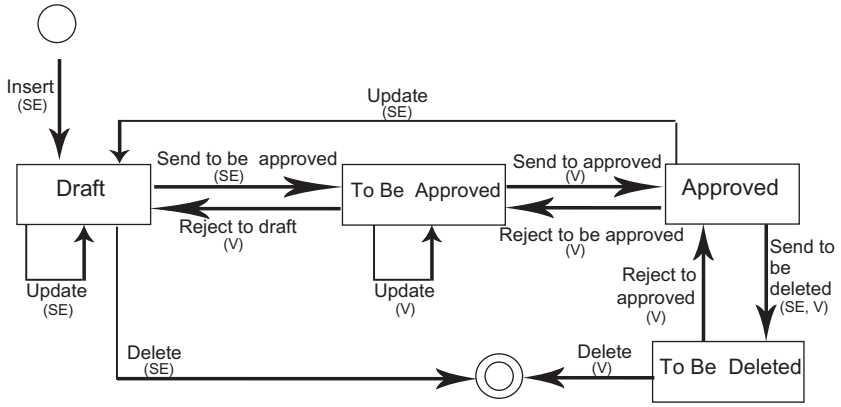


Fig. 3. Collaborative editorial workflow at the element level for the running example.

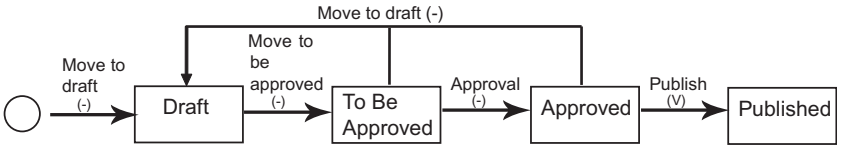


Fig. 4. Collaborative editorial workflow at the ontology level for the running example.

the way in which the different elements involved in this process (roles of designers, states of ontologies and elements, available collaborative actions) are related to each other.

Hence, we propose the use of *collaborative editorial workflows* to formalize the collaborative ontology development process. In our solution, we consider that an ontology consists of a set of axioms and facts, which can relate to entities such as classes, properties or individuals. These entities constitute the basic ontology elements that we consider.

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.

4.3.1. Illustration with running example

Figs. 3 and 4 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 Fig. 3) 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 SUBJECT EXPERT is confident with a change in DRAFT state, the element is passed to the TO BE APPROVED state and remains there until a VALIDATOR approves or rejects it.
- *Approved*: if a VALIDATOR 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.
- *To be deleted*: if a SUBJECT EXPERT 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 VALIDATOR will be able to definitively delete it.
- *Deleted*: this is the state assigned to any element when a VALIDATOR definitely deletes it after its state was TO BE DELETED. Additionally, this is the state assigned to any element when a SUBJECT EXPERT deletes it after its state was DRAFT. Note that this state is represented by the doubled circle in the figure.

The ontology state (see Fig. 4) 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 the DRAFT state and elements in the TO BE APPROVED state, then the state of the ontology is going to be DRAFT. Hence, ontology states are defined as follows:

- *Draft*: any change to an ontology in any state automatically sends it to DRAFT state.
- *To be approved*: when all ontology elements in an ontology version are in the TO BE APPROVED state (or DELETED), the ontology is automatically sent to the TO BE APPROVED state.
- *Approved*: when all ontology elements in an ontology version are in the APPROVED state (or DELETED), the ontology is automatically sent to the APPROVED state. Additionally, this is the default state of the initial version of a stable ontology.
- *Published*: only when the ontology is in the APPROVED state, it can be sent by a VALIDATOR to the 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.[16]

Note that during the workflow, actions are performed either:

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

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

- *Explicitly*: for example, validators explicitly approve or reject proposed changes. The action is recorded immediately when the user explicitly performs the action.

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 a workflow ontology. Having both models (ontology changes and workflow) formalized as ontologies facilitates 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 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.

4.4. 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 Fig. 5. In the figure, the elements above the horizontal line are generic elements of the workflow ontology that are independent of the collaborative development 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

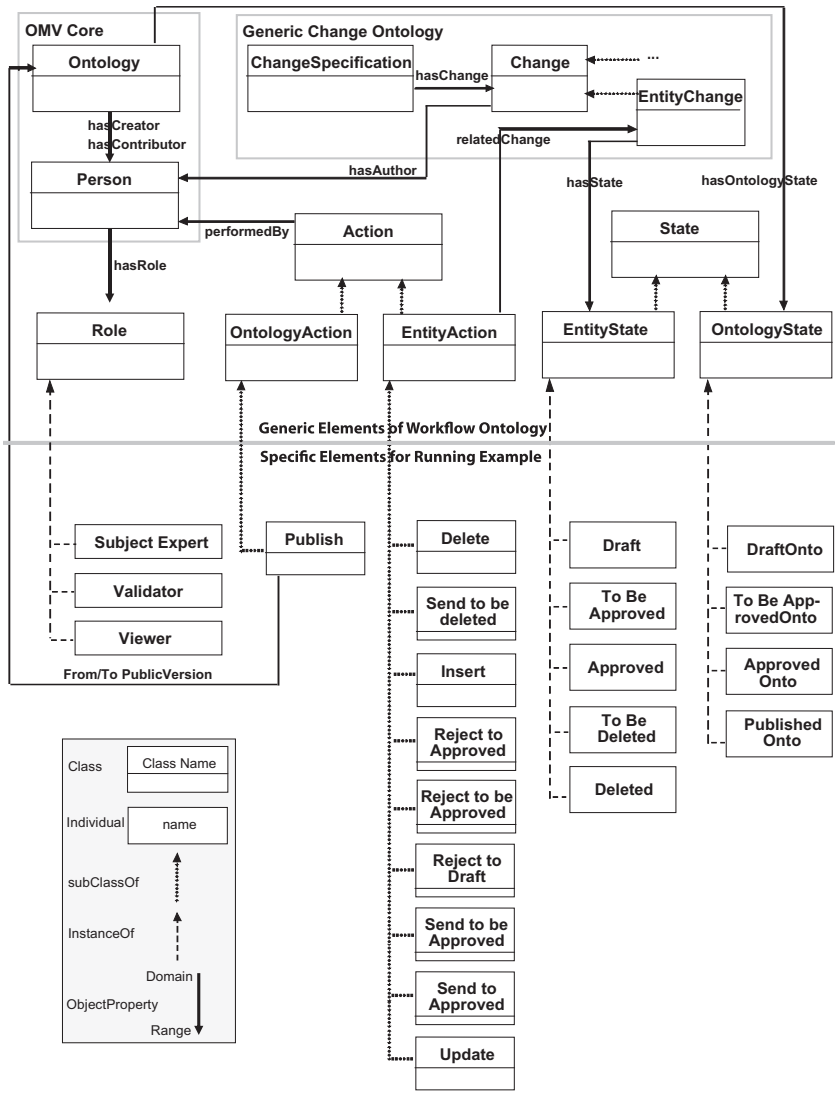


Fig. 5. Overview of the workflow ontology.

element has a particular state, we rely on the class EntityChange from the change ontology (described in Section 5.1) which is associated to a particular ontology element 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 at this point we are not considering composite changes in our workflow; they are treated as a sequence of changes at the entity level. Additionally, observe that the state is not assigned to the actual object (e.g., ontology element or ontology) but to the referring metadata entity (entity change or the ontology metadata) for two reasons: first, it is the actual referring entity that modifies the state of the object; second, all this information is managed by the ontology registry that stores all ontology related 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 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 are only specified, if necessary, between the explicit actions (at the ontology level) of the specific workflow and the class Ontology (see below).

4.4.1. Illustration with running example

As we can see in Fig. 5, 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 development process in our running example are not explicitly declared as part of the ontology. Those constraints are handled by the strategies for workflow management and enactment discussed in next section. The reasons for taking a procedural approach instead of a declarative one were mainly of pragmatic nature, as in our case the benefits of a declarative approach did not outweigh the additional development effort. Still, future work in this area should evaluate the practicability of a declarative approach.

4.5. Workflow management and enactment

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

- It requests users to identify themselves with their username and corresponding role.
- It enforces 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 are 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 supports transaction management capabilities. This means that if the performed action is rejected, all of its effects have to be undone.
- It creates and stores the appropriate individuals of the workflow ontology, if an action is successfully executed.
- It performs all consequences of an action if it is successfully executed. For instance, if a subject expert submits 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 enactment includes the provision of appropriate interfaces for the users to visualize and perform all the required actions. For instance, ontology editors (e.g., subject experts and validators) need to have different views of the ontologies developed collaboratively, depending on their role.

5. Change management in distributed environments

Our collaborative ontology development approach is based on the management of ontology changes in distributed environments. Our solution provides contributions to the representation of ontology changes (Section 5.1), their manipulation (Section 5.2) and propagation to distributed copies of the same ontology (Section 5.3).

5.1. Ontology change representation

A core element in our solution is the representation of changes. Although we found several approaches for the representation of changes (e.g., [17–19,21,22]), they have still some limitations (see Section 2). In particular, they are dependent on the underlying ontology model, and they only consider changes at the entity level (classes, properties, individuals). In [36] we presented our proposal for the representation of changes. In this paper we highlight only the most relevant parts: we propose a layered model for the representation of ontology changes that integrates many of the features of existing change ontologies (e.g., [17] and [18]). The core of this

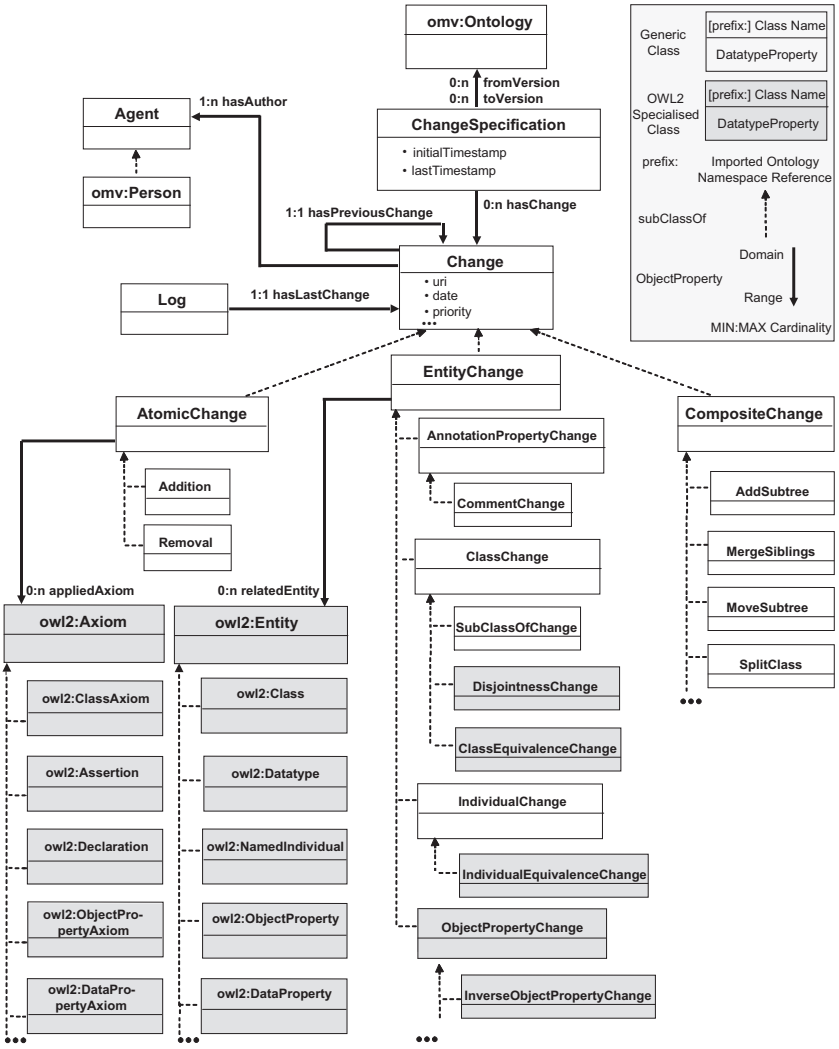


Fig. 6. Overview of the OWL 2 change ontology.

model consists of a generic change ontology, 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 [37]). The generic change ontology can be reused and specialized for specific ontology languages[17] (Fig. 6 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). The generic change ontology comprises three levels for the classification of changes:

- Atomic – the smallest and indivisible operation that can be performed in a specific ontology model.
- Entity – basic operations that can be performed over ontology elements.
- Composite – group of changes applied together that constitute a logical entity.

Besides the taxonomy of ontology changes, the generic change ontology 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.

The generic change ontology has been implemented as an extension of the OMV ontology metadata model, since we consider ontology changes as a special kind of ontology metadata.

5.2. Ontology change manipulation

The manipulation of changes includes supplementary methods and strategies that support the management of changes in distributed environments.

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, such as information gathering and data encoding, and to store them for future references.

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

[17] The generic change ontology and two specializations (for OWL 2 and RDFS) are available at http://omv.ontoware.org/.

steps: first, a change in an ontology from the ontology editor (Step 1), fires 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 corresponding individual(s). Finally, this change individual(s) is stored in an ontology registry for future processing and propagation (Step 5).

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 solution, such tasks are carried out by the ontology registry system.

5.2.2. Ontology version management

There are different definitions and understandings of what ontology versioning is (see [6]). For this work, we consider ontology versioning as a mechanism to keep track of ontology changes and to identify and maintain different variants of ontologies and their dependencies, with support to undo and 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. 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 (as described below). Furthermore, the information about the precise changes performed can be used to easily compute the difference between variants or to implement a multiple undo and redo mechanism.

Nevertheless, as it has been noted in the past (e.g., [38]), 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, our solution is based on the identification of ontologies that we proposed in the specification of OMV (see [39]) which consists of a tripartite identifier: the ontology URI, the ontology version (if present), and the ontology location. We also rely on OMV for representing the relationship between the different ontology versions (e.g., prior version and compatible with). Furthermore, 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. This phase normally follows a well-defined process for the coordination of changes (e.g., who can propose or approve changes, and 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). Particularly, in our solution, the first stable version (version 1) of an ontology (the ontology that satisfies all requirements) is considered the first approved ontology (which in some cases is also published in the Internet). 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 can receive many changes (by many ontology editors) without changing the version information until it becomes approved. Then, an authorized editor may want to publish this approved ontology version, and decide to either keep the version information or to specify a new one.

Note that in a distributed scenario, where ontology editors can work with local copies of the same ontology, the same new version can be created offline by two or more editors after applying some changes to the same base ontology. These copies will be synchronized once changes are propagated when they eventually go online, as described in Section 5.3.

5.2.3. Ontology change storage and maintenance

As we already mentioned before, the storage and maintenance of the change information is the responsibility of an ontology registry. The registry stores and manages ontology metadata information which includes information about changes. Hence, in our solution, the registry implements the following features for the management of changes (based on the models, methods and strategies described through this section):

- Stores the change history (log) for different ontologies, i.e., set of individuals of our change ontologies.
- Maintains the chronological order of changes. The set of individuals is maintained in a linked list, where each change individual is linked to the previous one.
- Supports different versions of ontologies at different locations.
- Provides access and advanced search capabilities to retrieve changes (based on the change representation model).
- Propagates changes to support, for instance, the synchronization of distributed copies of ontologies or the update of ontology related entities, e.g., metadata.

5.3. Ontology change propagation

Ontology change propagation is another task identified by most of the ontology evolution approaches and refers to the activity of updating all of the ontology dependent artifacts ([17]). As we discussed in Section 2, there are several limitations in this area. In general, existing approaches (e.g., [17,25,9,26,27]) consider only the propagation to related ontologies and individuals. Besides, for the propagation of changes to related ontologies, these 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, in this section we consider 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.

5.4. Change propagation to distributed copies of an ontology

One of the goals of propagating ontology changes in a distributed setting 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 capturing ontology changes.

Once the required changes are represented in a machine-understandable format, the system can propagate them to the distributed copies of the ontology. There are two possible approaches for the propagation: push or pull. The benefits and disadvantages of both approaches have already been analyzed (e.g., [17]). 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 URIs of the changes applied to each ontology (version). Note that, if the local node does not have 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 URIs of the changes 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 URIs of the changes applied to the ontology (e.g., O_i) in the remote node z and the corresponding set in the 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.[18]

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. Conflicts may be detected on a syntactic level (when the same entity is affected by changes from different users), or on a semantic level (when the combination of changes from different users results in a logical inconsistency) [40]. In case there are conflicting changes, a conflict resolution mechanism will be responsible for resolving them. In the simplest case, this mechanism removes the conflicting changes from the set and notifies the presence of the conflict (see below). Note that entity and composite changes must 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 of C_i (PC_{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.
- $PC_{C_i} = LC_{O_i}$, where LC_{O_i} is last change registered in the local log for O_i, i.e., the change applied to O_i with the newest timestamp.
- $PC_{C_i} \neq$ null and PC_{C_i} LC_{O_i}.

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} 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.

Fig. 7 illustrates the process described above. It 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 Fig. 7(a), before the synchronization, node x has five changes, while node z has seven changes. When x contacts 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 Fig. 7(b), after the synchronization, the log in x has 10 changes. This example

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

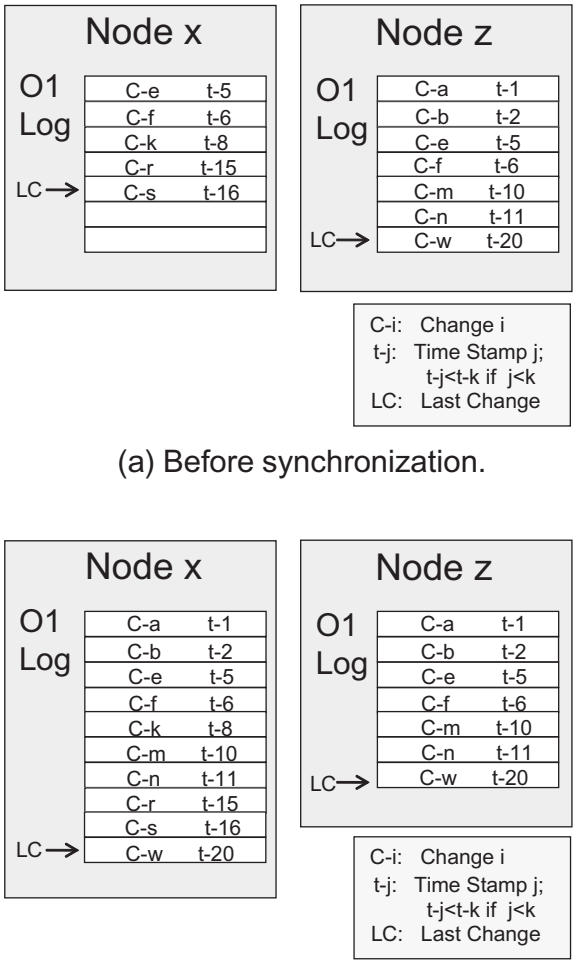


Fig. 7. Illustration of the synchronization process. Logs for ontology O1 in nodes *x* and *z* before and after synchronization, if changes are not conflicting.

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 paper. However, our strategy allows minimizing conflicts by running the synchronization process periodically in an automatic manner. Besides, the complete description of changes kept 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 [41] 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 must 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 will be queued for later processing.

Third, the periodicity of the synchronization process must 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 following additional considerations are needed: the local node has to process the unknown changes in order, starting with the first (oldest) unknown change. Additionally, each change has to be placed in the same location as in the remote log. That is, after the addition of C_i, the previous change of C_i will 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 or 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.

Notice that the strategies described above deal only with the propagation of changes in the domain ontology, that is, for individuals of the change ontology. The propagation of workflow ontology individuals is done by retrieving and processing the list of workflow actions in chronological order from the remote node, applying locally those actions that are not present in the local node. Applying those actions involves creating and storing the appropriate individuals of the workflow ontology and performing all the consequences of an action, such as changing the state of the associated changes, as described in Section 4.5.

6. Technological support for collaborative ontology development in distributed environments

The models, methods and strategies proposed in this work for collaborative ontology development and change management in distributed environments have been implemented within the NeOn Toolkit[19] by means of a set of plugins and extensions. A high level conceptual architectural diagram of the components involved is shown in Fig. 8.

6.1. Distributed registry

Ontologies are stored within a repository and their metadata is managed by the distributed registry Oyster[20] [42]. As an ontology registry, it provides services for storage, cataloguing, 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

[19] http://www.neon-toolkit.org/.
[20] http://ontoware.org/projects/oyster2/.

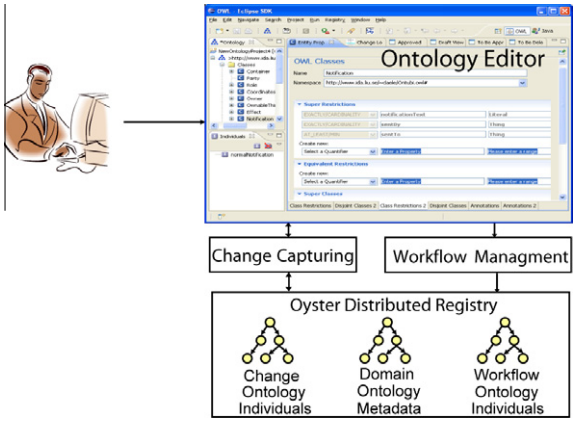


Fig. 8. Conceptual architecture for the collaborative ontology development support.

semantic searches of the registered objects) and providing services to support the management and evolution of ontologies in distributed environments.

The metadata includes information about ontologies and users (represented using OMV core), the changes to the ontology (represented using the generic change ontology and 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. 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 (i) gets the last registered change (using the "Log" class); (ii) adds it as the previous change of the current one; (iii) 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 5.4, thus allowing the notification of new changes to ontology editors. Oyster implements the synchronization process as follows: nodes periodically 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.

6.2. 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.

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 previous subsection). Finally, this plugin extends the NeOn Toolkit with a view to display the history of ontology changes.

6.3. Workflow management and enactment component

In our implementation, this component implements the strategies described in Section 4.5. It (i) takes care of enforcing the constraints imposed by the collaborative editorial workflow, (ii) creates the appropriate action individuals of the workflow

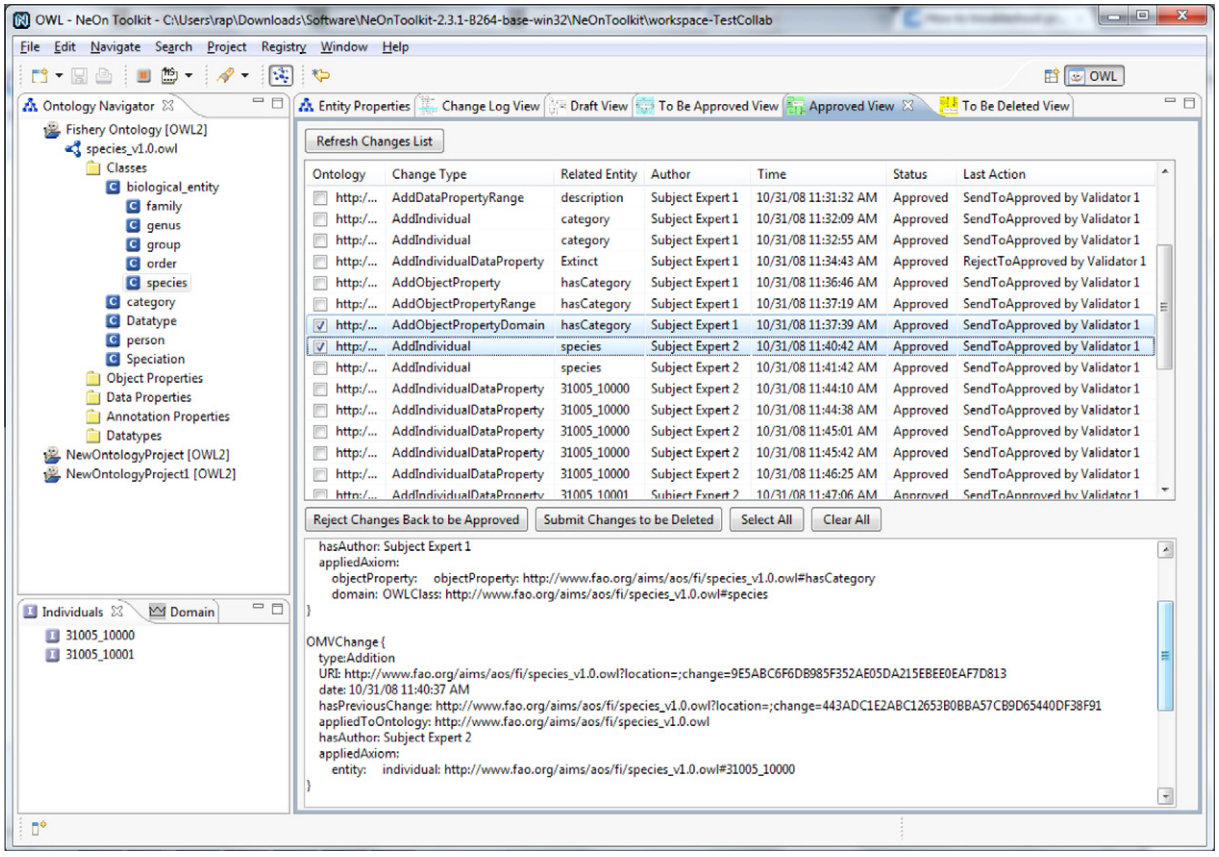


Fig. 9. Draft view in the NeOn Toolkit. The view (right pane of the window) illustrates the collaboration of three ontology editors (2 subject experts and 1 validator). The top pane of the view shows the list of changes in the log, whereas the bottom pane shows the details of the selected changes.

Table 1
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

ontology and (iii) registers them into the distributed registry ensuring a transactional support.

6.4. Ontology editing and visualization components

To support the workflow activities we rely on the NeOn Toolkit, which comes with an ontology editor that allows the editing of ontology elements. Additionally, 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 4.3) the applicable workflow actions (*approve, reject*, etc.), depending on their role. There are four views[21]: Draft view, Approved view, To Be Approved view and To Be deleted View. Each view is organized in two vertical panes. The top pane displays the list of changes in the log along with their most relevant information, such as the URI of the modified ontology, type of change, related entity, author, date and time, the status and last action (according to the workflow) associated to the change. The bottom pane shows all the details of the changes selected in the top pane, that is, a serialization of the change ontology individuals. Additionally, each view provides different buttons that allow ontology editors to perform the applicable workflow actions (e.g., send to be approved and reject to draft). Fig. 9 illustrates the draft view.

For a detailed description on how our framework can be configured to support the scenarios identified in Section 4.1, we refer the reader to [6].

7. Evaluation experiment

We have conducted an experiment in a real life scenario at the Food and Agricultural Organization (FAO) 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. Because of space constraints we present here only an overview of the experiment conducted, and in Section 8 we discuss some of the relevant results of the performed evaluation. For a complete description of the evaluations conducted we refer the reader to [6].

Following the phases considered in most software experimentation approaches [43–45], the experiment was performed in three consecutive phases. First, during the plan phase, the definition and design of the experiment was described, including the motivation, constraints, goals, beneficiaries and the experiment subject as well as the relevant characteristics, metrics and criteria, the data collection and analysis processes. Then, during the experiment phase, the execution of the experiment was described, including the particular configuration of the collaborative infrastructure and the software used as well as the description of the users involved, the activities they had to carry out and the time needed for the overall experiment. Finally, during the analysis phase, the experiment results were analyzed.

Table 1 summarizes the characteristics evaluated during the experiment along with the applied metrics and criteria. Note that for measuring the user satisfaction, we used as basis the Software Usability Measurement Inventory[22] (SUMI) which is a rigorously tested and proven method of measuring software quality from the end users point of view [46].

In summary, a team of three representative ontology editors, each playing one of two roles (subject expert, validator), worked collaboratively on the maintenance of the species ontology schema, one of the most frequently used fishery ontologies at FAO.[23] Following FAO priorities and time constraints, the infrastructure was configured as the scenario A described in Section 4.1. In particular, the configuration consisted of three clients, each running the NeOn Toolkit extended with the registry (Oyster), change management and collaboration plugins, and one server running Oyster (in server mode) and the NeOn collaboration server. The latter is software component that allows distributed users (running the NeOn toolkit) to remotely access, browse and edit concurrently a centralized copy of an ontology.

After a brief introduction to the system (30 min), editors followed a detailed and personalized guide of the tasks they had to perform. In a nutshell, each subject expert (SE) had to perform six main tasks while the validator (V) had to perform three 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 that were defined in collaboration with 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).

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

[22] http://sumi.ucc.ie/index.html.

[23] http://aims.fao.org/website/Ontologies-/sub2#species.

- Then, the validator was requested to analyze the changes performed and to approve some of them and reject the rest (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).

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 consisting of 60 questions (50 of the standard SUMI questionnaire and 10 specific for the collaborative ontology development).[24]

8. Evaluation summary and discussion

As a general conclusion we can say that the results of the experiment were positive. In particular, they 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. Ontology editors completed 64 out of the 68 specific tasks (94%) that were grouped in the 15 main tasks introduced in Section 7. However, the log showed that some editors performed other additional tasks (i.e., they added 13 additional changes) that were not in the guide. Additionally, there was a 90% rate of changes logged correctly (3 out of the 30 changes created from the guide were logged twice due to a limitation in the NeOn collaboration server – see [6]). Moreover, we obtained a 100% rate of success of the workflow management and enactment functionality, that is, for each of the 73 actions completed from the guide (SE1 performed 17 implicit actions and 20 explicit ones, SE2 performed 13 implicit actions and 16 explicit ones and V1 performed 7 explicit actions), the workflow action was correctly created, represented and registered and it had the expected consequences. Finally, 90% (18 out of 20) of the features of the infrastructure were tested during the experiment (as a result of the scenario setup). These numbers provide positive evidence with respect to the effectiveness of the infrastructure.
- The efficiency of the system was in general satisfactory. A 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 in [6]), providing positive evidence with respect to the efficiency of the infrastructure. Regarding the frequency of help use, users asked regularly for assistance during the experiment. This is reasonable 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 frequently. Moreover, after using the system for some minutes, users learned how to use most of the system features.

Moreover, the results of the survey to measure user satisfaction showed that users were in general satisfied with the infrastructure and generally agreed on its usefulness and correctness. However, even though the results of the experiment provide an indication of the real value and practical usability of the models, methods and strategies proposed in this work, additional experiments with more users will be needed to draw full conclusions.

Ontology editors 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. For example, some textual answers regarding the best features are: *"Seeing changes of others"*, *"A unified view of everything happening in the workflow"*, *"Great capability and real time update"*.

The overall results for each of the five SUMI dimensions have a similar pattern. In each case, only around one fourth (7 out of 30) of the total answers were negative, another fourth (approximately 8 out 30) were undecided and at least half of the answers (15 out of 30) were always positive.

Nevertheless, during the experiment and as part of the analysis of the results, we also learned important lessons. 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). Related to this issue is that of the granularity of changes: the change log captures the changes as performed in the ontology editor, where a set of multiple changes may reflect a change that – on the level of the user intent – should be a single composite change. Such aggregation or grouping into composite changes is currently not possible in our implementation. 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. From these (and other) observations we got some recommendations on how to improve our infrastructure, specifically at the GUI level. First we should improve our views with additional features such as sorting, grouping, filtering, and aggregating. Second, we should also add new 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. Finally, we should provide a tighter link between the ontology navigator and the information displayed in our views.

9. Conclusions

In this work we have presented our solution to support collaborative ontology development based on the management of ontology changes in distributed environments.

For our contribution to 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, which 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

[24] Online survey at http://www.oeg-upm.net/files/UsabilitySurvey/survey.htm. Guides, setup files and results of the experiment available at http://www.oeg-upm.net/files/material/experimentData.zip.

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 was implemented in a workflow ontology by reusing knowledge modeled by OMV and our change ontology.

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 ontology development process. We identify the set of tasks that have to be carried out, including those to enforce the constraints specified by the collaborative development process, and propose strategies to deal with them.

Our contributions also include an integrated infrastructure implemented within the NeOn Toolkit by means of a set of plugins and extensions that support collaborative ontology development. 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 development process that coordinates the proposal of ontology changes and a completely distributed control for the management of changes.

With respect to the management of changes, we have focused on two main activities: representation and propagation.

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, which can be reused and specialized for different ontology languages. Our model considers ontology changes at the lowest level of granularity (the real atomic operations) instead of at the entity level like other existing approaches. Since information about changes in ontologies is a type of ontology metadata, we implemented our change ontology as an 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 includes 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 undo and redo operations, or 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 the propagation of changes to distributed copies of the same ontology. The benefit of our solution is that it addresses previously 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.

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.

Our contributions also include the implementation of the distributed ontology registry Oyster for the management of ontology changes in distributed environments.

10. Future work

The most important topics in the context of collaborative ontology development in distributed environments that can extend our work are:

Configurable editorial workflows. An important feature to elaborate in the future is to extend our approach for the formalization of the collaborative development process to support any kind of collaborative editorial workflow (configurable at 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.

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.

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.

Acknowledgments

Research reported in this paper was partially supported by the EU in the IST project NeOn (IST-2006-027595), http://www.neon-project.org/. We are indebted to FAO for their help with the evaluation and for their insightful feedback on the tool.

References

[1] M. Fernández-López, A. Gómez-Pérez, J. Pazos-Sierra, A. Pazos-Sierra, Building a chemical ontology using methontology and the ontology design environment, IEEE Intelligent Systems 14 (1) (1999) 37–46.
[2] S. Staab, R. Studer, H.-P. Schnurr, Y. Sure, Knowledge processes and ontologies, IEEE Intelligent Systems 16 (1) (2001) 26–34.
[3] S. Pinto, S. Staab, Y. Sure, 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, Heraklion, Crete, Greece, 2004, pp. 16–30.
[4] A. de Moor, P. D. Leenheer, R. Meersman, DOGMA-MESS: A Meaning Evolution Support System fornm Interorganizational Ontology Engineering, in: Proceedings of the International Conference on Conceptual Structures, (ICCS 2006), Aalborg, Denmark, Springer, 2006.
[5] O. Muñoz-García, A. Gómez-Pérez, M. Iglesias-Sucasas, S. Kim, A Workflow for the Networked Ontologies Lifecycle. A Case Study in FAO of the UN, in: Proceedings of the CAEPIA-TTIA 2007, Springer, Spain, 2007.
[6] R. Palma, Ontology Metadata Management in Distributed Environments, Ph.D. Thesis, Universidad Politécnica de Madrid, Spain (December 2009). Available at: <http://www.oeg-upm.net/files/material/dissertation-RAP- Digital.pdf>.
[7] C. Tempich, Ontology Engineering and Routing in Distributed Knowledge Management Applications, Ph.D. Thesis, University of Karlsruhe (TH), Germany (2006).
[8] V. Komulainen, A. Valo, 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), 2005.
[9] E. Sunagawa, K. Kozaki, Y. Kitamura, R. Mizoguchi, An environment for distributed ontology development based on dependency management, in: D. Fensel, K. P. Sycara, J. Mylopoulos (Eds.), International Semantic Web Conference, Vol. 2870 of Lecture Notes in Computer Science, Springer, 2003, pp. 453–468.

[10] J. Bao, V. Honavar, Collaborative Ontology Building with Wiki@nt – A Multi-agent based Ontology Building Environment, in: Proceedings of Third International Workshop on Evaluation of Ontology-based Tools, located at the Third International Semantic Web Conference ISWC 2004, Hiroshima, Japan, 2004.
[11] S. Auer, S. Dietzold, T. Riechert, OntoWiki – A Tool for Social, Semantic Collaboration, in: The Semantic Web – ISWC 2006, Fifth International Semantic Web Conference, ISWC 2006, Springer, 2006, pp. 736–749.
[12] N. F. Noy, A. Chugh, W. Liu, M. A. Musen, A framework for ontology evolution in collaborative environments, in: I.F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, L. Aroyo (Eds.), International Semantic Web Conference, Vol. 4273 of Lecture Notes in Computer Science, Springer, 2006, pp. 544–558.
[13] T. Tudorache, N. Noy, Collaborative Protégé, in: Workshop on Social and Collaborative Construction of Structured Knowledge (CKC 2007) at WWW 2007, Banff, Canada, 2007.
[14] A. Sebastian, N. F. Noy, T. Tudorache, M. A. Musen, A generic ontology for collaborative ontology-development workflows, in: A. Gangemi, J. Euzenat (Eds.), Proceedings of the 16th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2008), Vol. 5268 of Lecture Notes in Computer Science, Springer, 2008, pp. 318–328.
[15] T. Tudorache, N. Noy, S. Tu, M. Musen, Supporting Collaborative Ontology Development in Protégé, in: International Semantic Web Conference, 2008.
[16] A. Sebastian, T. Tudorache, N.F. Noy, M.A. Musen, Customizable Workflow Support for Collaborative Ontology Development, in: Fourth International Workshop on Semantic Web Enabled Software Engineering (SWESE) at ISWC 2008, 2008.
[17] L. Stojanovic, Methods and Tools for Ontology Evolution, Ph.D. Thesis, University of Karlsruhe (TH), Germany, August 2004.
[18] M. Klein, Change Management for Distributed Ontologies, Ph.D. Thesis, Vrije Universiteit, Amsterdam, 2004.
[19] M. Klein, D. Fensel, A. Kiryakov, N.F. Noy, H. Stuckenschmidt, Versioning of Distributed Ontologies, Tech. Rep., Vrije Universiteit Amsterdam, December 2002.
[20] Y. Liang, Mini-Thesis: Enabling Active Ontology Change Management within Semantic Web-based Applications, Ph.D. Thesis, University of Southampton, 2006.
[21] M. Klein, N. Noy, A Component-based Framework for Ontology Evolution, in: Proceedings of the IJCAI'03 Workshop: Ontologies and Distributed Systems, Acapulco, Mexico, 2003.
[22] N.F. Noy, M.C.A. Klein, Tracking complex changes during ontology evolution, in: ISWC-2003 Poster Proceedings, Sanibel Island, Florida, 2003.
[23] P. Haase, Y. Sure, D. Vrandecic, D3.1.1 Ontology Management and Evolution: Survey, Methods and Prototypes, Tech. Rep. 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 >.
[24] G. Flouris, D. Plexousakis, Handling ontology change: survey and proposal for a future research direction, Tech. Rep. FORTH-ICS/TR-362, Institute of Computer Science, FORTH, September 2005. Available at: http://www.ics.forth.gr/isl/publications/paperlink/fgeo_TR362.pdf>.
[25] D. Oliver, Change Management and Synchronization of Local and Shared Versions of a Controlled Vocabulary, Ph.D. Thesis, Stanford University, 2000. Available at: <http://www.citeseer.ist.psu.edu/oliver00change.html>.
[26] L. Stojanovic, N. Stojanovic, S. Handschuh, Evolution of the Metadata in the Ontology-based Knowledge Management Systems, in: Proceedings of the First German Workshop on Experience Management, GI, 2002, pp. 65–77.
[27] D. Maynard, W. Peters, M. Sabou, M. d'Aquin, Change management for metadata evolution, in: International Workshop on Ontology Dynamics (IWOD) ESWC 2007 Workshop – 7 June – Innsbruck, 2007–06.
[28] Y. Liang, H. Alani, D. Dupplaw, N. Shadbolt, An Approach to Cope with Ontology Changes for Ontology-based Applications, in: Second Advanced Knowledge Technologies DTA Symposium, Aberdeen, 2006.
[29] 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.
[30] N. Noy, S. Kunnatur, M. Klein, M. Musen, Tracking Changes During Ontology Evolution, in: International Semantic Web Conference, 2004.
[31] D. Rogozan, G. Paquette, Managing Ontology Changes on the Semantic Web, in: WI '05: Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence, IEEE Computer Society, Washington, DC, USA, 2005, pp. 430–433.
[32] D. Ognyanov, 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, Springer-Verlag, London, UK, 2002, pp. 373–378.
[33] A. Gangemi, J. Lehmann, V. Presutti, M. Nissim, 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.
[34] J. Hartmann, R. Palma, Y. Sure, P. Haase, M. del Carmen Suárez-Figueroa, OMV – Ontology Metadata ocabulary, in: C. Welty (Ed.), ISWC 2005 – In Ontology Patterns for the Semantic Web, 2005.
[35] R. Palma, P. Haase, O. Corcho, A. Gómez-Pérez, Q. Ji, An Editorial Workflow Approach For Collaborative Ontology Development, in: ASWC '08: Proceedings of the Third Asian Semantic Web Conference on The Semantic Web, Springer-Verlag, Berlin, Heidelberg.
[36] R. Palma, P. Haase, O. Corcho, A. Gómez-Pérez, Change Representation for OWL 2 Ontologies, in: Fifth International Workshop OWL: Experiences and Directions (OWLED 2009), 2009.
[37] T.R. Gruber, A translation approach to portable ontology specifications, Knowledge Acquisition 5 (2) (1993) 199–220.
[38] M. Klein, D. Fensel, Ontology Versioning for the Semantic Web, in: Proceedings of the International Semantic Web Working Symposium (SWWS'01), Stanford University, California, USA, 2001.
[39] R. Palma, J. Hartmann, P. Haase, OMV – Ontology Metadata Vocabulary for the Semantic Web, Tech. Rep., Universidad Politécnica de Madrid, University of Karlsruhe, version 2.4, 2008. Available at: <http://omv.ontoware.org/>.
[40] P. Haase, L. Stojanovic, Consistent evolution of OWL ontologies, in: A. Gómez-Pérez, J. Euzenat (Eds.), ESWC, Vol. 3532 of Lecture Notes in Computer Science, Springer, 2005, pp. 182–197.
[41] R. Palma, P. Haase, Y. Wang, M. d'Aquin, D1.3.1 Propagation Models and Strategies, Tech. Rep. D1.3.1, UPM; NeOn Deliverable, November 2007, Available at: <http://www.neon-project.org/>.
[42] R. Palma, P. Haase, Oyster – Sharing and Re-using Ontologies in a Peer-to-Peer Community, in: International Semantic Web Conference, 2005, pp. 1059–1062.
[43] V.R. Basili, R.W. Selby, D.H. Hutchens, Experimentation in software engineering, IEEE Transactions on Software Engineering 12 (7) (1986) 733–743.
[44] S.L. Pfleeger, Experimental design and analysis in software engineering – Part 2: how to set up and experiment, SIGSOFT Software Engineering Notes 20 (1) (1995) 22–26.
[45] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, El.J. Rosenberg, Preliminary guidelines for empirical research in software engineering, IEEE Transactions on Software Engineering 28 (8) (2002) 721–734.
[46] E.P. van Veenendaal, Questionnaire Based Usability Testing, in: Proceedings of the European Software Quality Week, Brussels, 1998.