

# Customizable Workflow Support for Collaborative Ontology Development

Abraham Sebastian, Tania Tudorache, Natalya F. Noy, Mark A. Musen

Stanford University, Stanford, CA 94305, US

{abeseb, tudorache, noy, musen}@stanford.edu

**Abstract.** As knowledge engineering moves to the Semantic Web, ontologies become dynamic products of collaborative development rather than artifacts produced in a closed environment of a single research group. However, the projects differ—sometimes significantly—in the way that the community members can contribute, the different roles they play, the mechanisms they use to carry out discussions and to achieve consensus. We are currently developing a flexible mechanism to support a wide range of collaborative workflows in the Protégé environment. In this paper, we describe our overall architecture that supports flexible collaborative workflows in Protégé. This architecture comprises an ontology for representing workflows for collaborative ontology development, a customizable ontology-development environment that our system generates based on a declarative description of a workflow, and a run-time integration with a workflow execution engine.

## 1 Overview of Workflows for Collaborative Ontology Development

*Collaborative ontology development* has become an active area of research and practice. On most large projects today, ontology development is a **collaborative effort**, involving both ontology engineers and domain experts. The number of users participating in development ranges from a handful (e.g., the Foundational Model of Anatomy [7]), to a couple of dozens (e.g., the National Cancer Institute’s Thesaurus [9]), to the whole community contributing to the ontology in some way (e.g., the Gene Ontology [3]).

With larger groups of users contributing to ontology content, many organizations define formal **workflows** for collaborative development, describing how project participants reach consensus on definitions, who can perform changes, who can comment on them, when ontology changes become public and so on. Some collaborative projects have been publishing and refining their workflows for years (e.g., the Gene Ontology). In other projects, ontology researchers are working actively with domain experts to make these workflows explicit and to provide tooling for supporting the specific workflows (e.g., ontology development for the UN Food and Agriculture Organization (FAO) in the NeOn project [4]).

These workflows differ from project to project, sometimes significantly. A workflow for a specific project usually reflects that project’s organizational structure, the size and the openness of the community of contributors, the required level of rigor in quality control, the complexity of the representation, and other factors.

We are currently working on providing comprehensive support for collaborative ontology development in the Protégé system.<sup>1</sup> Protégé enables users to edit ontology in a

<sup>1</sup> <http://protege.stanford.edu>

distributed manner, to discuss their changes, to create proposals for new changes and to monitor and analyze changes. Integrating support for collaborative workflows in such a system would mean having the tool itself “lead” the user through the workflow steps. For example, the tool can enable or disable certain options, depending on the user’s role, indicate to the user the current stage of the workflow and the actions expected or required from this user at this stage, or enable a user to initiate new activities. Because workflows may differ significantly from project to project, developers must be able to custom-tailor the workflow support in terms of both the execution steps and the user interface. Our goal is to develop a framework that would support as wide a variety of workflows as possible. We envision that as a group of ontology developers defines the workflow process for a new ontology project, they will describe this process declaratively using our *workflow ontology*. This description may include the list of roles and corresponding privileges, the steps that a change must go through in order to be published, the way tasks get assigned and executed, and so on. Our tools will then use this description to generate a custom-tailored ontology-development environment. The team can then use this environment for their collaborative development, with a workflow engine controlling the execution steps.

This paper makes the following contributions:

- We develop an **architecture** for supporting customizable workflows for collaborative ontology development. This architecture integrates tools for ontology development (Protégé), declarative description of workflows (a workflow ontology), discussion support, and a workflow engine (Section 3). Our design is driven by a set of requirements (Section ??) that we have identified by studying a large number of projects that use collaboration in ontology development.
- We present a **prototype implementation** of the architecture by
  - enabling generation of a custom-tailored ontology-development environment from a set of workflow- ontology instances (Section 5);
  - mapping our workflow ontology to a workflow engine in JBoss (Section ??);
  - providing a Protégé-based implementation of specific activities in the workflow engine (Section ??).

## 2 Related Work

Research that affects our work and that we will discuss in this section comes from both the ontology community and the domain of business-process modeling.

C-ODO is an OWL meta-model for describing collaborative ontology design [2, 1]. The model focuses on describing design rationale, design decisions, and argumentation process. C-ODO also represents workflows, more specifically, *epistemic workflows*. Epistemic workflows describe the flow of knowledge from one rational agent to another. The focus of an epistemic workflow is the knowledge resource itself and the workflow is focused on the description of how the knowledge resource changes. In our work, we focus on the *execution workflows* describing the actions taken by the agents, which is a complementary view to the one taken by C-ODO.

Several environments implement specific workflows for ontology development. For example, the **Biomedical Grid Terminology (BiomedGT)**<sup>2</sup> is a terminology product

<sup>2</sup> <http://biomedgt.org>

launched by the National Cancer Institute (the developers of the NCI Thesaurus [9]) to enable the biomedical research community to participate in extending and refining the NCI Thesaurus. BiomedGT defines several roles for the members of the community and the steps required from each user to perform a change.

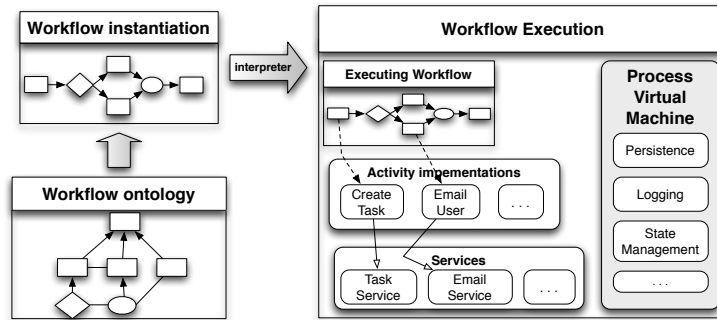
The DILIGENT methodology for collaborative development [10], which has been used in several European projects, and its implementation in the coefficientMakna system focuses on the formal argumentation process in the ontology development. The authors of the DILIGENT methodology have developed a formal argumentation ontology that defines different types of arguments and roles that users can play in the arguments. coefficientMakna [10]—also based on a wiki platform—is designed explicitly to support the DILIGENT argumentation-based approach to ontology engineering.

We use the insights provided by the authors of these tools and the specific solutions that they adopted in our approach. However, our goal is different from the goals of the authors of these systems: We want to develop a framework that works for a vast variety of *different* workflows, rather than for a specific one. We propose a general framework and a supporting implementation that is easily customizable for a new workflow description. In other words, the DILIGENT or BiomedGT workflows are specific instantiations of our framework. Most likely, the tools generated in this way would be slightly less “perfect” for each specific workflow. The other side of the trade-off, however, is that users do not need to design a new tool for each new workflow or to tweak an implementation each time their workflow changes. In our case, a user defining a new workflow needs to change only the formal description (instantiation) of the workflow.

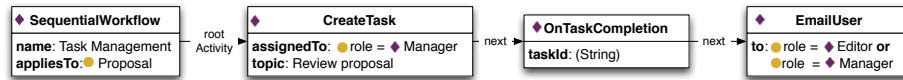
### 3 The Protégé Architecture for Flexible Workflow Support

Figure 1 shows our overall architecture that provides this generic and flexible support for collaborative workflows in Protégé. We are currently implementing components of this architecture. There are three main steps in this support: description, instantiation, tool-generation, and execution. The **description** level is a workflow ontology, defining what components a workflow can have. It provides a formal language for describing workflows for collaborative ontology development. The ontology contains the generic notion of roles, activities, tasks, and control structures (e.g., `CreateProposal`, `ChangeClassDefinition`). Developers then describe their specific workflows by creating a set of instances of the classes in the workflow ontology. During this **instantiation** step, developers specify which roles users have, what activities users with each role can perform, the order of activities, and the conditions that trigger new actions or new workflows. Finally, during ontology development, a **workflow execution engine** interacts with the Protégé environment and controls the flow of operations in which the distributed clients participate and **generates a custom-tailored ontology-development environment** that reflects the workflows for the specific project. The workflow engine maintains the state of the workflow and provides logging services to track all events and actions in the workflow execution.

We describe the workflow ontology and the instantiation process in Section 4. We discuss the tool generation and the integration with the workflow engine in Section 5.



**Fig. 1. Components of the workflow support in Protégé.** A specific workflow is modeled as instances of the Workflow ontology. The workflow engine instantiates the workflow model for a specific execution by traversing the graph of the workflow instance and executing the actions associated with each task. The workflow engine (in our case, PVM), performs such functions as providing persistence, logging, etc.



**Fig. 2. The set of instances in the workflow ontology describing the task-management workflow.** Each rectangle is an instance of a class in the workflow ontology. The top part of the rectangle indicated the name of the corresponding class.

## 4 Defining and Instantiating the Protégé Workflow Ontology

In our earlier work [8], we presented the details of the Protégé workflow ontology and we evaluated its generality by representing different workflows in it. The `Workflow` class represents the workflow object. Each instance of this class describes a workflow (e.g., an approval workflow or a voting workflow). Each workflow is associated with a set of initialization parameters, a workflow target, a partially ordered set of activities or states. Each workflow for collaborative ontology development is associated with ontology elements, ontology changes, or discussions about an ontology (the latter are themselves linked to ontology elements or changes)—a workflow target. Workflows are defined as a partially ordered set of *activities* (instances of the `Activity` class), with one activity invoking the next. Subclasses of the `Activity` class define specific activities in the collaboration process.

The workflow ontology provides the language and the primitives for describing collaborative workflows. For each specific project, developers create a set of instances of classes in this ontology to describe the process for that project. For example, Figure 2 shows a set of instances of classes in the workflow ontology that describes a simple task-management workflow. It is a sequential workflow (instance of `SequentialWorkflow` in the diagram). The workflow starts with a user with the role `Manager` creating the task (`CreateTask` instance). The workflow then waits for the task to be completed, and an email is sent to the `Manager` and the `Editor` once that happens.

For an instantiation of a more complex workflow, please see our earlier work [8].

## 5 Generating Customized Development Environment in Protégé

The Protégé open-source ontology-editing environment provides the framework and a set of major components for our implementation. Users can build ontologies in Protégé using different representation formalism ranging from *Frames*, to *RDF(S)* and *OWL*, and store them in file or database backends. Protégé is both robust and scalable and is being used in production environment by many government and industrial groups. The *ontology and knowledge base API* and the *plugin architecture*—some of the most successful features of Protégé—allow other developers to implement their own custom extensions that can be used either in the Protégé user interface or as part of other applications.

We use Protégé to generate some of the software components for our system from the ontologies that describe them. We then generate a custom-tailored Protégé environment that is based on the description of a specific workflow, and that is integrated with the Process Virtual Machine workflow engine.

In building Collaborative Protégé and the workflow support, we have used a number of ontologies and corresponding modules. Each of these ontologies drives one of the software components of the overall architecture. To use the ontologies in the Protégé toolset, we automatically generate the corresponding APIs from the ontologies themselves using a Java code generator available as a plugin in Protégé. Generating Java code directly from ontologies, helped increase the efficiency of our software development. The generated API provides high level access to concepts in an ontology while hiding the underlying storage details [5, 11].

Similarly, we use the workflow ontology to generate the Java API to access properties of a workflow definition in the interpreter. We have found the code generation to be a very effective technique for generating APIs in an ontology-driven software.

Once we generate a custom-tailored environment, we need a workflow engine to monitor the execution process. We chose the *Process Virtual Machine (PVM)*<sup>3</sup> as our workflow engine. PVM is a software framework from jBoss that provides a common set of services to build execution engines for workflow specification languages. It does not itself define a process language but instead provides common services such as workflow state management, persistence and logging. PVM can be mapped to any graph-based language for describing the workflow. Thus, we use our workflow ontology as a workflow-definition language and integrate it with PVM. PVM is embeddable in any Java application. This feature enables easy integration with the existing Protégé toolset.

A PVM workflow is a directed graph, in which nodes represent states in the workflow and edges are transitions between the states. Each node (i.e., state) in a PVM workflow may have associated with it an activity that defines a specific runtime behavior (e.g. send email to a user). A Java class can define the runtime behavior of a node by implementing the *Activity* Java interface provided by PVM. In our example (Figure 3), the first state in the workflow is `CreateTask`. We have associated to it a Java class called `CreateTask` that implements the *Activity* interface of PVM. Thus, when the workflow transitions to the `CreateTask` state, the workflow engine will invoke the behavior of the state (defined in the `CreateTask` Java class). In a similar manner, we

---

<sup>3</sup> <http://docs.jboss.com/jbpm/pvm/>

have implemented Java classes corresponding to each subclass of the `Activity` class in our workflow ontology.

Given a set of instances of the workflow ontology that describe a specific workflow (e.g., Figure 2), our interpreter parses the instances to derive the structure of an executable workflow (Figure 3). The interpreter thus does the function of mapping a workflow description in our ontology to a description that is executable by PVM. For example, we have defined the example workflow as a set of instances in the workflow ontology as described in Section ??, and our interpreter has parsed these instances and created the structure for an executable workflow in PVM. Specific events in the Protégé user interface (e.g. the user clicks on the “Create task” button) can trigger the execution of the generated workflow.

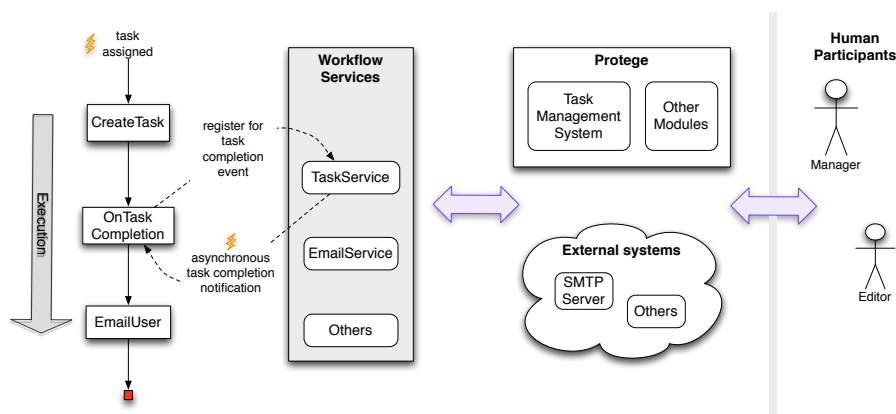
As we have mentioned, runtime behavior of a workflow state is defined by an activity. Activities may use facilities provided by one or more *services*. Services interface with external modules, such as the task-management service in Protégé (Figure ??), and provide an interface to all executing workflow instances. Services may pool and manage external resources; for example, they may use persistent storage, or persistent connections with external modules. We have implemented a task-management service that provides methods to create, update, or delete tasks and also to register for events such as task completion. Several activities (different states in the workflow) can invoke this service. In our example, the *TaskService* is invoked by the first state (i.e. `CreateTask`) and by the second state (i.e. `OnTaskCompletion`).

The workflow engine transparently persists the state of workflows as it executes so that they can survive events such as machine reboots. Thus implementations of activities only need to encode the domain-specific application logic.

## 6 Discussion and Future Work

In this paper, we have presented a generic architecture for supporting collaborative workflows for ontology development. Our work integrates semantic-web and software-engineering approaches in several ways and provides some lessons learned and new challenges to address.

First, we use a standard workflow engine to provide services for an ontology-development workflow. In essence, we tightly integrate the two sets of tools, and provide the proof of concept that an off-the-shelf workflow engine can indeed work as a workflow engine for ontology editing. The work presented here, however, is only a first step in this direction: We must implement other critical services that we have catalogued from our study of existing ontology-development projects. We envision some challenges as we implement these services. For instance, it may not always be possible to link user-interface elements to elements of the workflow ontology: there may not be a one-to-one correspondence between the two. Thus, we may need a more complex mechanism for the mapping. At this point, we have implemented fairly simple workflows. As we try more complex workflows, we expect to encounter other challenges in representing their structures. More complex workflows will also raise challenges in synchronizing workflows that are running in parallel for multiple users, projects, and sessions, and may depend on one another. As we gain more experience with implementing collaborative workflows for ontology development, we plan to address these challenges.



**Fig. 3. Integration of workflow engine with Protege.** This diagram uses the example of a simple AssignTask workflow to show how the workflow engine is integrated with Protégé modules and external systems. The workflow is initiated by the creation of a new task. The first activity *CreateTask* invokes the Protégé task-management system to create a task for the Manager. The next activity, *OnTaskCompletion*, registers with the workflow engine’s *TaskService* to be notified when the task is completed by the manager. When the *TaskService* is notified by the system that the user has completed the task, it proceeds with the workflow. The *EmailUser* activity is implemented using the *EmailService*. The *EmailService* sends emails using an SMTP server.

Second, we use the Model-Driven Architecture (MDA) approach to generate a large part of our software. Specifically, we generate an API for the workflow component from the workflow ontology itself. We use the same approach for generating APIs for other components of ontology evaluation and collaborative ontology development in Protégé. This approach enables us to describe ontology changes, types of annotations and other components declaratively directly in the Protégé GUI and simply to regenerate the API, as these components evolve. However, one of the painful lessons that we learned during our software development using the generated code from an ontology, was that whenever the ontology changed, we had to adapt the rest of the application code that depended on the generated code. If the ontology changed dramatically, the effort in code adaptation was considerable. Therefore, our advice is to generate the code only when the ontology has reached a reasonable level of maturity and when you do not expect it to change too much in the future. Another lesson was the need for better code generators that are able to generate the Java interfaces corresponding only to top level classes in the ontology. The application code should be dependent only on these top level interfaces. As a result, the application code is less prone to major modifications when the underlying ontology changes.

In Section ??, we have identified a set of requirements for supporting collaborative workflows. The architecture that we described in this paper addresses these requirements. First, our approach is *general* and can represent a wide variety of workflows. Second, developers can *extend* our workflow ontology and add new types of ontology-

related actions and to provide their implementation. Our tool-generation can automatically integrate these extensions. Developers can *reuse* components of our architecture through the API. Our workflow support is tightly *integrated* with the Protégé ontology-development environment and, for the most part, the user interaction is the same as in the “classic” Protégé environment. We used the `Role` ontology to drive the access policies. Finally, PVM provides *persistence* across sessions and server reboots and during the long-running workflows.

Our key next step will be implementing our architecture more fully, supporting other types of ontology- development activities. We plan to evaluate our approach in general and our implementation in particular by defining workflow process (instantiation of the workflow ontology) and creating custom-tailored editing tools for some of the projects that we collaborate with. These projects include the development of the National Cancer Institute’s Thesaurus [6] and the ATHENA project at the VA Palo Alto Healthcare System [11].

## Acknowledgments

This work was supported in part by a contract from the U.S. National Cancer Institute. Protégé is a national resource supported by grant LM007885 from the United States National Library of Medicine.

## References

1. C. Catenacci. Design rationales for collaborative development of networked ontologies state of the art and the collaborative ontology design ontology. Technical Report NeOn Project Deliverable D2.1.1, 2006.
2. A. Gangemi, J. Lehmann, V. Presutti, M. Nissim, and C. Catenacci. C-ODO: an OWL meta-model for collaborative ontology design. In *Workshop on Social and Collaborative Construction of Structured Knowledge at WWW2007*, Banff, Canada, 2007.
3. GOConsortium. Creating the Gene Ontology resource: design and implementation. *Genome Res*, 11(8):1425–33, 2001.
4. O. Muñoz García, A. Gómez-Pérez, M. Iglesias-Sucasas, and S. Kim. A Workflow for the Networked Ontologies Lifecycle: A Case Study in FAO of the UN. In *The Conf. of the Spanish Assoc. for Artificial Intelligence (CAEPIA 2007)*, LNAI 4788. Springer, 2007.
5. N. F. Noy, A. Chugh, W. Liu, and M. A. Musen. A framework for ontology evolution in collaborative environments. In *5th Intl. Semantic Web Conf., ISWC*, Athens, GA, 2006.
6. N. F. Noy, T. Tudorache, S. de Coronado, and M. A. Musen. Developing biomedical ontologies collaboratively. In *AMIA 2008 Annual Symposium*, Washington, DC, 2008.
7. C. Rosse and J. L. V. Mejino. A reference ontology for bioinformatics: The Foundational Model of Anatomy. *Journal of Biomedical Informatics.*, 2004.
8. A. Sebastian, N. F. Noy, T. Tudorache, and M. A. Musen. A generic ontology for collaborative ontology-development workflows. In *16th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2008)*, Catania, Italy, 2008. Springer.
9. N. Sioutos, S. de Coronado, M. Haber, F. Hartel, W. Shaiu, and L. Wright. NCI Thesaurus: A semantic model integrating cancer-related clinical and molecular information. *Journal of Biomedical Informatics*, 40(1):30–43, 2007.
10. C. Tempich, E. Simperl, M. Luczak, R. Studer, and H. S. Pinto. Argumentation-based ontology engineering. *IEEE Intelligent Systems*, 22(6):52–59, 2007.
11. T. Tudorache, N. F. Noy, S. Tu, and M. A. Musen. Supporting collaborative ontology development in protege. In *7th Intl. Semantic Web Conf., ISWC*, Karlsruhe, Germany, 2008.