OntoEdit: Collaborative Ontology Development for the Semantic Web

York Sure[1], Michael Erdmann[2], Juergen Angele[2],
Steffen Staab[1,2], Rudi Studer[1,2,3], and Dirk Wenke[2]

[1] Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany
{sure,staab,studer}@aifb.uni-karlsruhe.de
http://www.aifb.uni-karlsruhe.de/WBS/
[2] Ontoprise GmbH, Haid-und-Neu-Str. 7, 76131 Karlsruhe, Germany,
{angele,erdmann,wenke}@ontoprise.de
http://www.ontoprise.de/
[3] FZI Research Center for Information Technologies,
Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany
http://www.fzi.de/wim/

Abstract. Ontologies now play an important role for enabling the semantic web. They provide a source of precisely defined terms e.g. for knowledge-intensive applications. The terms are used for concise communication across people and applications. Typically the development of ontologies involves collaborative efforts of multiple persons. OntoEdit is an ontology editor that integrates numerous aspects of ontology engineering. This paper focuses on collaborative development of ontologies with OntoEdit which is guided by a comprehensive methodology.

1 Introduction

The vision of the Semantic Web introduces the next generation of the Web by establishing a layer of machine-understandable data e.g. for software agents, sophisticated search engines and web services. Ontologies play an important role for these knowledge-intensive applications as a source of formally defined terms for communication. They aim at capturing domain knowledge in a generic way and provide a commonly agreed understanding of a domain, which may be reused, shared, and operationalized across applications and groups. However, because of the size of ontologies, their complexity, their formal underpinnings and the necessity to come towards a shared understanding within a group of people, ontologies are still far from being a commodity.

In recent years, research aimed at paving the way for the construction of ontologies by ontology development environments [DSW+99,NFM00,ACFLGP01]. Thereby, we have seen different directions taken to support the engineering of ontologies.

1. Several seminal proposals for guiding the ontology development process by engineering methodology have been described [UK95,LGPSS99], which influenced the ontology development environments [ACFLGP01].
2. Inferencing mechanisms for large ontologies have been developed and implemented (e.g. [Hor98]) — also to support ontology engineering [BHGS01].

I. Horrocks and J. Hendler (Eds.): ISWC 2002, LNCS 2342, pp. 221–235, 2002.
© Springer-Verlag Berlin Heidelberg 2002

3. Finally, the need to achieve consensus about an ontology was reflected by collaborative environments [TS98,Dom98,SPKR96,FFR96] for ontology engineering.

However, only few of these seminal approaches (e.g. [ACFLGP01]) have worked towards combining all of these urgent desiderata. This observation seems to reflect our own experience, viz. that it is far from trivial to offer a sound integration of these aspects. Therefore, OntoEdit is an ontology engineering environment that is rather unique in its kind as it combines methodology-based ontology development with capabilities for collaboration and inferencing. This paper is about how methodology and collaboration interact to support the ontology engineering process.[1]

Concerning the methodology, OntoEdit focuses on three main steps for ontology development (our methodology is based on [SAA+99], a detailed description can be found in [SSSS01]), viz. requirements specification, refinement, and evaluation. Firstly, all requirements of the envisaged ontology are collected. Typically for ontology engineering, ontology engineers and domain experts are joined in a team that works together on a description of domain and goal of the ontology, design guidelines, available knowledge sources (e.g. reusable ontologies and thesauri etc.), potential users and use cases and applications supported by the ontology. The output of this phase is a semi-formal description of the ontology. Secondly, during the refinement phase the team extends the semi-formal description in several iterations and formalizes it in an appropriate representation language. The output of this phase is a mature ontology (aka. "target ontology"). Thirdly, the target ontology needs to be evaluated according to the requirement specifications. Typically this phase serves as a proof for the usefulness of developed ontologies and may involve the engineering team as well as end users of the targeted application. The output of this phase is an evaluated ontology, ready for the roll-out into a productive environment.

Support for these collaborative development steps within the ontology development methodology is crucial in order to meet the conflicting needs for ease of use and construction of complex ontology structures. We will now introduce a case study, the development of an ontology for a semantic portal about our institute AIFB. Based on the process illustrated with this example, the remainder of the paper will be structured.

2 Case Study: SEmantic portAL (SEAL) at Institute AIFB

Based on our conceptual framework for "SEmantic portALs" (SEAL) we have developed an ontology based portal for our own institute AIFB[2]. The aim of the web application is the presentation of information to human and software agents taking advantage of semantic structures. The portal targets mainly students and researchers and presents typical information about persons, teaching-related topics and research-related topics. A detailed description of SEAL can be found in [MSS+02]. We briefly sketch the architectural idea of SEAL and then focus on describing aspects of the collaborative ontology development itself.

[1] In a recently submitted companion paper [SSA+02], we have described how methodology and inferencing interact to support the ontology engineering process.

[2] http://www.aifb.uni-karlsruhe.de

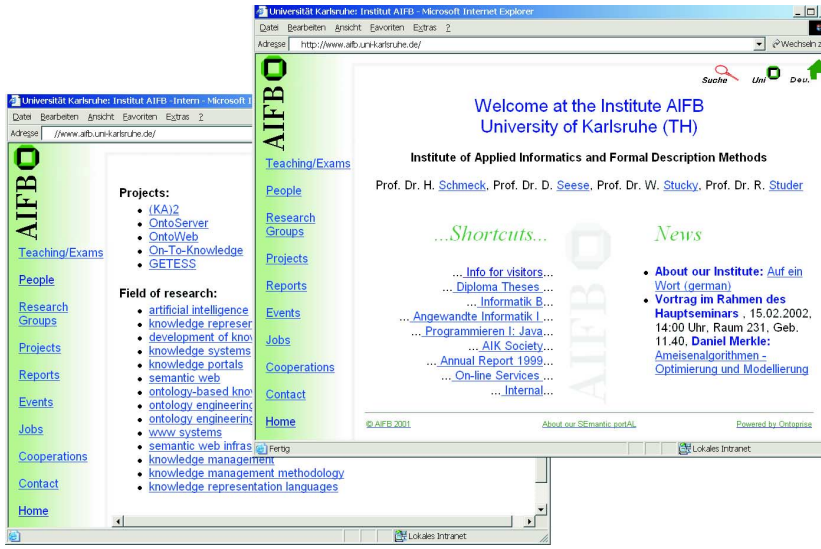

Fig. 1. The semantic portal at the institute AIFB

SEAL is a framework for managing community web sites and web portals based on ontologies. The ontology supports queries to multiple sources, but beyond that it also includes the intensive use of the schema information itself allowing for automatic generation of navigational views[3] and mixed ontology and content-based presentation. The core idea of SEAL is that Semantic Portals for a community of users that contribute *and* consume information require web site management *and* web information integration. In order to reduce engineering and maintenance efforts SEAL uses an ontology for semantic integration of existing data sources as well as for web site management and presentation to the outside world. SEAL exploits the ontology to offer mechanisms for acquiring, structuring and sharing information between human and/or machine agents.

For the AIFB ontology the ontology engineers had to deal mainly with (i) modeling navigational structures, (ii) the research topics investigated by different groups of the institute, (iii) topics related to teaching and last but not least (iv) personal information about members of the institute. Figure 1 shows two screenshots from our portal, viz. the entry point and parts of a researcher's homepage. On the left side of the screenshots the top-level navigational structure is shown. The right side of the researchers homepage shows projects this researcher is involved in and his personal research topics.

During the **requirements specification phase** (cf. Section 3) the ontology engineers set up the requirements specification for the ontology. To gather initial structures, they took themselves as an "input source" and collected a large set of research topics, topics related to teaching and personal information. Naturally, the ontology developers (viz. the knowledge management group) were not able to come up with all relevant structures by themselves and already in this phase they collaborated with domain experts, viz.

[3] Examples are navigation hierarchies that appear as `has-part` trees or `has-subtopic` trees in the ontology.

their colleagues from other research groups (information systems group, efficient algorithms group and complexity management group). In this section we show OntoEdit's support for capturing the requirements specification and his brainstorming support for the gathering of initial structures.

In the **refinement phase** (cf. Section 4) the "first draft" of the ontology was refined by structuring the concept hierarchy and addition of concepts, relationships and axioms. Like in the early stage, a tight collaboration of all groups was needed to refine the ontology, e.g. to formalize implicit knowledge like "someone who works in logic also works in theoretical computer science". In this section we present OntoEdit's advanced support for collaborative ontology engineering through transaction management.

Finally, in the **evaluation phase** (cf. Section 5) the ontology is evaluated according to the previously set up requirement specifications. In a first step, each group evaluated "their" requirements individually. In a second step the ontology as a whole was evaluated. In this section we illustrate OntoEdit's facilities for setting up test sets for evaluation, avoiding and locating errors in the ontology, using competency questions for evaluation and, last but not least, we describe how these components work together for support of collaborative evaluation.

3 Requirements Specification Phase

Like in software engineering and as proposed by [LGPSS99], we start ontology development with collecting requirements for the envisaged ontology. By nature this task is performed by a team of experts for the domain accompanied by experts for modeling. The outcome of this phase is (i) a document that contains all relevant requirement specifications (domain and goal of the ontology, design guidelines, available knowledge sources, potential users and use cases and applications supported by the ontology) (ii) a semi-formal ontology description, i.e. a graph of named nodes and (un-)named, (un-)directed edges, both of which may be linked with further descriptive text.

To operationalize a methodology it is desirable to have a tool that reflects and supports all steps of the methodology and guides users step by step through the ontology engineering process. Along with the development of the methodology we therefore extended the core functionalities of OntoEdit by two plug-ins to support first stages of the ontology development, viz. OntoKick and Mind2Onto[4].

OntoKick targets at (i) creation of the requirement specification document and (ii) extraction of relevant structures for the building of the semi-formal ontology description. Mind2Onto targets at integration of brainstorming processes to build relevant structures of the semi-formal ontology description. As computer science researchers we were familiar with software development and preferred to start with a requirement specification of the ontology, i.e. OntoKick. People who are not so familiar with software design principles often prefer to start with "doing something". Brain storming is a good method to quickly and intuitively start a project, therefore one also might begin the ontology development process with Mind2Onto.

[4] Describing the plug-in framework is beyond the scope of this paper, it is described in [Han01]. In a nutshell, one might easily expand OntoEdit's functionalities through plug-ins.

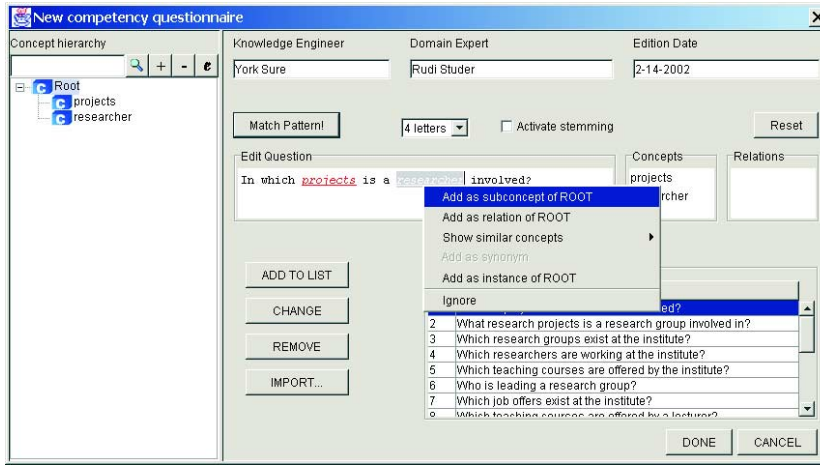

Fig. 2. OntoKick: Capturing of Competency Questions

OntoKick support for the collaborative generation of requirements specifications for ontologies. The collaborative aspect of OntoKick is not so much the support for distributed work of team members, but rather the support for personal interaction of ontology engineers and domain experts. This is a two step process. Firstly, OntoKick allows for describing important meta-aspects of the ontology, viz.: domain and the goal of the ontology, design guidelines, the available knowledge sources (e.g. domain experts, reusable ontologies etc.), potential users, use cases, and applications supported by the ontology. OntoKick guides the engineering team stepwise through all relevant aspects and stores these descriptions along with the ontology definitions.

Secondly, OntoKick supports the creation of a semi-formal ontology description. Naturally, domain experts are a valuable knowledge source for structuring a domain. A very common method for knowledge acquisition from domain experts are personal interviews. To structure the interviews with domain experts we use competency questions (CQ, cf. [UK95]). Each CQ defines a query that the envisaged ontology (respectively the ontology-based application) should be able to answer and therefore defines explicit requirements for the ontology. Typically, CQs are derived from interviews with domain experts and help to structure knowledge. We take further advantage by using them to create the initial version of the semi-formal description of an ontology and also for evaluation of the ontology in a later stage (cf. Section 5). Based on the assumption that each CQ contains valuable information about the domain of the ontology we extract relevant concepts, relations and instances (cf. Figure 2).

Figure 3 shows the main workbench of OntoEdit. It consists of several tabs, here the tab *Concepts & Relations* is selected. The left frame of the tab contains a concept hierarchy, the right frame contains relations[5] with their ranges that have the currently selected concept (here: Projekt) as domain. Further tabs include e.g. *Instances* for adding

[5] Please note that for exporting and importing of RDF/S one would prefer to talk about properties with domain and range restrictions instead of relations.

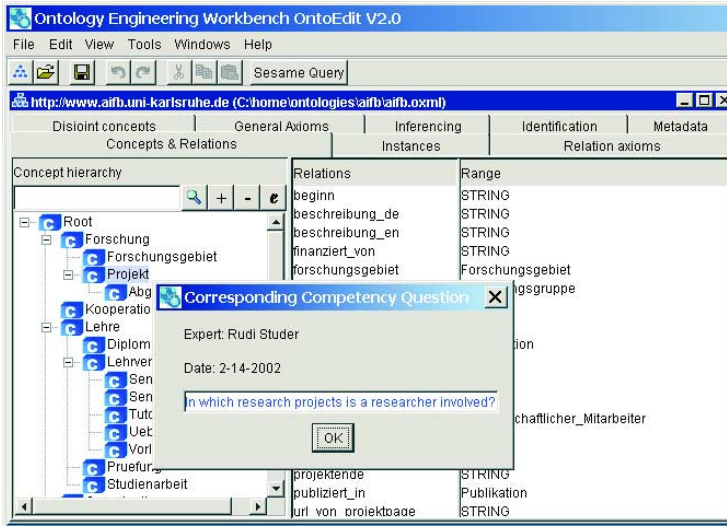

Fig. 3. OntoKick: Traceability of Competency Questions

and editing of instances or *Relation axioms* for adding and editing relational axioms like transitivity, symmetry and inverseness of relations (e.g.).

OntoKick establishes and maintains links between CQs and concepts derived from them. Figure 3 also includes the relevant CQ for the selected concept Projekt (provided by a right-click context menu). This allows for better traceability of the origins of concept definitions in later stages and improves quality assurance during the development process, i.e. by documenting the origins and the context of concepts, relations and instances. Therefore, a reached level of quality of the ontology can be reengineered by other ontology engineers.

"Real life" ontology modeling is supported by OntoKick as the following example illustrates with our SEAL scenario. First, the ontology engineer interviews an expert (i.e. his colleague). Thereby they identify CQs, e.g. "In which research projects is a researcher involved?" (cf. Figures 2 and 3). Based on these CQs the ontology engineer creates a first draft of the semi-formal description of the ontology and already graphically models it in OntoEdit. He identifies relevant concepts, relations and instances from of the above-mentioned CQ, e.g. the concept Projekte (German for projects). The instances might e.g. be prototypical instances that are used for evaluation purposes (cf. Section 5). After capturing CQs and modeling the ontology with OntoEdit the ontology engineer is able to retrieve corresponding CQs for each concept, relation, and instance. This helps him and others to identify the context in which these elements were modeled.

Mind2Onto is a plug-in for supporting brainstorming and discussion about ontology structures. Especially during early stages of projects in general, brainstorming methods are commonly used to quickly capture pieces of relevant knowledge. A widely used method are mind maps™ [Buz74], they were originally developed to support more efficient learning and evolved to a management technique used by numerous companies. In general, a mind map™ provides information about a topic that is structured in a tree.

Each branch of the tree is typically named and associatively refined by it's subbranches. Icons and pictures as well as different colors and fonts might be used for illustration based on the assumption that our memory performance is improved by visual aspects. There already exist numerous tools for the electronically creation of mind maps™. Many people from academia and industry are familiar with mind maps™ and related tools – including potential ontology engineers and domain experts. Therefore the integration of electronic mind maps™ into the ontology development process is very attractive (cf. e.g. [LS02]).

We relied on a widely used commercial tool[6] for the creation of mind maps™. It supports collaborative engineering of mind maps™ through peer-to-peer communication and has advanced facilities for graphical presentations of hierarchical structures, e.g. easy to use copy&paste functionalities and different highlighting mechanisms. It's strength but also it's weakness lies in the intuitive user interface and the simple but effective usability, which allows for quick creation of mind maps™ but lacks of expressiveness for advanced ontology modeling. By nature, mind maps™ have (almost) no assumptions for it's semantics, i.e. branches are somehow "associatively related" to each other. This assumption fits perfectly well during early stages of ontology development for quick and effective capturing of relevant knowledge pieces and makes the mind map™ tool a valuable add-on.

Mind2Onto integrates the mind map™ tool into the ontology engineering methodology. Currently OntoEdit and the mind map™ tool interoperate through import and export facilities based on XML (cf. Figure 4). In our scenario we used the mind map™ tool to facilitate discussions about the research topic hierarchy. Most of the domain experts were already familiar with the tool, the other ones learned it's usage very quickly. Initially all workgroups of our institute created in joint sessions a mind map™ of relevant research topics. The peer-to-peer communication of the mind map™ tool provided the necessary workgroup functionalities for this effort.

4 Refinement Phase

The goal of this phase is to refine the semi-formal description of the ontology according to the captured requirements into a mature ontology, which is the output of this phase. Especially during the refinement phase, several teams were working simultaneously with OntoEdit on developing the ontology for our AIFB portal. Several sessions were necessary until e.g. the navigational structures were accepted by all members. In this phase also numerous relationships were added to refine the ontology (a task that e.g. the brainstorming tool is not capable of). After reaching a consensus, the created mind map™ (see previous section) was restructured in cooperation with the ontology engineers to facilitate the interoperability with OntoEdit, i.e. as a simple assumption we took the elements of a mind map™ as concepts and branches as SUBCONCEPTOF relationships between concepts (cf. Figure 4). After several discussions we decided to model research topics as subconcepts of the concept Forschungsgebiet (German for ResearchTopic) to form a hierarchy of research topics. The final version of the research topic hierarchy then was included into the ontology.

[6] MindManager™ 2002 Business Edition, cf. http://www.mindjet.com

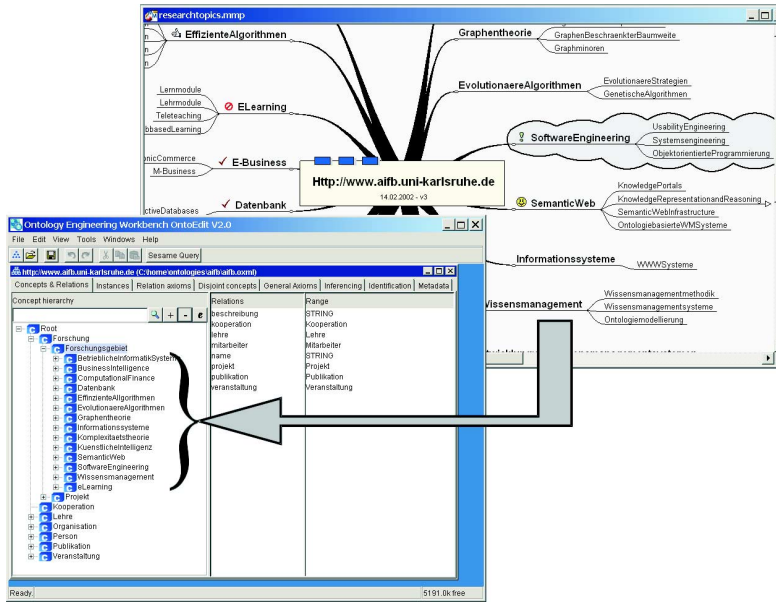

Fig. 4. Mind2Onto: Research topics as a mind map and in the ontology

In the current version of OntoEdit members of an engineering team can collaborate even though they are geographically distributed and still modify the ontology at the same time. We have developed a client/server architecture (cf. Figure 5) in which the clients connect to an ontology server and can change or extend the ontology. All clients are immediately informed of modifications the other ontologists do to the ontology. Engineers can store comments (e.g. explaining design decisions) in a documentation field for each concept and relation. By this way, one of the main features of ontologies, i.e. their consensual character, is supported. Collaborating ontologists must agree on the modeling decisions that are made. Therefore the possibility to monitor the development process of all collaborators is essential for reaching the goal of a shared ontology.

Transaction Management. In a distributed development environment certain mechanisms must be implemented to ensure safe development conditions, such as consistency of the models and the provision of a minimum degree of concurrency. To reach this goal we employed a locking and transaction protocol and implemented a distributed event model on the basis of Java-RMI (remote method invocation).

To guarantee consistent models the clients are forced to obtain locks for each resource (e.g. concept, instance, relation) that they want to modify (e.g. add a superconcept, add an attribute-value pair to an instance, or change the arity of a relation).[7] The server denies the (write-) access to a resource if the resource is not locked by the client that attempts

[7] The grounding datamodel of OntoEdit is OXML 2.0. This frame–based model offers a number of meta–classes, like *ontology*, *concept*, *relation*, but also *predicate* or *axiom*, with a rich set of properties and associations to facilitate ontology modeling: cf. http://www.ontoprise.de/download/oxml2.0.pdf for a reference manual.

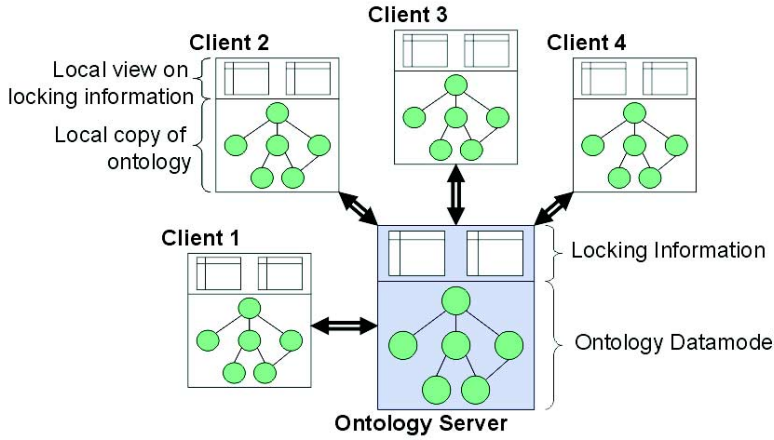

Fig. 5. Client/server architecture of OntoEdit

to modify it. Clients can obtain locks either by explicitly locking these resources, or more conveniently, by a begin of transaction (BOT) that is accompanied with a list of needed resources. If not all resources can be assigned to the calling client the BOT fails and the transaction is immediately aborted. Otherwise the server locks the needed resources for the client, so that no other client can manipulate them until the end of the transaction is reached. Now the client can manipulate the locked resources until it commits the transaction. After a commit all locked resources are freed again and the operations performed in the body of the transaction are actually applied to the datamodel. Afterwards, events are created to inform the other clients of the modifications performed. If the transaction needs to be aborted by the client all operations are undone, all locks are removed, and no events are fired.

Transactions may be nested to make complex operations possible without the need of rollback mechanisms. E.g. the datamodel procedure of moving a concept from one superconcept to another one consists of two subtransactions (remove a superconcept-relationship to the first superconcept and establish a new one for the second concept) that must be performed all together or none at all. Because of the necessity of nested transactions we implemented a strict two phase locking protocol (S2PL). In this protocol additional resources can be achieved (and locked) within the body of a transaction. Our implementation of the S2PL allows for arbitrarily nested transactions. The execution of inner transactions and the release of all locked resources is postponed until the outermost commit or abort is finally reached. Again, only after the final commit events are sent to the other clients. We employ the S2PL because (i) it allows for nested transactions and (ii) prevents cascading aborts. Thus, clients can be immediately informed if a planned operation will commit or is prohibited due to unavailable resources. (iii) S2PL prevents also deadlocks since resources are only locked in a BOT if all locks can be achieved. Other locking protocols are either too inflexible (like conservative locking (C2PL) that cannot lock resources in addition to the locks of the BOT and thus, is not suitable for nested transactions) or provide chances of deadlocks that must be appropriately handled.

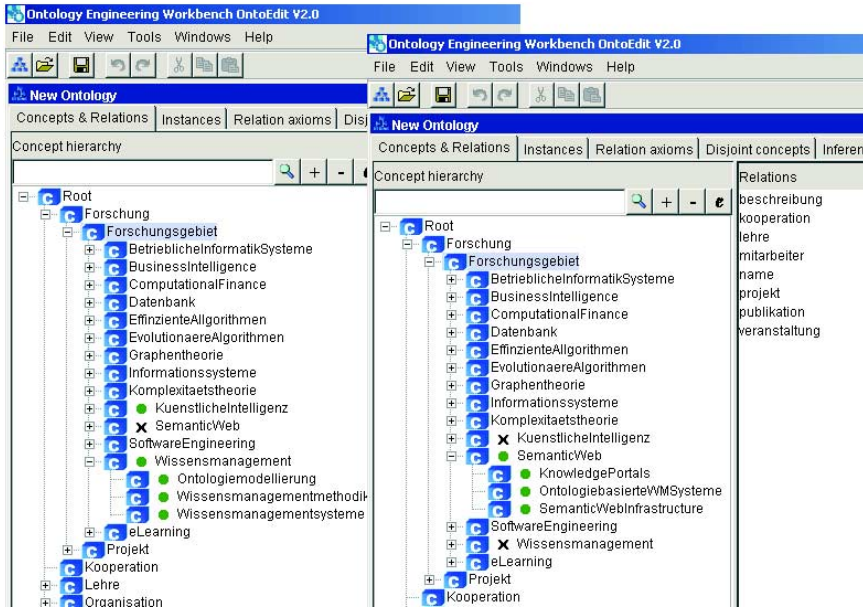

Fig. 6. Locked trees in OntoEdit

To reduce communication overhead, save bandwidth and because transactions are relatively short lived no information about transactions (esp. not about locked objects within a BOT) is communicated from the server to other clients, i.e. the local view on locking information within a client (cf. Figure 5) contains all resources that are locked by this client (by a BOT) but none that have been locked by a BOT of any other client. Nevertheless, another kind of locking information *is* distributed to all clients. An ontologist can lock a whole subtree of the concept hierarchy. The server informs all clients of this locking operation.

Locking Subtrees of the Concept Hierarchy. A common practice in ontology engineering is to start with a top level structure and to refine it later on. Different parts of an ontology can be refined by different ontologists or groups (cf. the research topics of the AIFB ontology). These collaborators should be able to work on their parts of the ontology with as few interference with other ontologists as possible. This is achieved in OntoEdit by the possibility of locking a complete subtree of the concept hierarchy. After the subtrees have been locked no conflicts can arise anymore, and what is equally important, the need to check for locking information *with the server* is reduced drastically. Since most modeling operations will occur within the scope of the subtrees, i.e. will mainly access already locked resources, the client can decide *locally* whether these operations are permitted or not.

This (tree-) locking information is distributed to all other clients and visually indicated in the GUI (cf. Figure 6). Crosses mark concepts that are locked by other clients and may not be edited. Bullets mark concepts that may be edited, altered and removed at

will. Concepts without additional icons are currently not locked and therefore available for locking by any user.

Due to the distribution of this information clients can often check locally whether a transaction will be permitted or not. If all needed resources are marked as "locked by me" in the local view on the locking information (cf. Figure 5) a BOT can be safely accepted. If at least one resource is marked as being locked by another client the current client can definitively reject a BOT (or a lockSubTree request). Only if resources are requested in a BOT for which no information is locally available, the server has to be consulted.

What Does Locking a Concept Mean? Locking resources in relational databases means the DB administrators or application developers must decide whether to lock an attribute, a tuple, or a complete table (i.e. relation). Since the basic datamodel for ontologies is much richer (esp. due to hierarchical relationships between concepts, between relations, and between instances and concepts) the decision of what a lock entails is more complex.

The most simple answer would be to lock the complete ontology with all its components. But this solution is ruled out since it would disallow any kind of concurrency and distributed collaboration. Another simple answer would be to lock the resources that are to be modified within a transaction, e.g. the resource X in the transaction that states that concept X has a superconcept Y. Apparently, for this transaction concept Y should also be locked since a new subconcept for Y is defined. Thus, the second simple approach seems to lock too few resources.

Due to hierarchical relationships between concepts locking a concept X implies *read-locks* for all super-concepts of X and all their super-concepts, recursively. A read-lock marks a resource as being read-only, i.e. modifications to it are currently disallowed. If a read-lock for at least one superconcept cannot be achieved X will not be locked and the BOT fails. Thus, no operations may modify X. Read-locks can be yielded to multiple clients at the same time without conflict. If a client is the only one that read-locked a resource the client can achieve a stricter (write-)lock. Other clients cannot.

The reason why a lock propagates from one resource to another in the ontology can be seen in the following example scenario: Assume, X is a subconcept of Y and Y has a slot A with range Y. Assume, we want to restrict the value range of A for X from Y to X. Thus, in the BOT we just lock the concept X and call the appropriate operation on X. Before we send the commit another client (after locking Y) changes the name of A to B and commits. If we now commit our transaction the semantics of the combined operations is not defined. Does X now have two independent attributes A and B? Or is attribute A totally lost as well as our newly defined range restriction? Both situations are unsatisfactory. Thus, to prevent them superconcepts need to be read-locked.

The same holds for locking complete subtrees of the concept hierarchy. Here all subconcepts are locked in the same way as the root of the subtree and all superconcepts of the root. All superconcepts of the subconcepts of the root must be read-locked. This is necessary only if multiple-inheritance is allowed. Because the same rules for computing super- and subobjects of concepts etc. are available in the client and in the server some decisions whether a transaction is allowed or not may be made on the client side without connecting to the server. Thus the amount of queries sent over the network is reduced and processing times are enhanced.

5 Evaluation Phase

The last step in ontology development is about evaluating the formal ontology. The goal of the evaluation phase is to check whether the ontology fulfills the requirements specified during the first stage of the methodology (cf. Section 3). OntoEdit tackles several aspects for evaluation, i.e. (i) test sets of instances and axioms can be used for the analysis of typical queries, (ii) a graphical axiom editor in combination with an underlying inference engine[8] allows for error avoidance and location, (iii) competency questions might be formalized into queries and evaluated by using the facilities of (i) and (ii) and, last but not least, (iv) a namespace mechanism allows for using the facilities (i) – (iii) collaboratively.

Analysis of Typical Queries. The ontology engineer may interactively construct and save instances and axioms. OntoEdit contains a simple instance editor and an axiom editor that the ontology engineer can use to create test sets. A test set can be automatically processed and checked for consistency. Once the ontology evolves and needs changes to remain up-to-date, a test set may be re-used for checking validity of the ontology. This basic functionality is e.g. needed during the usage of Competency Questions.

Error Avoidance and Location. While the generation and validation of test cases allows for detection of errors, it does not really support the localization of errors. The set of all axioms, class and instance definitions express sometimes complex relationships and axioms often interact with other axioms when processed. Thus it is frequently very difficult to overview the correctness of a set of axioms and detect the faulty ones. In order to avoid problems, OntoEdit allows for defining several standardized properties of relationships by clicking on the GUI (viz. symmetry, transitivity and inverseness of relations) and a graphical rule editor for other types of axioms. In order to locate problems, OntoEdit takes advantage of the underlying inference engine, which allows for introspection and also comes with a debugger. A very simple but effective method to test axioms with test cases is e.g. to switch off and switch on axioms. A more sophisticated approach uses visualizations of proof trees by tracking back the drawn inferences to the test instances. Therefore semantic errors in rules may be discovered. A more detailed description of OntoEdit facilities for the "analysis of typical queries" and "error avoidance and location" can be found in [SSA+02].

Usage of Competency Questions. Competency Questions may help in two ways for evaluation. Firstly, they might provide prototypical instances for a test set (see above). Secondly, CQs itself define requirements for the ontology (–based application), therefore they provide a checklist of questions the ontology should be able to answer. E.g. from the CQ "Who is head of a research workgroup?" the concept Workgroup and the relation HEADOFGROUP (with domain Researcher and range Workgroup) are identified as relevant elements and therefore modeled in the ontology. A prototypical instance is e.g. an instance of Researcher, viz. "Rudi Studer", who is HEADOFGROUP of an instance of Workgroup, viz. the "Knowledge Management Group". Each CQ may now be formalized with the facilities described above into a query which is executed by the inference engine. The query result may be used to check whether the requirements expressed by the CQs are fulfilled by the current ontology.

[8] The underlying inference engine used for processing of axioms is Ontobroker (cf. [DEFS99])

Collaborative Evaluation. The three facilities above can be used collaboratively through support from the backbone inference engine for the handling of multiple test sets. A namespace mechanism allows for syntactically splitting up ontologies or ontology parts (i.e. concepts, relations, instances and axioms) into modules that can be processed by a single instance or separate instances of the inference engine. Members of the engineering team usually have different requirements and use case scenarios, e.g. expressed by their different CQs, therefore they typically need separate test sets for evaluation. In a two step approach we (i) evaluate locally each test set, i.e. each member (or e.g. each pair of ontology engineer and domain expert) evaluates his CQs, and (ii) evaluate globally the conjunction of test sets.

6 Related Work

A good overview, viz. a comparative study of existing tools up to 1999, is given in [DSW+99]. Typically the internal knowledge model of ontology engineering environments is capable of deriving is-a hierarchies of concepts and attached relations. On top of that we provide facilities for axiom modeling and debugging. Naturally, it could not fully consider the more recent developments, e.g. Protégé [NFM00], and WebODE [ACFLGP01].

WebODE has a well-known methodological backbone, viz. METHONTOLOGY, and is designed to integrate numerous aspects of an ontology lifecycle. [ACFLGP01] mentions that it offers inferencing services (developed in Prolog) and an axiom manager (providing functionalities such as an axiom library, axiom patterns and axiom parsing and verification), but the very brief mentioning of these functionalities is too short to assess precisely. About collaboration, it is said that this is supported at the knowledge level, but how this is achieved remains open.

Environments like **Protégé** [NFM00] or **Chimaera** [MFRW00] offer sophisticated support for ontology engineering and merging of ontologies. Protégé also has a modular plug-in design rational like OntoEdit, but lacks of sophisticated support for collaborative engineering. They provide limited methodological and collaborative support for ontology engineering. A system well-known for it's reasoning support is **OilEd** [BHGS01] in combination with the description logics (DL) reasoner **FaCT** [BHGS01]. Their collaborative and methodological support is rather weak.

Some tools explicitly support collaboration during ontology engineering. **APECKS** [TS98] is targeted mainly for use by domain experts, possibly in the absence of a knowledge engineer, and its aim is to foster and support debate about domain ontologies. It does not enforce consistency nor correctness, and instead allows different conceptualisations of a domain to coexist. **Tadzebao** [Dom98] supports argument between users on the ontology design, using text, GIF images and even hand drawn sketches. The strength of these approaches lies in the advanced support for communication. In contrast we provide a more sophisticated support for fine-granular locking of the ontology.

The web-based **Ontosaurus** [SPKR96] combines support for collaboration with reasoning and allow individuals to add to an ontology only when consistency is retained within the ontology as a whole. This approach takes advantage of the underlying representation language LOOM's reasoning abilities and consistency checking. Ontosaurus

was inspired by the **Ontolingua** system [FFR96], which does not integrate an inferencing support as integral part of the ontology development environment. Due to the simple "state-less" HTML interaction, both systems have several limitations. E.g. does a server not maintain any state information about users, i.e. clients. Nor is it possible for a server to initiate an interaction on its own, e.g. alerting users to simultaneous changes by others. In general, no other approach is known to us that implemented fine-granular locking of ontologies like we do.

7 Conclusion

In this paper we presented (i) a motivational scenario, viz. the development of an ontology for the semantic portal of our institute, (ii) the methodology for ontology development that was applied in the scenario, (iii) the advanced collaborative tool support of OntoEdit for each step of the methodology. OntoEdit also has some features that could not be presented here, e.g. an extremely capable plug-in structure, a lexicon component, and an ontology mapping plug-in.

For the future, OntoEdit is planned to be developed in several directions: (i) The collaborative facilities will be further expanded, e.g. by adding a rights- and user-management layer on top of the locking mechanism and integrating communication and workgroup facilities (ii) new im- and exports will be developed, (ii) the integration of ontology construction with requirements specification documents will be generalized by means of semantic document annotation and (iv) the mind map™ tool will be integrated tighter into the ontology engineering process, e.g. through enabling direct communication between the tool and an ontology server, to name but a few.

Acknowledgements

Research for this paper was partially funded by EU in the project IST-1999-10132 "On-To-Knowledge".

References

ACFLGP01. J.C. Arprez, O. Corcho, M. Fernandez-Lopez, and A. Gomez-Perez. WebODE: a scalable workbench for ontological engineering. In *Proceedings of the First International Conference on Knowledge Capture (K-CAP) Oct. 21-23, 2001, Victoria, B.C., Canada*, 2001.

BHGS01. S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: A reason-able ontology editor for the semantic web. In *KI-2001: Advances in Artificial Intelligence*, LNAI 2174, pages 396–408. Springer, 2001.

Buz74. T. Buzan. *Use your head*. BBC Books, 1974.

DEFS99. S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology based access to distributed and semi-structured information. In R. Meersman et al., editor, *Database Semantics: Semantic Issues in Multimedia Systems*. Kluwer Academic, 1999.

Dom98. J. Domingue. Tadzebao and WebOnto: Discussing, browsing, and editing ontologies on the web. In *Proceedings of the 11th Knowledge Acquisition for Knowledge-Based Systems Workshop, April 18th-23rd. Banff, Canada*, 1998.

DSW+99. A. J. Duineveld, R. Stoter, M. R. Weiden, B. Kenepa, and V. R. Benjamins. WonderTools? A comparative study of ontological engineering tools. In *Proc. of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management. Banff, Alberta, Canada. October 16-21, 1999*, 1999.

FFR96. A. Farquhar, R. Fickas, and J. Rice. The Ontolingua Server: A tool for collaborative ontology construction. In *Proceedings of the 10th Banff Knowledge Acquisition for KnowledgeBased System Workshop (KAW'95)*, Banff, Canada, November 1996.

Han01. Siegfried Handschuh. Ontoplugins – a flexible component framework. Technical report, University of Karlsruhe, May 2001.

Hor98. I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proceedings of KR 1998*, pages 636–649. Morgan Kaufmann, 1998.

LGPSS99. M. F. Lopez, A. Gomez-Perez, J. P. Sierra, and A. P. Sierra. Building a chemical ontology using Methontology and the Ontology Design Environment. *Intelligent Systems*, 14(1), January/February 1999.

LS02. T. Lau and Y. Sure. Introducing ontology-based skills management at a large insurance company. In *Proceedings of the Modellierung 2002*, Tutzing, Germany, March 2002.

MFRW00. D. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *Proceedings of KR 2000*, pages 483–493. Morgan Kaufmann, 2000.

MSS+02. A. Maedche, S. Staab, R. Studer, Y. Sure, and R. Volz. SEAL – Tying up information integration and web site management by ontologies. *IEEE-CS Data Engineering Bulletin, Special Issue on Organizing and Discovering the Semantic Web*, March 2002. To appear.

NFM00. N. Fridman Noy, R. Fergerson, and M. Musen. The knowledge model of Protégé-2000: Combining interoperability and flexibility. In *Proceedings of EKAW 2000*, LNCS 1937, pages 17–32. Springer, 2000.

SAA+99. G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. Van de Velde, and B. Wielinga. *Knowledge Engineering and Management — The CommonKADS Methodology*. The MIT Press, Cambridge, Massachusetts; London, England, 1999.

SPKR96. B. Swartout, R. Patil, K. Knight, and T. Russ. Toward distributed use of large-scale ontologies. In *Proceedings of the 10th Knowledge Acquisition Workshop (KAW'96)*, Banff, Canada, November 1996.

SSA+02. Y. Sure, S. Staab, J. Angele, D. Wenke, and A. Maedche. OntoEdit: Guiding ontology development by methodology and inferencing. In *Submitted to: Prestigious Applications of Intelligent Systems (PAIS), in conjunction with ECAI 2002, July 21-26 2002, Lyon, France*, 2002.

SSSS01. S. Staab, H.-P. Schnurr, R. Studer, and Y. Sure. Knowledge processes and ontologies. *IEEE Intelligent Systems, Special Issue on Knowledge Management*, 16(1), January/Febrary 2001.

TS98. J. Tennison and N. Shadbolt. APECKS: A tool to support living ontologies. In *Proceedings of the 11th Knowledge Acquisition Workshop (KAW'98)*, Banff, Canada, April 1998.

UK95. M. Uschold and M. King. Towards a methodology for building ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95*, Montreal, Canada, 1995.