Collaborative editing of Ontologies using Fluent Editor and Ontorion

A. Seganti[1], P. Kapłanski[1 2], and P. Zarzycki[1]

[1] Cognitum, Wał Miedzierzynski 631, Warszaw (Poland)
[2] Gdansk University of Technology, Gdansk (Poland)

Abstract. In this paper we present two tools that we are developing at Cognitum for manage large knowledge bases: Fluent Editor and the Ontorion Server. We will how we have been able to build a collaborative knowledge management system using these two tools. We will show how this system can be used for concurrent modification of knowledge and how we are managing multiple modification to the same knowledge.

1 Introduction

In this paper we will show how we have used Fluent Editor and Ontorion to build a collaborative editing tool for large ontologies that uses controlled natural language and modularization.

Fluent Editor is a editing tool for modifying ontologies using Ontorion Controlled Natural Language as an interface to editing (OCNL). Fluent Editor main features are: autocompletition helping the user in writing the correct sentences, many tools to interact with third party components (R plugins, Protégé plugin,...), OCNL interface, reasoning and materialization, complex CNL queries to the current ontology, complete reference management (import/export/referencing of OWL/RDF ontologies), graphical representation of the ontologies and interaction with the Ontorion Server. With its 2000+ users, Fluent Editor is quickly becoming an alternative to the OWL based editors like Protégé [1].

Ontorion Server [2] is the server equivalent of Fluent Editor, designed for having scalable reasoning, OCNL interface (querying and saving), SPARQL interface, OWL2/SWRL compatible, tunable reasoning (currently OWL-DL and OWL-RL profiles are available) and high availability. Ontorion has advanced reference management, giving the user the possibility to define a prefix to namespace map to be used for all entities. Inside Ontorion, an innovative modularization algorithm (based on [3]) is used to modularize the knowledge allowing for scalable reasoning.

Both Fluent Editor and Ontorion Server are exposing an OCNL interface. OCNL is a controlled natural language designed on the one hand to be fully compatible with OWL2 and SWRL W3C standards and on the other hand to be intuitive enough for people with little knowledge of logic to write knowledge bases. Internally both products are using description logic as the interface between these two worlds.

In a first part we will briefly introduce Fluent Editor and the Ontorion Server. Then we will show the system architecture of the Collaborative Knowledge editing that we have built using these two tools. In this last part we will show how the modularization algorithm has been used to simplify knowledge modification.

2 Fluent Editor and the Ontorion Server

Fluent Editor and the Ontorion Server are products created by Cognitum to manage knowledge. Both products are fully compatible with the W3C standards and use internally description logic for all logic related operation. As a user interface for both systems, we are using the Ontorion Controlled Natural Language (OCNL) that is an controlled language equivalent to OWl2 and SWRL languages.

2.1 Fluent Editor

Fluent Editor is a knowledge editor tool for editing standard W3C ontologies. In Fluent Editor, we provide: the reasoner, the taxonomy tree, the materialized graph, an interface to R programming using our ROntorion package, possibility to import/export from/to Protégé editor, graphical representation of the ontology, reference management, reasoning profile validation of the current ontology and Ontorion interoperability.

In Fluent Editor two different interfaces for reasoning have been implemented: the reasoner and the materialized graph. The reasoner is using a standard OWL-DL reasoner for reasoning over the content of the ontology. By default Fluent Editor loads HermiT [4] but it is possible to implement a simple C# interface to add other reasoners to Fluent Editor. In the materialized graph, we are using a customer reason to materialize the knowledge in a graph and then query the materialized graph. In this tab we support SWRL builtins and two different reasoning profiles: OWL-DL or OWL-RL+. In both of the reasoning tabs, the user can make OCNL queries to the ontology and the results are displayed in OCNL. Due to the difference in the underlying reasoning, the query expressivity in the two windows is slightly different and the result window also is different: in the reasoner the result will consist of all entities answering the query together with sub and super concepts while in the materialized graph the instances answering to the query are shown.

2.2 Ontorion Server

The Ontorion Server [2] is a knowledge management server that offers scalable and tunable reasoning. Ontorion has an OCNL interface through which the user can make queries to the knowledge together with a SPARQL interface for complex SPARQL queries. It is possible to use the Ontorion Server through a WCF API that exposes all main functions to manage and query knowledge, manage users and databases and to use more advanced functionalities.

Reasoning in Ontorion is very similar to what is done in Fluent Editor's materialized graph. The main difference is that each time that something is saved, Ontorion will extract the module of the knowledge related to the sentences that are being saved and reason over this knowledge. This is possible because Ontorion as an implementation of a modularization algorithm (based on [3]) and allows to scale reasoning also for a large number of entities. In Ontorion two reasoning modes can be user: OWL-DL and OWL-RL+.

In Ontorion, the adminstrator of the system can manage multiple knowledge bases and each user can have different access to the knowledge bases currently loaded. Furthermore there is a knowledge versioning management to know when something has changed in the knowledge.

2.3 OCNL

OCNL [5] is a controlled natural language for writing ontologies that is compatible with OWL2+SWRL standards. OCNL is currently implemented for English, Polish and German but can be extended to other languages. A typical sentence in OCNL will look like *Every man is a human-being.*. In this sentence we can see that in OCNL complex words needs to be separated by hyphens.

In OCNL it is possible to express all OWL constructs, SWRL rules (together with builtins) and use OWL references (prefixes *man[pfx]* or full namespaces *man[<http://www.mynamespace.com>]* can be used). Additionally, OCNL defines two extensions to the standards: integrity constraint rules (ICR) and active rules. ICR are a way of expressing modalities (e.g. *Every man* ***must*** *have-name (some string value)*), currently must, can and should are available. It is then possible to reason the instances that fulfill or not the modalities. Using active rules, it is possible to define imperative code (currently C#) that should be executed when a certain condition applies (e.g. *If a man has-cat a cat then* ***for*** *the man and the cat execute (C# code...)*). Using the active rules, we have implemented in Ontorion an agent based system that act on changes in the knowledge.

3 Collaborative knowledge editing

3.1 System architecture

The collaborative knowledge editing environment that we have implemented is using Fluent Editor and the Ontorion Server together. The architecture of the system is presented in Figure 1.

The server side of the collaborative editing architecture is comprehensive of Ontorion and the Ontorion WCF API. The API exposes all Ontorion functionalities and has been used in Fluent Editor to implement the client. Ontorion has a complete user management system in which each user can have different access level to each database in Ontorion. For the collaborative editing, each user will connect using his/her Ontorion credentials and the changes to the ontology will be logged.

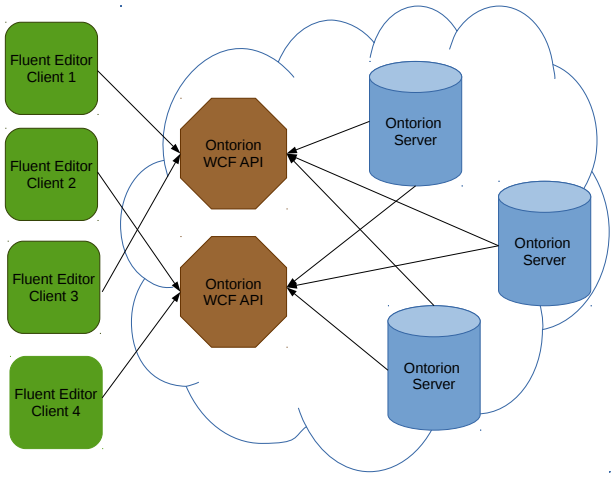

Fig. 1. Schema of the collaborative editing architecture using Fluent Editor and Ontorion.

On the client side, Fluent Editor is installed locally on the computer of each user. After the user opens Fluent Editor, he/she can connect to Ontorion and enters in the Ontorion Mode in Fluent Editor. In this mode it is possible to download/modify/add knowledge, see the taxonomy tree and make SPARQL queries to the knowledge in Ontorion.

3.2 Ontorion Mode

User interface When in Ontorion mode 2, it is possible to Save, Download, Refresh and Clear the module regarding the knowledge the user is interested in. The whole point of the Ontorion mode is to manage modules of knowledge (see 3.3): the users adds entities to the signature, the module corresponding to the signature is downloaded and the user modifies or add knowledge in the module.

Module management is based on the "signature" of the module. In the signature there are the entities (concepts, instances or properties) that are of interest to the user and each time a new entity is added, the module corresponding to the signature will be downloaded. For example, if the user adds to the signature the instance 'New-York' and the relation 'lives-in', the module will contain all knowledge relative to New-York and lives-in.

The Ontorion mode is particularly interesting when used for collaboratively editing a big knowledge base stored in Ontorion. Indeed more than one user can be connected to Ontorion at the same time and it is possible that both users are working on the same knowledge at the same time. In this case, when one of the users will commit his/her changes to Ontorion, the other user will be notified of the changes as he/she will see on the bottom left of Fluent Editor the resync icon.

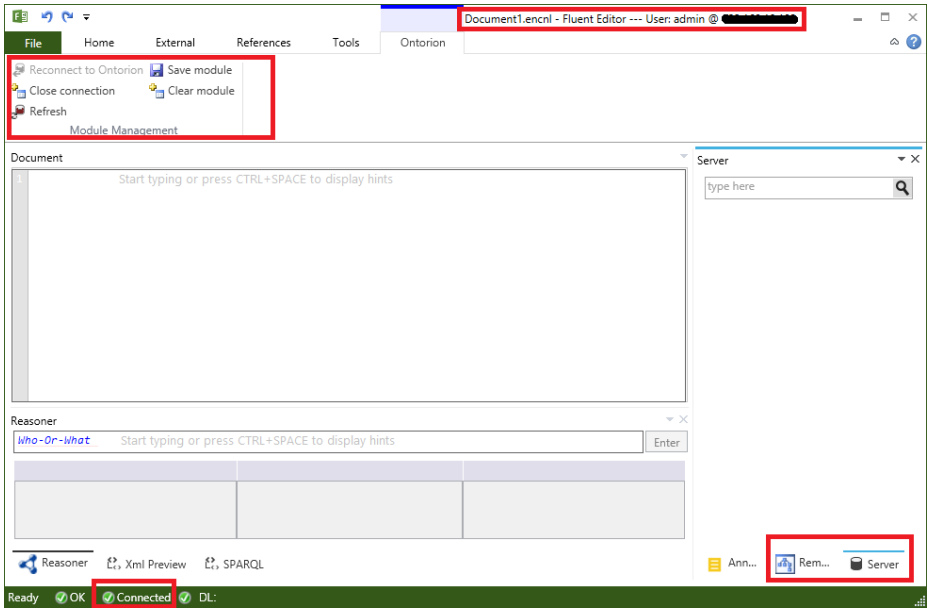

Fig. 2. Screenshot of the tab showing in Ontorion Mode inside Fluent Editor.

In this case the user need to click the **Refresh** button in the Ontorion tab that will show her/him the sentences added from the other user to the module that is currently loaded (for example in Figure 3 someone added a new attribute for the instance Eli)

By clicking **Change**, this sentence will be added to the knowledge currently loaded into Fluent Editor.

Taxonomy Tree and SPARQL queries Taxonomy tree is displayed together with each ontology being edited in Fluent Editor and is generally built upon data from the current ontology and all referenced ontologies. Selecting an element on the Taxonomy tree will search expressions in the CNL Editor that are related explicitly to the selected element.

Taxonomy tree is divided into four parts:

- **thing**: shows *is-a* relations between concepts and instances.
- **nothing**: shows concepts that cannot have instances.
- **relation**: shows hierarchy of relations between concepts and/or instances.
- **attribute**: shows hierarchy of attributes.

In Ontorion mode, the taxonomy tree will load the structure of the knowledge currently stored in Ontorion. Furthermore, in Ontorion mode, it is possible to make SPARQL queries to the knowledge in Ontorion.

Fig. 3. Screenshot of the window that will be shown to the user when some changes are detected in the current knowledge that the user is editing.

Annotation Support and References Ontorion and Fluent Editor have full support for annotations and reference management. In Ontorion, it is possible to store a prefix to namespace map that will be used to translate all namespaces contained in the entities to the corresponding prefix (e.g. label[rdf]). If no namespace is found, the entity will contain the full namespace (e.g. label[`<http://www.w3.org/1999/02/22-rdf-syntax-ns#>`]). Thus each time that the user downloads the knowledge, Fluent Editor will automatically download also the references and the annotations related to the sentences that have been downloaded. It is then possible to see the annotations in the annotation window.

It is also possible to see annotations reative to all the elements in the taxonomy tree by right clicking on one of the node and selecting the Show annotation command. In this case the annotation will be shown without the need to download the knowledge relative to this entity.

3.3 Module management

The modularization algorithm is a proprietary algorithm based on [3]. Essentially a module in Ontorion is the knowledge relative to the entities in the signature together with the knowledge "around" it. This means that when asking for the module of an entity in Ontorion, the user will get back all sentences where the entity is mentioned together will all the sentences that are related to this entity. Modularization is used internally by Ontorion to decide the part of the knowledge on which it needs to reason but at the same time it used in the collaborative editing of knowledge to shw the user only the knowledge he/she wants to work on.

In Fluent Editor, additional effort is done to manage module synchronization. In order to do this, we are distinguishing three modules: the current "local" module that the user is editing, the local remote module (the last module that the user downloaded) and the current remote module (the module that is currently stored in Ontorion). Using this distinction, we are able to identify the knowledge that has been added/removed locally to the module that was downloaded from

Ontorion. Furthermore in the background we are checking if the knowledge in Ontorion has changed. If it has we can, by comparing the remote module with the last downloaded module, understand if new knowledge has been added and warn the user.

A non trivial problem related to managing the module synchronization is given by managing time synchronization. In order to avoid problems related to managing a common clock and other time related problems in distributed systems, we opted for a simpler solution: knowledge versioning. Each time that a user makes a change to the knowledge base, the knowledge version changes. In this way, it is enough to download in Fluent Editor the knowledge version together with the module in order to know if the module we have downloaded is up to date or not.

4 Discussion and Summary

In this paper we presented a collaborative knowledge editing environment using Fluent Editor and the Ontorion Server. We have shown that using these tools it is possible to set up a collaborative knowledge editing environment where the user downloads the modules of knowledge that he/she is interested in and modifies it using OCNL language. Furthermore the user can explore the knowledge currently stored in Ontorion by using SPARQL queries and the taxonomy tree.

We are currently working in improving the editing capabilities of the environment by: add user access levels to each "module" of the knowledge (instead of the whole database), using knowledge modification logging to allow for undo functionalities and add versioning management.

References

1. Holger Knublauch, Matthew Horridge, Mark A Musen, Alan L Rector, Robert Stevens, Nick Drummond, Phillip W Lord, Natalya Fridman Noy, Julian Seidenberg, and Hai Wang. The protege owl experience. In *OWLED*, 2005.
2. Paweł Kapłanski and Paweł Weichbroth. Cognitum ontorion: Knowledge representation and reasoning system. In *to be pusblished in FEDCSIS2015*, 2015.
3. Pawel Kaplanski. Syntactic modular decomposition of large ontologies with relational database. In *New Challenges in Computational Collective Intelligence*, pages 65–72. Springer, 2009.
4. Rob Shearer, Boris Motik, and Ian Horrocks. Hermit: A highly-efficient owl reasoner. In *OWLED*, volume 432, page 91, 2008.
5. Paweł Kapłański. Controlled english interface for knowledge bases. *Studia Informatica*, 32(2A):485–494, 2011.