


An Editorial Workflow Approach For Collaborative Ontology Development

View metadata, citation and similar papers at core.ac.uk

brought to you by  CORE

provided by Servicio de Coordinación de Bibliotecas de la Universidad...

¹Ontology Engineering Group, Laboratorio de Inteligencia Artificial
Facultad de Informática, Universidad Politécnica de Madrid, Spain
{rpalma, ocorcho, asun}@fi.upm.es

²Institute AIFB, University of Karlsruhe, Germany
{pha, qiji}@aifb.uni-karlsruhe.de

Abstract. The widespread use of ontologies in the last years has raised new challenges for their development and maintenance. Ontology development has transformed from a process normally performed by one ontology engineer into a process performed collaboratively by a team of ontology engineers, who may be geographically distributed and play different roles. For example, editors may propose changes, while authoritative users approve or reject them following a well defined process. This process, however, has only been partially addressed by existing ontology development methods, methodologies, and tool support. Furthermore, in a distributed environment where ontology editors may be working on local copies of the same ontology, strategies should be in place to ensure that changes in one copy are reflected in all of them. In this paper, we propose a workflow-based model for the collaborative development of ontologies in distributed environments and describe the components required to support them. We illustrate our model with a test case in the fishery domain from the United Nations Food and Agriculture Organisation (FAO).

1 Introduction

The growing use and application of ontologies in the last years has lead to an increased interest of researchers in the development of ontologies, either from scratch or by reusing existing ones. Ontology development and maintenance activities are addressed by many different methodologies (e.g. Methontology, On-To-Knowledge, DILIGENT, etc.). However, most of them only consider the development of ontologies by single users or a small group of ontology engineers placed in the same location. More important is that even though they address the methodological aspects, in general they focus less on the process followed by organisations to coordinate the collaborative ontology development. In practice ontologies may be distributed, and a whole team of ontology engineers with different roles may collaborate in the development and maintenance, usually following a well defined process. Examples of such collaborative development processes can be found in international institutions like the United Nations Food and Agriculture Organisation (FAO), who are developing and maintaining large ontologies in the fishery domain [8]. Other similar examples are those of the Gene Ontology (GO)

project¹, which addresses the need for consistent descriptions of gene products in different databases, the caGrid project², which aims at providing a virtual informatics infrastructure that connects data, research tools, scientists, and organizations, etc.

Consequently, in this collaborative organisational setting, existing approaches are not enough to support all ontology development and maintenance needs. Furthermore, although recently some proposals and tools have been designed specifically to support collaborative ontology development (e.g. client-server mode in Protégé along with the PROMPT and change-management plugins), they generally only address parts of the overall problem (see section 4). Most of the existing advanced ontology tools (e.g. Protégé core system, SWOOP, etc.) support only the single-user scenario, where there is just one user involved in the development and later modification of the ontologies. With such tools, a typical scenario of collaborative ontology development would look as follows: An editor changes an ontology using his ontology editor system and then sends (e.g. using email or uploading it to an ontology repository) his locally changed ontology to other users (i.e. to add more changes using their own Protégé system, or review current changes). Even in the scenario where all users are editing the same ontology stored in a central server (e.g. using client-server mode in Protégé), the coordination of the actions of the editors (e.g. when editors want their changes to be reviewed or what kind of actions they can perform) is not yet fully supported.

As we can see from the previous discussion, in this type of collaborative scenario, change management is central. Hence, we need appropriate procedures (and corresponding infrastructure) to control and support the management of ontology changes. This procedure can be modelled as a collaborative workflow, which according to [2], is a special case of epistemic workflow characterized by the ultimate goal of designing networked ontologies and by specific relations among designers, ontology elements, and collaborative tasks. The need for such workflows has also been acknowledged in the past by other related works (e.g. [17]). An example of such workflow is that followed by the FAO (described in [8]), which we take as a use case in our work, in order to derive a generic set of required activities to support it.

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

In this paper we present our approach for the **management of collaborative ontology development in a distributed scenario by means of an editorial workflow**. We analyse the collaborative development process using as illustrating scenario the case study at FAO and derive a set of functional requirements to support the process. We then introduce our proposal to support the collaborative ontology development where we address the identified problems. In particular, we propose a formal model for the representation of the workflow and describe the relationship with other models and

¹ <http://www.geneontology.org/>

² <http://www.cagrid.org/>

methods required in the management of ontology changes in distributed environments. Our contribution also includes the implementation of the proposed approach. The remainder of this paper is organised as follows: In section 2 we analyse the collaborative workflow scenario at FAO and derive a set of requirements based on the editorial workflow. In section 3 we introduce our approach for the collaborative ontology development based on the requirements derived in the previous section and present our implementation that provides the technological support to the presented models and methods. Section 4 provides a brief summary of related approaches to collaborative ontology development. Finally we conclude with a discussion in section 5.

2 Requirements for Collaborative Ontology Development

In this section we present the most relevant requirements to support the collaborative ontology development based on the analysis of the process (i.e. workflow) typically followed by organisations in the development and maintenance of ontologies³. For our analysis we considered existing processes for collaborative ontology development, also taking similar works in the state of the art into account (e.g. [10]). We use a case study of the NeOn project⁴ for illustration: Specifically, we consider the editorial workflow of the fisheries ontologies lifecycle from FAO [8].

2.1 Overview of the Fisheries Ontologies Lifecycle

Within this case study, NeOn partners 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, the goal of the case study is to implement an ontology-based Fishery Stock Depletion Assessment System (FSDAS) as well as an application to manage the fishery ontologies and their lifecycle.

The full lifecycle of the fisheries ontologies is introduced in [9], we here focus on the ontology engineering phase: In a nutshell, there are several actors involved in the engineering phase of the fishery ontology lifecycle, including experts in ontology modeling, that are in charge of defining the original skeleton of the ontology, ontology editors, that are in charge of the everyday editing and maintenance of the ontologies, and subject matter experts who know about the domain to be modeled. Finally, validators are subject experts who can move a change to production status for external availability. Ontology development follows a well defined collaborative workflow, which needs to be supported in the engineering environment. The editorial workflow allows ontology editors to consult, validate and modify the ontology keeping track of all changes in a controlled manner. Finally, once editors in charge of validation consider the ontology final, they are authorized to release it and make it available to end users and systems.

2.2 Functional Requirements

The functional requirements for the collaborative editorial workflow specify the specific functionality of the workflow, including the specification of the workflow behavior.

³ In the remainder of this paper we refer to this process as the collaborative editorial workflow.

⁴ <http://www.neon-project.org/>

Some of the requirements that we introduce in the following are similar to the ones that have been already identified in the past (see the analysis of [10] in section 4). However, in our work, we further identified additional requirements to support the process followed typically by many organisations to coordinate the collaborative ontology development. When appropriate, we illustrate the requirement using FAO scenario.

Lifecycle Requirements. A collaborative editorial workflow should implement the necessary mechanisms to allow ontology editors to: **consult**, **modify** and **validate** ontologies. In some cases, ontologies may also need to be **published** on the internet once they are fully validated. Furthermore, the process should ensure that the aforementioned activities are carried out in a controlled and coherent manner. Hence, the editorial workflow is responsible for the coordination of who (depending on the user *role*) can do what (i.e. what kind of *actions*) and when (depending on the *status* of the ontology elements – classes, properties and individuals – and the role of the user).

The activities of the editorial workflow are being done by users that are ontology editors in charge of the everyday editing and maintenance work of the ontologies. Each user is assigned a specific role (which has associated permissions) by the organisation based on his expertise and responsibilities. Depending on the user permissions, he can be in charge of developing specific fragments of ontologies, revising work done by others, or developing new versions of ontologies. Ontology editors know about the ontologies domain, but usually know little or nothing about ontology software or design issues. For instance, one approach for assigning the user role can be driven by the module of the ontology the user is responsible for (e.g. [6]). As another example, in FAO, an ontology editor can be assigned one of the following roles:

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

To enforce permissions, it is required that (i) the system supports the different user roles and (ii) users identify themselves to the system before using it. Furthermore, to control when the ontology editors are allowed to work with an ontology element, in addition to the user roles, every ontology element is required to have a *status*. Ontology editors can change the status depending on their role. For instance, the possible status ontology elements can have in FAO include: *Draft* for the proposed additions or updates, *To Be Approved* for the proposed changes that are ready to be reviewed by a validator, *Approved* for the accepted changes, *To Be Deleted* for the proposed removals and *Published* for the changes released to the internet.

Workflow Activities. Activities required to support the editorial workflow include the operations (or possible *actions*) the ontology editors are allowed to perform depending on their roles and the status of the ontology elements.

Edit ontology element

Insert an ontology element. This operation triggers the start of the editorial workflow.

Update an ontology element. Editors can update ontology elements. Depending on their role and the status of the element, this operation could trigger also the start of the editorial workflow. For instance, in our illustrative scenario, a SE can only update elements in "Draft" or "Approved" status. In both cases the status of the element is automatically reset by the system to "Draft", and the element will need to pass through the whole workflow again.

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

Change status of ontology element. While inserting, updating and deleting elements, their status is automatically changed by the system. There are other cases where a specific action from editors is required to move an element from one status to another and make the editorial workflow to function (e.g. SE's need to explicitly send elements in "Draft" status to the "To Be Approved" status).

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

Visualization Requirements

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

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

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

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

Change Management

Representation of changes. A main requirement is the explicit representation of the changes that editors are able to perform to ontologies. The representation should ensure the accessibility and interoperability with other components (e.g. workflow, ontology metadata, etc.) and the maintenance of the chronological order of the changes to support e.g. undo/rollback operations or reconstruction of performed operations (e.g. when synchronizing/propagating changes). Additionally, to facilitate the previous tasks and provide an efficient link between what the user sees (e.g. ontology elements) and what the system manage internally (e.g. axioms), the representation should provide a flexible classification of changes that considers the actual "atomic" operations that can be performed over ontologies in addition to operations at the element level (e.g. to support the different status that each ontology element can have during the editorial workflow) or the complex operations that have been considered in the past (see section 4). Information about changes should include e.g. the operation performed, the time of the operation, the user, the element associated, the previous change and the description.

Capture ontology changes. The system should automatically log ontology changes.

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

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

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

3 A Workflow-Based Collaborative Ontology Development Approach

In this section we present our solution to support the collaborative ontology development and describe how it tackles the aforementioned requirements. We first present the conceptual models⁵ that provide the foundations to represent the required information in our solution and then we present the implementation support.

3.1 Conceptual Models

Change Representation. A core element in our approach is the representation of changes (c.f. *change management requirement*). In [13] we presented our proposal for the representation of changes which integrates many of the features of the existing approaches (e.g. [15], [5]) in a consistent layered manner. In this paper we highlight only the most relevant parts of our representation of changes: We refine and extend existing work and propose a layered approach for the representation of changes that consists of a generic ontology that models generic operations in a taxonomy of changes that are expected to be supported by any ontology language and that can be specialized for specific ontology languages (e.g. OWL) while still providing a common, independent model for the representation of ontology changes. It comprises three levels for the classification of changes: Atomic (i.e. the smallest and indivisible operation that can be performed in a specific ontology model), Entity (i.e. basic operations that can be performed over ontology elements usually from an ontology editor) and Composite (i.e. group of changes applied together that constitute a logical entity). It also provides the link to capture the argumentation of changes and it relies and uses some of the knowledge defined in our early work, the Ontology Metadata Vocabulary (OMV) [3] to refer to ontologies and users. OMV is a metadata schema that captures relevant information about ontologies such as provenance, availability, statistics, etc. Besides the main class

⁵ Our conceptual models are available in OWL at <http://omv.ontoware.org>

Ontology, OMV also models additional classes and properties required to support the reuse of ontologies, such as Organisation, Person, LicenseModel, OntologyLanguage and OntologyTask among others. Our change ontology has been implemented as an OMV extension because it models specific ontology metadata (i.e. ontology changes).

Furthermore, the 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 (*versioning requirement*). This is not a trivial issue: even though ontologies are in general identified by an URI, in practice it is not enough to identify a particular ontology version (i.e. different versions of the same ontology have the same URI). Hence, the management of ontology versions requires a clear definition of the ontology identification. In our solution, we rely on the identification of ontologies that we presented in [3], which consists of a tripartite identifier: the ontology URI, the ontology version (if present), and the ontology location.

Finally, to keep track of the actual sequence of changes (i.e. the order in which changes were performed), our ontology relies on two elements: each change is linked to its predecessor via the "hasPreviousChange" object property and a "Log" class provides the pointer to the last change in the ontology history.

Workflow Model. Based on the analysis presented in section 2 we found that some of the possible actions and states in the editorial workflow apply at different levels of abstraction. Therefore our solution considers the editorial workflow at two levels: ontology level and ontology element level. Although the workflows can be used independently of the underlying ontology model, the specific set of ontology elements depend on the ontology model. In our approach we are mainly considering the OWL ontology model, in which an OWL ontology consists of a set of axioms and facts⁶. Facts and axioms can relate to classes, properties or individuals, and hence that is the set of ontology elements we are considering.

As previously discussed, the workflow details (e.g. the specific roles, actions, etc.) depend on the organisation setting. To exemplify, in the rest of this section we discuss our solution for the particular scenario in FAO. Figures 1 and 2 show the two different workflow levels (i.e. element and ontology level). States are denoted by rectangles and actions by arrows. The information in parenthesis 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 (see Figure 1) that can be assigned to ontology elements are:

- *Draft*: This is the status assigned to any element when it passes first into the editorial workflow, or when it was approved and then updated by a subject expert.
- *To be approved*: Once a "SE" is confident with a change in draft status the element is passed to the "To Be Approved" status, and remains there until a "V" approves/rejects it.
- *Approved*: If a "V" approves a change in an element in the "To Be Approved" status, it passes to the "Approved" status. Additionally, this is the default state for every element of the initial version of a stable ontology.

⁶ In our current implementation we support the upcoming OWL 2 language. See <http://www.w3.org/TR/owl2-syntax/>

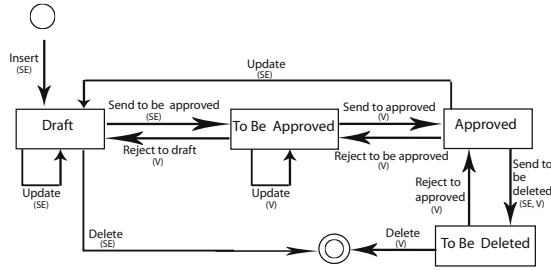


Fig. 1. Editorial workflow at the element level

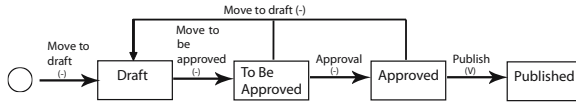


Fig. 2. Editorial workflow at the ontology level

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

The ontology has a state (see Figure 2) that is automatically assigned by the system (denoted with "-" in Figure 2), except from the "published" state as described below:

- *Draft*: Any change to an ontology in any state automatically sends it into draft state.
- *To be approved*: When all changes to an ontology version are in "To Be Approved" state (or deleted) the ontology is automatically send to "To Be Approved" state.
- *Approved*: When all changes to an ontology version are in "Approved" state (or deleted) the ontology is automatically send to "Approved" state. Additionally, this is the default state of the initial version of a stable ontology.
- *Published*: Only when the ontology is in "Approved" state, it can be sent by a validator to "Published" state.

As described in section 2.1, the editorial 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"⁷.

Note that during the editorial workflow, actions are performed either implicitly or explicitly. For instance, when a user updates an element he does not explicitly perform an update action. In this case the action has to be captured from the user interface and recorded when the ontology is saved. In contrast, Validators explicitly approve/reject proposed changes and the action is recorded immediately when performed.

Similarly to our change ontology, we decided to model the workflow elements (i.e. roles, status, actions) using an (OWL-Lite) ontology (i.e. a workflow ontology) that allows the formal and explicit representation of knowledge in a machine-understandable

⁷ In a different scenario, the workflow could start with an empty ontology (without elements), which we could assume that will be by default in "Approved" state.

format. Furthermore, having both models (i.e. ontology changes and workflow) formalized as ontologies will facilitate the representation of the tight relationship that exists between both of 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.

Workflow ontology. The main classes and properties of the workflow ontology and its relationships with the other ontologies in our approach are shown in Figure 3.

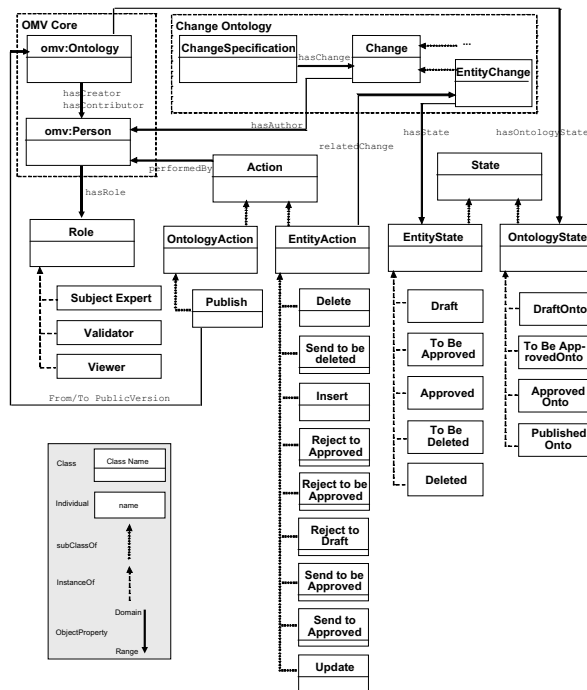


Fig. 3. Workflow ontology

The different roles of the ontology editors are modelled as individuals of the `Role` class that is related to the `Person` class of the OMV core ontology (i.e. a person has a role). To explicitly model the separation between the possible states of ontology elements (i.e. classes, properties and individuals) and the possible states of the ontology itself, the `State` class is specialized in two subclasses (i.e. `EntityState` and `OntologyState`). Similarly to the roles, the possible values of the states are modelled as individuals of their respective subclass. Furthermore, the two subclasses of `State` allow to represent the appropriate relationships at the element and ontology level: To specify that an ontology element has a particular state we rely on the class

EntityChange from the change ontology which is associated to a particular ontology element (as described in section [13]) and associate it with subclass EntityState, and 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.

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 (i.e. EntityAction and OntologyAction. To track the whole process (and keep the history) of the workflow, the possible actions are modelled as subclasses of the appropriate Action subclass. Similar to the states, the two subclasses of Action also allow to represent 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. As we explained before, actions at the ontology level are performed automatically by the system except from publish which changes the public version of the ontology. Therefore, the only subclass of OntologyAction is Publish that is associated to the class Ontology to specify the previous and next public version of the ontology.

3.2 Implementation Support

Our approach has been implemented within the NeOn Toolkit⁸, an extensible ontology engineering environment based on Eclipse, by means of a set of plugins and extensions. A high level conceptual architectural diagram of the involved components is shown in Figure 4. We present in the following, first the change capturing related components (i.e. left side of the figure), then the workflow management related components (right side of the figure), next the user related components for editing and visualizing ontologies (and related information) in the editorial workflow (upper part of the figure) and finally our distributed registry implementation (bottom part of the figure).

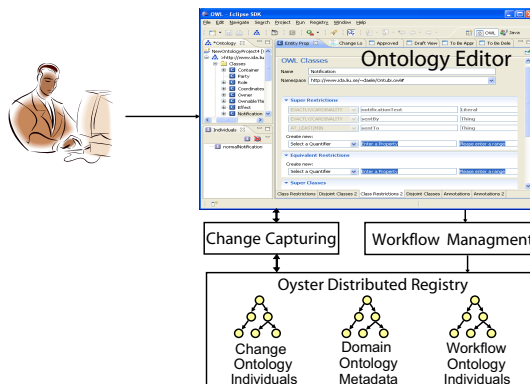


Fig. 4. Conceptual architecture for the collaborative ontology development support

Change Capturing Components. Once the ontology editor specifies that he wants to monitor an ontology, changes are automatically captured (*change management*

⁸ <http://www.neon-toolkit.org/>

requirement) from the ontology editor by a change capturing plugin. This plugin is notified about events that consist of ontology changes performed by the user in the ontology editor. For each of these events, the change is represented according to the change ontology by creating the appropriate individual. For example, adding a class individual in the ontology editor creates the entity change "Add Individual" and the two corresponding atomic changes (OWL 2 axioms): "Add Declaration" and "Add ClassMember". As described by the change ontology, each individual includes relevant information such as the author, the time, the related ontology, etc. The individuals are stored into the Oyster distributed registry [12]. This plugin is also in charge of applying changes received from other clients to the same ontology after Oyster synchronizes the changes in the distributed environment (see last subsection). Finally, this plugin extends the NeOn Toolkit with a view to display the history of ontology changes (*visualisation requirements*).

Workflow Management Components. In our implementation, the workflow management component (i) takes care of enforcing the constraints imposed by the collaborative workflow, (ii) creates the appropriate action individuals of the workflow ontology and (iii) registers them into the distributed registry. Hence, whenever a new workflow action is performed, the component performs the following tasks:

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

Ontology Editing and Visualization Components. To support the workflow activities (*workflow activities requirements*) we rely on the NeOn Toolkit which comes with an ontology editor that allows the editing of ontology elements. Additionally, according to the *visualisation requirements* the NeOn Toolkit is extended with a set of views that allow editors to (i) see the appropriate information of ontologies in the editorial workflow and (ii) perform (as described in 3.1) the applicable workflow actions (*approve, reject, etc.*), depending on their role. There are four views⁹:

- *Draft view*: Shows all proposed changes (from all editors) to that ontology version. In accordance to FAO scenario the changes of the current editor are editable while changes from other editors are non editable (see Figure 5).
- *Approved view*: Shows the approved changes.
- *To Be Approved view*: Shows all changes (from all editors) pending to be approved.
- *To Be Deleted view*: Shows all proposed deletions (from all editors).

⁹ Subject experts see the first two views, validators see the latter three.

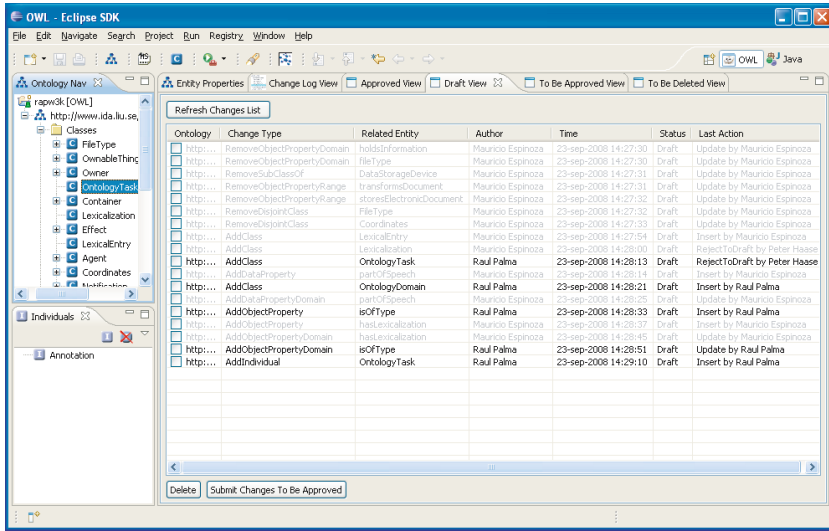


Fig. 5. Draft View in the NeOn Toolkit

Distributed Registry. Ontologies are stored within a repository and their metadata is managed by the Oyster distributed registry¹⁰ (*change management requirement*). The metadata includes information about ontologies and users (represented using OMV), the changes to the ontology (represented using the change ontology) and about the actions performed (represented using the workflow ontology). For each change the status is also kept to support the editorial workflow. When a new change is registered into an Oyster node, Oyster automatically updates the log history keeping track of the chronological order of changes: It gets the last registered change (using the "Log" class) and adds it as the previous change of the current one. Then it updates the "Log" class to point to the current change.

The local Oyster nodes contact each other creating a distributed ontology registry. In this distributed environment, Oyster also propagates the ontology changes, thus allowing the notification of new changes to ontology editors (*change management requirement*). That is, once we have the required changes in a machine-understandable format, the system propagates them to the distributed copies of the ontology. For this task, we follow a synchronization approach that is a combination of a push and pull mechanism. During the synchronization, 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 super node) such that if a node goes offline before all other nodes pull the new changes, the node changes are not lost. In this way, Oyster minimizes the conflicts or inconsistencies due to concurrent editing as it automatically synchronizes changes periodically (and it allows to force the synchronization immediately) in the distributed environment such that every editor will have an up-to-date copy of the ontology with the proposed changes (*concurrency control and conflict*

¹⁰ <http://ontoware.org/projects/oyster2/>

resolution requirement). Nevertheless, conflicts in the collaborative workflow could still occur as logical conflicts in the form of inconsistencies or conflicts due to concurrent editing of an ontology. The strategies to deal with those potential problems are out of the scope of this paper, and we refer the reader to [13] for additional information.

4 Related Work on Collaborative Ontology Development

The problem of collaborative ontology development has been partially addressed in the literature with methodological and technological results, which are not necessarily aligned. In the remainder of this chapter we mainly focus on the existing works of ontology development where some kind of reviewing process has been acknowledged.

In [16], [14] the authors introduce DILIGENT, an ontology engineering process for decentralized cases of knowledge sharing. It identifies several key roles involved in collaboratively building the same ontology. The process entails different users in the creation of a shared ontology and adaptation to local needs and a control board in charge of deciding how the shared ontology will be changed based on the user requests. DILIGENT also considers the provision of arguments for the requested changes and design decisions in a semi-formal way (similar to [15]).

A related work – DOGMA-MESS – is presented in [1] (and [7]). The authors propose a generic model for understanding the interorganisational ontology engineering process where the knowledge moves in an upward spiral starting at the individual level, moving up to the organisational level, and finally up to the interorganisational level.

In [10] the authors present the Change and Annotation Ontology (CHAO) to represent changes between two versions of an ontology and user annotations related to these changes, and two Protégé plugins: The Change-management plugin that provides access to a list of changes (i.e. instances of CHAO) and enables users to add annotations to changes and the PROMPT plugin (also introduced in [11]) that provides comparisons between two versions of an ontology, allowing to examine the list of users who performed changes and to accept and reject changes. Finally they introduce the client/server mode in Protégé for synchronous editing by multiple users.

Another similar tool support is presented in [17]. The authors introduce an extension of the existing Protégé system that supports collaborative ontology development (i.e. Collaborative Protégé). The extension enables (1) the annotation of ontology components and ontology changes and (2) the searching and filtering of user annotations based on simple or complex criteria. The authors also propose two types of mechanisms for voting change proposals (i.e. a *5-star* voting or a *Agree/Disagree* type of voting).

Although [16] and [1] consider the collaborative development of ontologies in a distributed setting, it is not clear how change requests are represented, there is no explicit tracking of the change operations in the shared ontology that would be useful for local users to identify the approved changes or compare it with the local copy and there is no history of the rejected changes. Moreover, local users are not notified automatically of changes and consequently they could be working with different versions of the ontology which might hamper the interoperability. Also, [1] does not consider how users interact in the process depending on their role.

A main difference in [10] and [17] with respect to our solution is that the tracking of changes and curator actions (i.e. accept/reject changes) is done in a centralized manner i.e. either in a local copy or in a centralized server. Additionally, although the approaches consider the reviewing of changes (e.g. acceptance/rejection of changes), it is not clear what kind of roles (and related permissions) are considered, how those actions are traced or how is the process flow for the reviewing.

In our solution, we rely on a formal representation of changes which provides the basis for the creation of change logs that support e.g. the comparison of ontologies or the synchronization of distributed copies of the ontology. Moreover, we identify different user roles involved in the process of the ontology development and propose to formalize the reviewing process in a machine-understandable format such that all taken actions can be tracked and exchanged. In our solution, users are able to use the same version of the ontology (i.e. a local copy) and work in a decentralized manner given that changes and curator actions are maintained in a distributed registry which is in charge of synchronizing the information automatically.

5 Discussion

The need for a systematic approach to ontology development in a highly distributed environment has been emphasized many times in the past in the ontology engineering community. As a result, different solutions have been proposed ranging from informal or lightweight strategies (e.g. [4]) to semi-formal approaches like in the Gene Ontology project to formal methodologies (e.g. [16]). In this paper we have presented our solution to support the collaborative ontology development in distributed environments. It consists of a formal strategy where the ontology development process is explicitly represented. The criteria for choosing this strategy was based on the analysis of the requirements of large organisations using as test case the FAO scenario.

Hence, we proposed two generic workflows specialised at different levels of abstraction (i.e. ontology element level and ontology level). Our proposal includes the definition of a workflow ontology for the formal representation of the workflow process. Additionally, we introduced the role of the workflow in the infrastructure required for the management and control of ontology changes and describe its relationships with other components and activities (i.e. change representation, versioning, etc.). We illustrated in a simple scenario how the workflow ontology supports the collaborative ontology development and its tight relationship with the ontology for the representation of changes. Finally, we introduced our implementation to support the proposed model.

Although we are already evaluating individual components of our approach, our next step is the complete evaluation of our approach within FAO and other scenarios. In the future, we plan to provide additional features to the NeOn toolkit such as a *Change View* to shows the changes (diff) between two versions of an ontology according to the change ontology or a full undo/redon support. Further, we are working on the integration of our work with other threads of related work, such as argumentation support.

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/>).

References

1. de Moor, A., De Leenheer, P., Meersman, R.: DOGMA-MESS: A meaning evolution support system for interorganizational ontology engineering. In: Proc. of the International Conference on Conceptual Structures (ICCS 2006), Aalborg, Denmark. Springer, Heidelberg (2006)
2. Gangemi, A., Lehmann, J., Presutti, V., Nissim, M., Catenacci, C.: 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)
3. Hartmann, J., Palma, R.: OMV - Ontology Metadata Vocabulary for the Semantic Web, vol. 1.0 (2005), <http://omv.ontoware.org/>
4. Hepp, M., Bachlechner, D., Siorpaes, K.: Ontowiki: Community-driven ontology engineering and ontology usage based on wikis (2005)
5. Klein, M.: Change Management for Distributed Ontologies. PhD thesis, Vrije Universiteit, Amsterdam (2004)
6. Kozaki, K., Sunagawa, E., Kitamura, Y., Mizoguchi, R.: A framework for cooperative ontology construction based on dependency management of modules. In: Proceedings of the International Workshop on Emergent Semantics and Ontology Evolution (ESOE2007) at ISWC/ASWC2007, Busan, South Korea (November 2007)
7. De Leenheer, P., Mens, T.: Ontology Evolution. State-of-the-art and Future Directions. In: Ontology Management. Semantic Web, Semantic Web Services, and Business Applications. Springer, Heidelberg (2007)
8. Muñoz-García, Ó., Gómez-Pérez, A., Iglesias-Sucasas, M., Kim, S.: A workflow for the networked ontologies lifecycle. A case study in FAO of the UN. In: Proceedings of the CAEPIA-TTIA 2007, Spain. Springer, Heidelberg (2007)
9. Muñoz-García, O., Kim, S., Iglesias Sucasas, M., Caracciolo, C., Bagdanov, A., Wang, Y., Haase, P., Suarez-Figueroa, M., Gomez-Perez, A.: Software architecture for managing the fisheries ontologies lifecycle. Technical Report D7.4.1, NeOn Consortium (October 2007)
10. Noy, N., Chugh, A., Liu, W., Musen, M.: A framework for ontology evolution in collaborative environments. In: International Semantic Web Conference, pp. 544–558 (2006)
11. Noy, N., Kunnatur, S., Klein, M., Musen, M.: Tracking changes during ontology evolution. In: International Semantic Web Conference (2004)
12. Palma, R., Haase, P.: Oyster - sharing and re-using ontologies in a peer-to-peer community. In: International Semantic Web Conference, pp. 1059–1062 (2005)
13. Palma, R., Haase, P., Wang, Y., d'Aquin, M.: D1.3.1 propagation models and strategies. Technical Report D1.3.1, UPM; NeOn Deliverable, November (2007)
14. Pinto, S.: Ontoedit empowering swap: a case study in supporting distributed, loosely-controlled and evolving engineering of ontologies (diligent) (2004)
15. Stojanovic, L.: Methods and Tools for Ontology Evolution. PhD thesis, University of Karlsruhe (TH), Germany (August 2004)
16. Tempich, C.: Ontology Engineering and Routing in Distributed Knowledge Management Applications. PhD thesis, University of Karlsruhe (TH), Germany (2006)
17. Tudorache, T., Noy, N.: Collaborative protege. In: Workshop on Social and Collaborative Construction of Structured Knowledge (CKC 2007) at WWW 2007, Banff, Canada (2007)