OntoEdit empowering SWAP: A case study in supporting DIstributed, Loosely-controlled and evolvInG Engineering of oNTologies (DILIGENT)

Sofia Pinto[1 2], Steffen Staab[2], York Sure[2], and Christoph Tempich[2]

[1]Dep. de Engenharia Informática, Instituto Superior Técnico, Lisboa, Portugal
http://www.dei.ist.utl.pt/
sofia.pinto@dei.ist.utl.pt
[2]Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany
http://www.aifb.uni-karlsruhe.de/WBS/
{staab,sure,tempich}@aifb.uni-karlsruhe.de

Abstract. Knowledge management solutions relying on central repositories sometimes have not met expectations, since users often create knowledge ad-hoc using their individual vocabulary and using their own decentral IT infrastructure (e.g., their laptop). To improve knowledge management for such decentralized and individualized knowledge work, it is necessary to, first, provide a corresponding IT infrastructure and to, second, deal with the harmonization of different vocabularies/ontologies. In this paper, we briefly sketch the technical peer-to-peer platform that we have built, but then we focus on the harmonization of the participating ontologies.
Thereby, the objective of this harmonization is to avoid the worst incongruencies by having users share a core ontology that they can expand for local use at their will and individual needs. The task that then needs to be solved is one of distributed, loosely-controlled and evolving engineering of ontologies. We have performed along these lines. To support the ontology engineering process in the case study we have furthermore extended the existing ontology engineering environment, OntoEdit. The case study process and the extended tool are presented in this paper.

1 Introduction

The knowledge structures underlying today's knowledge management systems constitute a kind of ontology that may be built according to established methodologies *e.g.* the one by [1]. These methodologies have a centralized approach towards engineering knowledge structures requiring *knowledge engineers*, *domain experts* and others to perform various tasks such as *requirement analysis* and *interviews*. While the user group of such an ontology may be huge, the development itself is performed by a — comparatively — small group of domain experts who *represent* the user community and ontology engineers who *help structuring*.

In Virtual Organizations [2], organizational structures change very often, since organizations frequently leave or join a network. Therefore, working based on traditional, centralized knowledge management systems becomes infeasible. While there are some

technical solutions toward Peer-to-Peer knowledge management systems (e.g., [3]) — and we have developed a technically sophisticated solution of our own as part of our project, SWAP — Semantic Web and Peer-to-Peer [4], traditional methodologies for creating and maintaining knowledge structures appear to become unusable like the systems they had been developed for in the first place.

Therefore, we postulate that ontology engineering must take place in a Distributed, evolvInG and Loosely-controlled setting. With DILIGENT we here provide a process template suitable for distributed engineering of knowledge structures that we plan to extend towards a fully worked out and multiply tested methodology in the long run. We here show a case study we have performed in the project SWAP using DILIGENT with a virtual organization. DILIGENT comprises five main activities of ontology engineering: **build**, **local adaptation**, **analysis**, **revision**, and **local update** (cf. Section 3).

The case study (cf. Section 4) suggests that the resulting ontology is indeed shared among users, that it adapts fast to new needs and is quickly engineered. With some loose control we could ensure that the core ontology remained consistent, though we do not claim that it gives a complete view on all the different organizations.

In the following, we briefly introduce the organizational and technical setting of our case study (Section 2). Then we sketch the DILIGENT process template (Section 3), before we describe the case study (Section 4).

2 Problem Setting

2.1 Organizational setting at IBIT case study

In the SWAP project, one of the case studies is in the tourism domain of the Balearic Islands. The needs of the tourism industry there, which is for 80% of the islands' economy, are best described by the term 'coopetition'. On the one hand the different organizations *compete* for customers against each other. On the other hand, they must *cooperate* in order to provide high quality for regional issues like infrastructure, facilities, clean environment, or safety — that are critical for them to be able to compete against other tourism destinations.

To collaborate on regional issues a number of organizations now collect and share information about *indicators* reflecting the impact of growing population and tourist fluxes in the islands, their environment and their infrastructures. Moreover, these indicators can be used to make predictions and help planning. For instance, organizations that require *Quality & Hospitality management* use the information to better plan, *e.g.*, their marketing campaigns. As another example, the governmental agency IBIT[1], the Balearic Government's co-ordination center of telematics, provides the local industry with information about *new technologies* that can help the tourism industry to better perform their tasks.

Due to the different working areas and objectives of the collaborating organizations, it proved impossible to set up a centralized knowledge management system or even a centralized ontology. They asked explicitly for a system without a central server, where

[1] `http://www.ibit.org`

knowledge sharing is integrated into the normal work, but where very different kinds of information could be shared with others.

To this end the SWAP consortium — including us at Univ. of Karlsruhe, IBIT, Free Univ. Amsterdam, Meta4, and empolis — have been developing the SWAP generic platform and we have built a concrete application on top that allows for satisficing the information sharing needs just elaborated.

2.2 Technical setting at SWAP

The SWAP platform (Semantic Web And Peer-to-peer; short Swapster) [4] is a generic infrastructure, which was designed to enable knowledge sharing in a distributed network. Nodes wrap knowledge from their local sources (files, e-mails, etc.). Nodes ask for and retrieve knowledge from their peers. For communicating knowledge, Swapster transmits RDF structures [5], which are used to convey conceptual structures (e.g., the definition of what a conference is) as well as corresponding data (e.g., data about ESWS-2004). For structured queries as well as for keyword queries, Swapster uses SeRQL, an SQL-like query language that allows for queries combining the conceptual and the data level and for returning newly constructed RDF-structures.

In the following we describe only the SWAPSTER components that we refer to later in this document (for more see [4]).

Knowledge Sources: Peers may have local sources of information such as the local file system, e-mail directories, local databases or bookmark lists. These local information sources represent the peer's body of knowledge as well as its basic vocabulary. These sources of information are the place where a peer can physically store information (documents, web pages) to be shared on the network.

Knowledge Source Integrator: The Knowledge Source Integrator is responsible for the extraction and integration of internal and external knowledge sources into the Local Node Repository. This task comprises (1) means to access local knowledge sources and extract an RDF(S) representation of the stored knowledge, (2) the selection of the RDF statements to be integrated into the Local Node Repository and (3) the annotation of the statements with metadata. These processes utilize the SWAP metadata model presented later in this section.

Local Node Repository: The local node repository stores all information and its meta information a peer wants to share with remote peers. It allows for query processing and view building. The repository is implemented on top of Sesame [6].

User Interface: The User Interface of the peer provides individual views on the information available in local sources as well as on information on the network. The views can be implemented using different visualization techniques (topic hierarchies, thematic maps, etc). The *Edit* component described here is realized as a plug-in of the OntoEdit ontology engineering environment.

Communication Adapter: This component is responsible for the network communication between peers. Our current implementation of the Communication Adapter is build on the JXTA framework [7].

Information and Meta-information. Information is represented as RDF(S) statements in the repository. The SWAP meta model[2] (*cf.* [4]) provides meta-information about the statements in the local node repository in order to memorize where the statements came from and other meta-information. The SWAP meta model consists of two RDFS classes, namely Swabbi and Peer. Every resource is related to an instance of Swabbi in order to describe from which instances of Peer it came from, etc.

Besides the SWAP meta data model the SWAP environment builds on the SWAP common ontology.[3] The SWAP common model defines concepts for *e.g.* File and Folder. Purpose of these classes is to provide a common model for information usually found on a peer participating in a knowledge management network.

Querying for Data. SeRQL[8] is an SQL like RDF query language comparable to *e.g.* RQL [9]. The main feature of SeRQL that goes beyond the abilities of existing languages is the ability to define structured output in terms of an RDF graph that does not necessarily coincide with the model that has been queried. This feature is essential for defining personalized views in the repository of a SWAP peer.

OntoEdit. [10] is an ontology engineering environment which allows for inspecting, browsing, codifying and modifying ontologies. Modelling ontologies using OntoEdit means modelling at a conceptual level, *viz.* (i) as much as possible independent of a concrete representation language, (ii) using graphical user interfaces (GUI) to represent views on conceptual structures, *i.e.* concepts ordered in a concept hierarchy, relations with domain and range, instances and axioms, rather than codifying conceptual structures in ASCII.

3 DILIGENT process

3.1 Process overview

As we have described before, decentralized cases of knowledge sharing, like our example of a virtual organization, require an ontology engineering process that reflects this particular organizational setting [11].[4] Therefore, we have drafted the template of such a process — we cannot claim that it is a full-fledged methodology yet. The result, which we call DILIGENT, is described in the following. In particular, we elaborate on the high-level process, the dominating roles and the functions of DILIGENT, before we go through the detailed steps in Sections 3.2. Subsequently, we give the concrete case in Section 4 as an indicator for the validity of our ontology engineering process design.

Key roles: In DILIGENT there are several experts, with different and complementary skills, involved in collaboratively building the same ontology. In a virtual organization they often belong to competing organizations and are geographically dispersed. Ontology builders may or may not use the ontology. Vice versa, most ontology users will typically not build or modify the given ontology.

Overall process: An initial ontology is made available and users are free to use it and modify it locally for their own purposes. There is a central board that maintains and

[2] `http://swap.semanticweb.org/2003/01/swap-peer#`

[3] `http://swap.semanticweb.org/2003/01/swap-common#`

[4] In fact, we conjecture that the majority of knowledge sharing cases falls into this category.

assures the quality of the shared core ontology. This central board is also responsible for deciding to do updates to the core ontology. However, updates are mostly based on changes re-occurring at and requests by *decentral*ly working users. Therefore the board only *loosely controls* the process. Due to the changes introduced by the users over time and the on-going integration of changes by the board, the ontology *evolves*. Let us now survey the DILIGENT process at the next finer level of granularity. DILIGENT comprises five main steps: (1) **build**, (2) **local adaptation**, (3) **analysis**, (4) **revision**, (5) **local update** (*cf.* Figure 1).

Build. The process starts by having *domain experts*, *users*, *knowledge engineers* and *ontology engineers* **build** an initial ontology. In contrast to existing ontology engineering methodologies (cf. [12–16]), we do not require completeness of the initial shared ontology with respect to the domain. The team involved in building the initial ontology should be relatively small, in order to more easily find a small and consensual first version of the shared ontology.

Local adaptation. Once the core ontology is available, users work with it and, in particular, adapt it to their local needs. Typically, they will have their own business requirements and correspondingly evolve their local ontologies (including the common core) [17, 18]. In their local environment, they are also free to change the reused core ontology. However, they are not allowed to directly change the core ontology from which other users copy to their local repository. Logging local adaptations (either permanently or at control points), the control board collects change requests to the shared ontology.

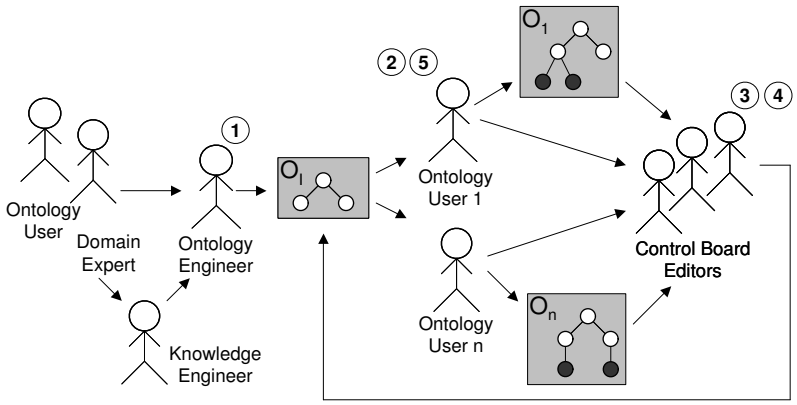

Fig. 1. Roles and functions in distributed ontology engineering

Analysis. The board **analyzes** the local ontologies and the requests and tries to identify similarities in users' ontologies. Since not all of the changes introduced or requested by the users will be introduced to the shared core ontology,[5] a crucial activity of the board is deciding which changes are going to be introduced in the next version of the shared ontology. The input from users provides the necessary arguments to underline change

[5] The idea in this kind of development is not to merge all user ontologies.

requests. A balanced decision that takes into account the different needs of the users and meets user's evolving requirements[6] has to be found.

Revise. The board should regularly **revise** the shared ontology, so that local ontologies do not diverge too far from the shared ontology. Therefore, the board should have a well-balanced and representative participation of the different kinds of participants involved in the process: knowledge providers, domain experts, ontology engineers and users. In this case, users are involved in ontology development, at least through their requests and re-occurring improvements and by evaluating it, mostly from an usability point of view. Knowledge providers in the board are responsible for evaluating the ontology, mostly from a technical and domain point of view. Ontology engineers are one of the major players in the analysis of arguments and in balancing them from a technical point of view. Another possible task for the controlling board, that may not always be a requirement, is to assure some compatibility with previous versions. Revision can be regarded as a kind of ontology development guided by a carefully balanced subset of evolving user driven requirements. Ontology engineers are responsible for updating the ontology, based on the decisions of the board. Revision of the shared ontology entails its evolution.

Local update. Once a new version of the shared ontology is released, users can **update** their own **local** ontologies to better use the knowledge represented in the new version. Even if the differences are small, users may rather reuse *e.g.* the new concepts instead of using their previously locally defined concepts that correspond to the new concepts represented in the new version.

3.2 Tool support for DILIGENT steps

We support the participants in the DILIGENT process with a tool (*cf.* Figure 2). It is an implementation of the *Edit* component of the SWAP environment, thus it works on the information stored in the local node repository, and is realized as an OntoEdit plug-in. We will now describe in detail how the tool supports the actions **building**, **locally adapting**, **analyzing**, **revising** and **locally updating**.

Build

The first step of the ontology engineering task is covered by established methodologies and by common OntoEdit functions. Some major tool functionality includes support for knowledge elicitation from domain experts by means of competency questions and mind maps and further support for the refinement process.

In contrast to a common full ontology engineering cycle the objective of this Build task is not to generate a complete and evaluated ontology but rather to *quickly* identify and formalize the main concepts and main relations.

[6] This is actually one of the trends in modern software engineering methodologies (see Rational Unified Process).

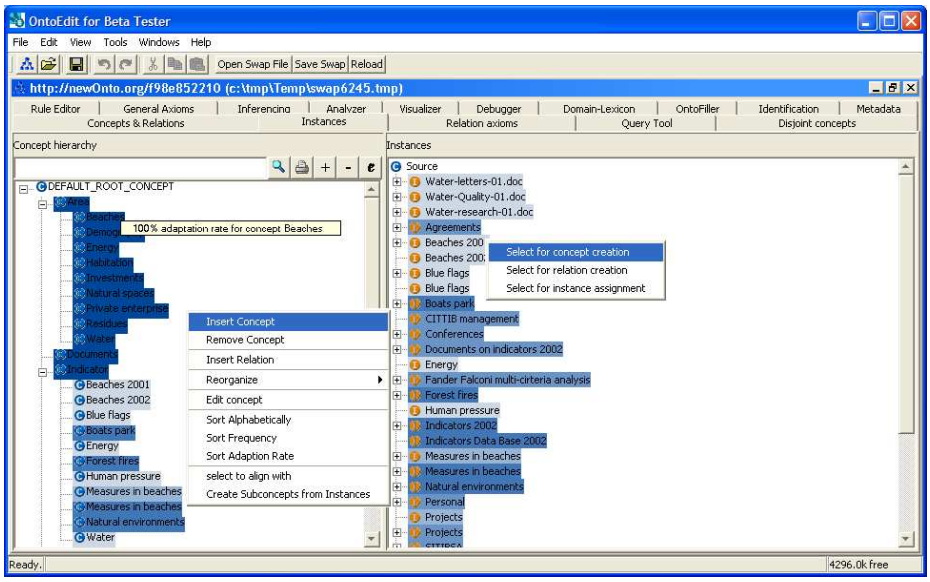

Fig. 2. OntoEdit plug-in to support DILIGENT

Local Adaptation

We distinguish two main types of users. The less frequent type is the user with ontology engineering competence who analyzes his personal needs, conceptualizes and formalizes them. He uses established ontological guidelines [19] in order to maintain soundness and validity. Besides, he annotates his knowledge according to his locally extended ontology.

The more common type of user reuses the categorizations he had defined in his daily work before (*e.g.*his folder structures) and just aligns them with the shared ontology. To illustrate this use case we must point forward to some issues we found in the case study. In the case study, users expect from a peer-to-peer system primarily the possibility to share their documents with others. Users already organize their files in folder structures according to their individual views. Hence, they will extend the core ontology with concepts and relations corresponding to folder structures found in their file or email system.

Concept creation. Our tool supports the creation of concepts and thus the extension of the shared ontology in two ways. The reader may note that both methods have been heavily influenced by our targeted system, SWAPSTER, and may be supplemented or overridden by other methods for other target systems:

1. OntoScrape — part of the SWAPSTER knowledge source integrator — can extract information from the user's local file and email system. OntoScrape extracts *e.g.*the folder hierarchy and builds up an RDFS representation in which the folder names are used to create instances of class Folder. This information is stored in the local

node repository. Then, the user can pick a set of instances of Folder and create concepts or relations using the folder names. In case of "concept creation" he would select a certain concept and the system would subclass that concept using the names of the previously selected folders.
The user may also reuse the folder hierarchy given by the inFolder relation to construct a subClassOf hierarchy.

2. Furthermore, a user can query other participants for their local subconcepts of the core ontology. He can use the gathered information to directly extend his own structures by integrating retrieved information. Alternatively, he may use the query result only for inspiration and create own extensions and modifications.
SWAPSTER integrates a component for semi-automatic alignment. Alignment detection is based on similarities between concepts and relations(cf., *e.g.*, [20]). The user may either select a set of classes and ask for proposed alignment for these classes, or he can look for alignments for the entire class hierarchy. The reader may note that even the best available alignment methods are not very accurate and hence some user involvement is required for aligning ontologies.

We are well aware of the drawbacks of this approach since the created structures will not be "clean" ontologies. However, as our case study indicates the created structures are good enough to be a fair input for the revision phase.

Instance assignment. Besides instances of the created concepts the user has mainly instances of concept Source *e.g.*Folder and File and wants to relate them to his concepts. In particular, documents play a predominant role in our case study. Since the global ontology certainly differs from existing local structures, we face the typical bootstrapping problem that the documents need to be aligned with the defined concepts. Our tool offers two possibilities to facilitate the assignment of documents to classes.

Manual Assignment Instances of concept Source can manually be selected and assigned to any concept in the ontology.

Automatic Assignment Automatic text classification is nowadays very effective. Hence we provide an interface for classifiers to suggest document classifications. Classifier training can take place remotely for the core ontology or according to established procedures [21]. The classifier has to produce a set of RDFS statements, stating which files should be classified where in the concept hierarchy. This has not been implemented yet.

Analyzing

As described in the methodology, the board will come together in fixed time lines or when a certain threshold of change requests has been reached. They will subsequently analyze the activities which have taken place. They will gather the ontologies from all participating peers on one central peer. The main task of the board is to incorporate the change requests into the core ontology and to identify common usage patterns. Our tool supports the board members in different ways to fulfill their task.

View selection. The number of newly created concepts within the peer network can be large. The board members can use queries to select only parts of the ontology to be visualized. Instead of loading the entire local node repository, a SeRQL query can be used

to generate a view on the repository. Queries can be defined manually, or predefined ones — visualizing certain branches of the ontology — can be selected.

Colors. The board needs to separate extensions made by different users and is interested in their relative activity. Since each peer uses its own name space to create URIs, extensions to the core made by different peers can be distinguished. The tool highlights the concepts, relations and instances of different peers by changing their background color. The saturation and brightness of the color indicates the number of concepts coming from a particular peer.[7] White is preserved for name spaces which the users can chose not to highlight (*e.g.* the local, swap-peer and swap-common name space are excluded from highlighting by default).

Adaptation rate. The averaged adaptation rate[8] of concepts from the core ontology and also of concepts from different users is an indicator of how well a concept fits the user needs. If a concept of the core ontology was not accepted by the users it probably has to be changed. Alternatively, a concept introduced by a user which has been reused by many other users can easily be integrated into the core ontology. The adaptation rate is visualized as a tool tip. In our case study *e.g.* the concept beaches was adapted by all users. It is calculated from the information stored in the SWAP data model.

Visualizing alignments. Instead of reusing concepts from other users, they can align them. The semantics of both actions is very similar. However, alignment implies, in most cases, a different label for the concept, which is determined by the board.

Sorting. To facilitate the analysis process, concepts, relations and instances may be sorted alphabetically, according to their adaptation rate or the peer activity. Concepts with the same label, but from different peers can be identified. Equally the concepts reused by most peers may be recognized.

Revision

The analysis is followed by the revision of the core ontology. The change requests as well as the recognized common usage patterns are integrated. In a traditional scenario the knowledge engineer introduces the new concepts and relations or changes the existing ones while the system meets the requirements described in [18]. The ontology changes must be resolved taking into account that the consistency of the underlying ontology and all dependent artifacts are preserved and may be supervised.

Additionally we require, that the reasons for any change do not require too much effort from the individual user. In particular, changes to the core ontology made because of overarching commonalities should be easy to integrate for users who created the concepts in the first place.

Local update

The changes to the core ontology must be propagated to all peers afterwards. The list of changes is transmitted to the different peers by the *Advertisement* component. Maedche

[7] Brighter and less saturated means less concepts than darker and more saturated.

[8] The adaptation rate of a concept indicates how many users have included the concept into their local ontology: $\text{adaptation rate} := \frac{\text{No of participant who have locally included the concept}}{\text{No of participants}}$

et al. describes in [22] the necessary infrastructure to enable consistent change propagation in a distributed environment. We do not require that all users adapt their ontology to the changes introduces by the board members. Furthermore, we allow that they use different evolution strategies when they accept changes (see [18] for an overview of different strategies).

After the *local update* took place the iteration continues with *local adaptation*. During the next *analysis* step the board will review which changes were actually accepted by the users.

4 Case study

We are now going to describe how DILIGENT ontology engineering is taking place in the IBIT case study and how OntoEdit is supporting it.

In the case study one organization with seven peers took part. The case study lasted for two weeks. The case study will be extended in the future to four organizations corresponding to 21 peers and it is expected that the total number of organizations will grow to 7 corresponding to 28 peers.

Building. In the IBIT case study two knowledge engineers were involved in building the first version of the shared ontology with the help of two ontology engineers. In this case, the knowledge engineers were at the same time also knowledge providers. In addition they received additional training such that later, when the P2P network is going to be up and running on a bigger scale, they will be able to act as ontology engineers on the board. This they did already during this study — together with two two experts from the domain area.

The ontology engineering process started by identifying the main concepts of the ontology through the analysis of competency questions and their answers. The most frequent queries and answers exchanged by peers were analyzed. The identified concepts were divided into three main modules: "Sustainable Development Indicators", "New Technologies" and "Quality&Hospitality Management". From the competency questions we quickly derived a first ontology with 22 concepts and 7 relations for the "Sustainable Development Indicator" ontology. This was the domain of the then participating organizations. The other modules will be further elaborated in future efforts.

Based on previous experience of IBIT with the participants we could expect that users would mainly specialize the modules of the shared ontology corresponding to their domain of expertise and work. Thus, it was decided by the ontology engineers and knowledge providers involved in building the initial version that the shared ontology should only evolve by addition of new concepts, and not from other more sophisticated operations, such as restructuring or deletion of concepts.

Local Adaptation. The developed core ontology for "Sustainable Development Indicator" was distributed among the users and they were asked to extend it with their local structures. With assistance of the developers they extracted on average 14 folders. The users mainly created sub concepts of concepts in the core ontology from the folder names. In other cases they created their own concept hierarchy from their folder structure and aligned it with the core ontology. They did not create new relations. Instance

assignment took place, but was not significant. We omitted the use of the automatic functions to get a better grasp of the actions the users did manually.

Analyzing. The members of the board gathered the evolving structures and analyzed them with help of the OntoEdit plug-in. The following observations were made:

Concepts matched A third of the extracted folder names was directly aligned with the core ontology. A further tenth of them was used to extend existing concepts.

Folder names indicate relations In the core ontology a relation inYear between the concept Indicator and Temporal was defined. This kind of relation is often encoded in one folder name. *e.g.* the folder name "SustInd2002" matches the concepts Sustainable Indicator and Year[9]. It also points to a modelling problem, since Sustainable Indicator is a concept while "2002" is an instance of concept Year.

Missing top level concepts The concept project was introduced by more than half of the participants, but was not part of the initial shared ontology.

Refinement of concepts The top level concept Indicator was extended by more than half of the participants, while other concepts were not extended.

Concepts were not used Some of the originally defined concepts were never used. We identified concepts as used, when the users created instances, or aligned documents with them. A further indicator of usage was the creation of sub concepts.

Folder names represent instances The users who defined the concept project used some of their folder names to create instances of that concept *e.g.* "Sustainable indicators project".

Different labels The originally introduced concept Natural spaces was often aligned with a newly created concept Natural environments and never used itself.

Ontology did not fit One user did create his own hierarchy and could use only one of the predefined concepts. Indeed his working area was forgotten in the first ontology building workshop.

From the discussions with the domain experts we have the impression that the local extensions are a good indicator for the evolution direction of the core ontology. However, since the users made use of the possibility to extend the core ontology with their folder names, as we expected, the resulting local ontologies represent the subjects of the organized documents. Therefore, a knowledge engineer is still needed to extend the core ontology, but the basis of his work is being improved significantly. From our point of view there is only a limited potential to automate this process.

Revision. The board extended the core ontology where it was necessary and performed some renaming. More specifically the board introduced (1) one top level concept (Project) and (2) four sub concepts of the top level concept Indicator and one for the concept Document. The users were further pointed to the possibility to create instances of the introduced concepts. *E.g.* some folder names specified project names, thus could be enriched by such an annotation.

Local update. The extensions to the core ontology were distributed to the users. The general feedback of the users was generally positive. However, due to the early development stage of the SWAP environment a prolonged evaluation of the user behavior and second cycle in the ontology engineering process has not yet been performed.

[9] Year is sub class of class Temporal

5 Lessons learned

The case study helped us to generally better comprehend the use of ontologies in a peer-to-peer environment. First of all our users did understand the ontology mainly as a classification hierarchy for their documents. Hence, they did not create instances of the defined concepts. However, our expectation that folder structures can serve as a good input for an ontology engineer to build an ontology was met.

Currently we doubt that our manual approach to analyzing local structures will scale to cases with many more users. Therefore, we look into technical support to recognize similarities in user behavior. Furthermore, the local update will be a problem when changes happen more often. Last, but not least, we have so far only addressed the ontology creation task itself – we have not yet measured if users get better and faster responses with the help of DILIGENT-engineered ontologies. All this remains work to be done in future.

In spite of the technical challenges, user feedback was very positive since (i) the tool was integrated into their daily work environment and could be easily used and (ii) the tool provided very beneficial support to perform their tasks.

6 Related work

An extensive state-of-the-art overview of methodologies for ontology engineering can be found in (*cf.* [14]). We here briefly present some of the most well-known ontology engineering methodologies.

CommonKADS [1] is not *per se* a methodology for ontology development. It covers aspects from corporate knowledge management, through knowledge analysis and engineering, to the design and implementation of knowledge-intensive information systems. CommonKADS has a focus on the initial phases for developing knowledge management applications, one can therefore make use of CommonKADS e.g. for early feasibility stages.

Methontology [14] is a methodology for building ontologies either from scratch, reusing other ontologies as they are, or by a process of re-engineering them. The framework consists of: identification of the ontology development process where the main activities are identified (evaluation, configuration, management, conceptualization, integration implementation, *etc.*); a lifecycle based on evolving prototypes; and the methodology itself, which specifies the steps to be taken to perform each activity, the techniques used, the products to be output and how they are to be evaluated.

Even though Methontology already mentions evolving prototypes, none of these (and similar others) methodologies responds to the requirements for distributed, loosely controlled and dynamic ontology engineering.

There exists a plethora of 'ontology editors'. We briefly compare two of the most well-known ones to OntoEdit *viz.* Protégé and WebODE. The design of Protégé [23] is very similar to OntoEdit since it actually was the first editor with an extensible plug-in structure and it also relies on the frame paradigm for modelling. Numerous plug-ins from external developers exist. WebODE [24] is an ontology engineering workbench

that provides various services for ontology engineering. Similar to OntoEdit it is accompanied by a sophisticated methodology of ontology engineering, see above Methontology. However, no support of these tools is so far known for distributed, loosely controlled and evolving ontology engineering such as we have presented for OntoEdit.

There are a number of technical solutions to tackle problems of remote collaboration, *e.g.* ontology editing with mutual exclusion [25, 26], inconsistency detection with a voting mechanism [27] or evolution of ontologies by different means [17, 18, 22]. APECKS [28] allows users to discuss different modelling decisions online. All these solutions address the issue of keeping an ontology consistent. Obviously, none supports (and do not intend to) the work process of the ontology engineers by way of a methodology.

The development of the National Cancer Institute Thésaurus [29] could be an interesting application scenario for DILIGENT, because their processes seem to follow our process templates. However, they focus on the creation of the thésaurus itself rather than on a generalizable methodology.

7 Conclusion

It is now widely agreed that ontologies are a core enabler for the Semantic Web vision. The development of ontologies in centralized settings is well studied and established methodologies exist. However, current experiences from projects suggest, that ontology engineering should be subject to continuous improvement rather than a one time action and that ontologies promise the most benefits in decentralized rather than centralized systems. Hence, a methodology for distributed, loosely-controlled and dynamic ontology engineering settings is needed. The current version of DILIGENT is a step towards such a methodology.

DILIGENT comprises the steps **Build**, **Local Adaptation**, **Analysis**, **Revision** and **Local Update** and introduces a board to supervise changes to a shared core ontology. The DILIGENT methodology is supported by an OntoEdit plug-in, which is an implementation of the *Edit* component in the SWAP system. The plug-in supports the board mainly in recognizing changes to the core ontology by different users during the analysis and revision steps and highlights commonalities. It thus supports the user in extending and changing the core.

We have applied the methodology with good results in a case study at IBIT, one of the partners of the SWAP project. We found that the local extensions are very document centered. Though we are aware that this may often lead to unclean ontologies, we believe it to be one (of many) important step(s) towards creating a practical semantic web in the near future.

Acknowledgements. Research reported in this paper has been partially financed by EU in the IST project SWAP (IST-2001-34103), the IST thematic network OntoWeb (IST-2000-29243), the IST project SEKT (IST-2003-506826) and Fundação Calouste Gulbenkian (21-63057-B). In particular we want to thank Immaculada Salamanca and Esteve Lladó Martí from IBIT for the fruitful discussions and the other people in the SWAP team for their collaboration towards SWAPSTER.

References

1. Schreiber, G., et al.: Knowledge Engineering and Management — The CommonKADS Methodology. The MIT Press, Cambridge, Massachusetts; London, England (1999)
2. Camarinha-Matos, L.M., Afsarmanesh, H., eds.: Processes and Foundations for Virtual Organizations. Volume 262 of IFIP INTERNATIONAL FEDERATION FOR INFORMATION PROCESSIN. Kluwer Academic Publishers (2003)
3. Bonifacio, M., Bouquet, P., Mameli, G., Nori, M.: Peer-mediated distributed knowldege management. [30] To appear 2003.
4. Ehrig, M., Haase, P., van Harmelen, F., Siebes, R., Staab, S., Stuckenschmidt, H., Studer, R., Tempich, C.: The swap data and metadata model for semantics-based peer-to-peer systems. In: Proceedings of MATES-2003. First German Conference on Multiagent Technologies. LNAI, Erfurt, Germany, Springer (2003)
5. Klyne, G., Carroll, J.J.: Resource Description Framework (RDF): Concepts and abstract syntax. `http://www.w3.org/TR/rdf-concepts/` (2003)
6. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A generic architecture for storing and querying RDF and RDFSchema. [31] 54–68
7. Gong, L.: Project JXTA: A technology overview. Technical report, Sun Micros. Inc. (2001)
8. Broekstra, J.: SeRQL: Sesame RDF query language. In Ehrig, M., et al., eds.: SWAP Deliverable 3.2 Method Design. (2003) 55–68
9. Karvounarakis, G., et al.: Querying RDF descriptions for community web portals. In: Proceedings of The French National Conference on Databases 2001 (BDA'01), Agadir, Maroc (2001) 133–144
10. Sure, Y., Angele, J., Staab, S.: OntoEdit: Multifaceted inferencing for ontology engineering. Journal on Data Semantics, LNCS **2800** (2003) 128–152
11. Pinto, H.S., Martins, J.: Evolving Ontologies in Distributed and Dynamic Settings. In Fensel, D., Giunchiglia, F., McGuinness, D., Williams, M., eds.: Proc. of the 8th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR2002), San Francisco, Morgan Kaufmann (2002) 365–374
12. Staab, S., Schnurr, H.P., Studer, R., Sure, Y.: Knowledge processes and ontologies. IEEE Intelligent Systems **16** (2001) Special Issue on Knowledge Management.
13. Gangemi, A., Pisanelli, D., Steve, G.: Ontology integration: Experiences with medical terminologies. In Guarino, N., ed.: Formal Ontology in Information Systems, Amsterdam, IOS Press (1998) 163–178
14. Gómez-Pérez, A., Fernández-López, M., Corcho, O.: Ontological Engineering. Advanced Information and Knowlege Processing. Springer (2003)
15. Pinto, H.S., Martins, J.: A Methodology for Ontology Integration. In: Proc. of the First Int. Conf. on Knowledge Capture (K-CAP2001), New York, ACM Press (2001) 131–138
16. Uschold, M., King, M.: Towards a methodology for building ontologies. In: Proc. of IJCAI95's WS on Basic Ontological Issues in Knowledge Sharing, Montreal, Canada (1995)
17. Noy, N., Klein, M.: Ontology evolution: Not the same as schema evolution. Knowledge and Information Systems (2003)
18. Stojanovic, L., et al.: User-driven ontology evolution management. In: Proc. of the 13th Europ. Conf. on Knowledge Eng. and Knowledge Man. EKAW, Madrid, Spain (2002)
19. Guarino, N., Welty, C.: Evaluating ontological decisions with OntoClean. Communications of the ACM **45** (2002) 61–65
20. Noy, N., Musen, M.: The PROMPT suite: Interactive tools for ontology merging and mapping. Technical report, SMI, Stanford University, CA, USA (2002)
21. Sebastiani, F.: Machine learning in automated text categorization. ACM Computing Surveys **34** (2002) 1–47

22. Maedche, A., Motik, B., Stojanovic, L.: Managing multiple and distributed ontologies on the semantic web. The VLDB Journal **12** (2003) 286–302
23. Noy, N., Fergerson, R., Musen, M.: The knowledge model of Protégé-2000: Combining interoperability and flexibility. In Dieng, R., Corby, O., eds.: Proc. of the 12th Int. Conf. on Knowledge Eng. and Knowledge Man.: Methods, Models, and Tools (EKAW 2000). Volume 1937 of LNAI., Juan-les-Pins, France, Springer (2000) 17–32
24. Arpírez, J.C., et al.: WebODE: a scalable workbench for ontological engineering. In: Proceedings of the First Int. Conf. on Knowledge Capture (K-CAP) Oct. 21-23, 2001, Victoria, B.C., Canada. (2001)
25. Farquhar, A., et al.: The ontolingua server: A tool for collaborative ontology construction. Technical report KSL 96-26, Stanford (1996)
26. Sure, Y., Erdmann, M., Angele, J., Staab, S., Studer, R., Wenke, D.: OntoEdit: Collaborative ontology development for the semantic web. [31] 221–235
27. Pease, A., Li, J.: Agent-mediated knowledge engineering collaboration. [30] 405–415
28. Tennison, J., Shadbolt, N.R.: APECKS: a Tool to Support Living Ontologies. In Gaines, B., Musen, M., eds.: 11th Knowledge Acquisition for Knowledge-Bases Systems Workshop (KAW98). (1998) 1–20
29. Golbeck, J., Fragoso, G., Hartel, F., Hendler, J., Parsia, B., Oberthaler, J.: The national cancer institute's thesaurus and ontology. Journal of Web Semantics **1** (2003)
30. van Elst, L., et al., eds. LNAI. Springer, Berlin (2003)
31. Horrocks, I., Hendler, J., eds. In Horrocks, I., Hendler, J., eds.: Proc. of the 1st Int. Semantic Web Conf. (ISWC 2002). Volume 2342 of LNCS., Sardinia, IT, Springer (2002)