Ontology Engineering and Evolution in a Distributed World Using DILIGENT

H. Sofia Pinto[1], C. Tempich[2], and Steffen Staab[3]

[1] Dep. de Engenharia Informática, Instituto Superior Técnico, Av. Rovisco Pais, 1049-001 Lisboa, Portugal, sofia.pinto@dei.ist.utl.pt
[2] Institute AIFB, University of Karlsruhe (TH), 76128 Karlsruhe, Germany, tempich@aifb.uni-karlsruhe.de
[3] ISWeb, University of Koblenz Landau, 56016 Koblenz, Germany, staab@uni-koblenz.de

Summary. Existing mature ontology engineering approaches are based on some basic assumptions that are often neglected in practice.

Ontologies often need to be built in a decentralized way, ontologies must be given to a community in a way such that individuals have partial autonomy over them, ontologies have a life cycle that involves an iteration back and forth between construction/modification and use and ontologies should support the participation of non-expert users in ontology engineering processes.

While recently there have been some initial proposals to consider these issues, they lack the appropriate rigor of mature approaches. i.e. these recent proposals lack the appropriate depth of methodological description, which makes the methodology usable, and they lack a proof of concept by concrete cases studies. In this paper, we describe the DILIGENT methodology that takes decentralization, partial autonomy, iteration and non-expert builders into account and we demonstrate its proof-of-concept in two real-world organizational case studies.

1 Introduction and Motivation

Ontologies are used to improve the quality of communication between computers, between humans and computers as well as between humans. Therefore an ontology should result from an agreement between its different stakeholders and this agreement must be reached in a comprehensive ontology engineering process. There are several mature methodologies that have been proposed to structure this process and thus to facilitate it (cf. chapter "Ontology Engineering Methodology" and [4, 17, 24]) and their success has been demonstrated in a number of applications. Nevertheless, these methodologies make some basic assumptions about the way the ontology engineering process takes place and about the way the resulting ontologies are used. In practice, we thus observe that these methodologies neglect some important issues:

1. *Decentralization*: Existing methodologies do not take into account that even a medium size group of stakeholders of an ontology is often quite *distributed* and does not necessarily meet often or easily. These methodologies approach ontology engineering in the same style that knowledge-based systems were approached in the past: while the user group of a resulting ontology may be large, its development is performed by a comparatively small group of (1) domain experts who *represent* the user community and (2) ontology engineers who *help structuring* that knowledge.
   In contrast, we have observed that ontology-based applications tend to be built and used in a more widely distributed fashion. By distributed we mean, not only geographically dispersed, but also involving a large number of interested parties from different organizations, with different areas of expertise and competence, different kinds of users with different requirements, etc. For instance, the Gene Ontology (GO), as reported in its web page,[1] is a result of a consortium with 99 members from 18 organizations distributed worldwide, and statistics show above 1,000 hits per week of the GO download web page, on average. Therefore, it almost seems a characteristic of ontologies, that they are more useful if the systems that they support are reaching out over several locations, several independent information systems and several, if not many, independent groups of users. However, applications that are heavily distributed, e.g. applications for virtual organizations[2] or ontology-based Peer-to-Peer applications[3] or Semantic Web applications, have people and organizations frequently leaving or joining a network. Therefore, ontology engineering processes targeting more traditional, centralized knowledge structures do not provide a representative picture of what the stakeholders of the ontology require. In such a scenario, the ontology development process needs to integrate a wider group of stakeholders, and take into account that stakeholders will hardly ever gather in one place – not even in a virtual space.
   Therefore, ontology engineering methodologies need to consider *decentralization* in depth and provide corresponding methodological support.
2. *Partial Autonomy*: We have had the experience that potential users of an ontology are typically forced to use an ontology as is, but that they are commonly not able to influence its development and have to forget about it if it does not fit their needs exactly. A typical situation that we have encountered was that people want to retain a part of the shared ontology and *modify it locally*, i.e. personalize it [13].

[1] `http://www.geneontology.org`
[2] `http://www.virtuelle-fabrik.com`
[3] `http://swap.semanticweb.org`

There have been very few approaches that have touched upon the issue of adaptation to individual purposes [10, 14]. Most of these approaches have targeted this question by considering the re-use of (parts of) ontologies for constructing a new and rather independent ontology, while in the setting of individual adaptation one rather needs to construct a living view onto an existing ontology that is augmented by individual, idiosyncratic extensions.

Thus, existing methodologies have not really dealt with users adapting ontologies for personal use.

3. *Iteration*: Existing ontology engineering methodologies mention the problem of evolving the ontology, but the actual cases that support the methodologies are typically cases where the ontology construction phase strictly precedes its usage phase.
   In contrast, we often see the need for interleaving ontology construction and use [13]. Moreover, there is a lack of case studies that support hypotheses about how to iterate in the ontology *evolution* process.
   Therefore, evolution needs to be addressed in real, and long run case studies.
4. *Non-expert builders*: Existing ontology engineering methodologies have been derived in a style useful for knowledge engineers. These methodologies propose check lists to guide the engineering process which have been shaped by the needs of knowledge engineers to cope with a complex process and to come up with an often intricate resulting system or ontology. In contrast, in the distributed, evolving cases we consider, the participation of a knowledge engineer is often restricted to a, possibly complex, core ontology. Beyond the core, typical application cases involve extensive participation and, comparatively simple, concept formation by *domain experts* and/or *users*. Support for their participation is mostly lacking in these methodologies.

These issues arise naturally for many ontologies, e.g. [15] or GO and one might claim for all ontologies in the Semantic Web! Recently a number of other approaches that touch these issues have been proposed [1, 6]. However, *none* goes very far from a methodological point of view, namely they do not provide elaborated methodological support, or were extensively used in concrete case studies with regard to these four issues, such as actions to take, their input and output, etc.

Therefore, to account for some of the differences between classical knowledge engineering and ontology engineering methodologies derived from there, we thus have started to develop DILIGENT, a methodology for:

1. DIstributed
2. Loosely-controlled, and
3. evolvInG Engineering of oNTologies that is able to
4. support non-expert ontology builders

While developing DILIGENT, we also had to consider two general methodological objectives:

First, we wanted to provide guidance to the knowledge engineer, the ontology engineer and the non-expert ontology builders that was as fine-grained as possible to make the sequence of tasks as concrete and re-producible by novices as possible.

Second, we needed to check DILIGENT by some concrete case studies to show that it can live up to its promises. Clearly, it is very difficult to near impossible to match any methodology, which constitutes an abstraction of many processes, onto an instantiated process in detail. Nevertheless without a reasonable substantiation of the proposed steps in concrete case studies a proposal like DILIGENT would remain vacuous.

We will therefore describe DILIGENT in detail and some experiences where it was shown how it maps onto comprehensive case studies. Nevertheless, it will not be possible to describe the finest grain size of DILIGENT. At the finest grain size of methodological support, we have proposed an argumentation framework, an argumentation ontology, technical support and several case studies to investigate only these aspects. Including all these investigations in depth as required by a sound scientific presentation would have doubled the size of this paper, hence we only refer to this work here [12, 19, 20, 23] and sketch it briefly in Sect. 4.

In the following, we present our ontology engineering methodology, DILIGENT. In Sect. 2 we give an overview of how we have proceeded to design and validate DILIGENT. In Sect. 3 we describe DILIGENT elaborating the hierarchical task structure in detail. In Sect. 4, we briefly describe how we have applied DILIGENT in some comprehensive case studies, i.e. a distributed knowledge management scenario supported by an ontology-based peer-to-peer knowledge sharing platform and supported by wikis. Eventually, we compare with related work in Sect. 5 and conclude.

2 Developing the DILIGENT Ontology Engineering Methodology

In order to arrive at a sound Ontology Engineering (OE) methodology we have proceeded in five steps to develop DILIGENT.

Around 2000, ontology engineering efforts with a clear distributed, loosely-controlled and dynamic flavor were taking place. For instance SUMO[4] was being collaboratively developed by a group of worldwide distributed researchers, in a loosely-controlled and evolutionary fashion. No particular methodologies were being followed to control these new features, but these processes were clearly following different process models from the ones that were being tackled by the methodologies available at that time. These new efforts [13] provided the initial ideas to conceive our initial DILIGENT framework.

[4] `http://suo.ieee.org/`

Second, the first step in DILIGENT consists of the construction of a core ontology (cf. Sect. 3). In this step DILIGENT does not introduce any special or new requirements for the core ontology when compared to the ones dealt with by existing methodologies (cf. Sect. 1). Therefore, with regard to this step, we have decided not to develop a new methodology, but to borrow from existing work. We expect that any mature methodology can be used. In our case studies, we have exploited the OTK-methodology (chapter "Ontology Engineering Methodology").

Third, in order to validate the combined methodology we proceed in two fronts. On the one hand, we analyzed its potential for the past and ongoing development process of the biological taxonomy of living species. When we analyze its evolution since 1735 one can realize that it completely follows the 5-step DILIGENT process, as briefly described in Sect. 3. On the other hand, we conducted a lab experiment case study to specifically investigate whether some argumentation structures dominate the progress in the ontology engineering task and should therefore be accounted for in a fine-grained methodology. Our experiments [12] provide strong indication – though not full-fledged evidence – that a restriction of arguments can enhance the ontology engineering effort in a distributed environment. Moreover it also shows us that proper social management procedures and tool support helps to reach consensus in a smoother way (cf. [2]).

Fourth, we started a real-life, cross-organizational case study in the tourism industry. We reported about its initial state supporting means in [11]. In this case study, the process template was realized in a decentralized, autonomous and collaborative setting with high personalization requirements. The process was supported by a peer-to-peer system and tools were specifically developed to support non-ontology engineering experts. Two rounds following DILIGENT were monitored over a 3 month period.

Fifth, by the sum of these initial process templates,[5] cases and experiments, we arrived at the new and refined DILIGENT methodology that we present here. The focus of the refinement has been on decentralization, iteration and partial autonomy as well as on guiding users who are not ontology engineering experts. The methodology has been validated by the iterative case study presented in Sect. 4 and others reported in the literature [23, 25]. Thus, we have repeatedly switched between hypothesis formulation and validation.

3 The DILIGENT Methodology

In order to give the necessary context for the detailed process description as described in Sect. 3.2 we start by summarizing the overall DILIGENT process model.

[5] In our terminology, a methodology for an engineering artefact is a tested and validated process template abstracting over all possible successful engineering processes for engineering the artefact.

The *DILIGENT* process [11] supports its participants, in collaboratively building one shared ontology. In DILIGENT we assume that there are several participants, with different and complementary skills, which, in most of the cases, are geographically distributed, and which have genuine interest in collaboratively building or using one ontology. For instance, in a virtual organization, the different participants may be in a "coopetition" relationship: on the one hand they may be from different but similar organizations that compete for the same resources, but on the other, to compete against external threats, they should cooperate to improve their chances of success. In this case, it may be important, for instance to promote interoperability between their applications, that they all agree on a given ontology, the shared ontology, and use it as a common ground of understanding.

There are different kinds of participants in the DILIGENT process: (1) domain experts, that know about the domain that is targeted (2) ontology engineers, that know how to build ontologies (3) knowledge engineers, that know how to build knowledge or information systems based on ontologies, and (4) users, that use the ontology resulting from the process in their systems for their own uses. The participants directly involved in building the ontology, may or may not use the ontology. However, most ontology users will typically not build or modify the given ontology. DILIGENT supports trained ontology engineers as well as typical users of information systems likewise. The ontology engineers perform the defined activities with more accuracy and awareness of the process, while the non-ontology-engineering-expert users will tend to follow them implicitly guided by the provided tools. At some points of the process there is a subset of participants that plays a special role and has added responsibilities: the board. As in the other steps of the process, the composition of the board is not fixed, that is members can enter or leave, although it should have a more stable composition than that of the participants involved in the DILIGENT process. This board is responsible for the shared ontology: in the beginning it builds the initial version of the ontology, in the iterations that follow it is responsible for the evolution of the shared ontology.

3.1 General Process

The process comprises five main activities: (1) *build*, (2) *local adaptation*, (3) *analysis*, (4) *revision*, (5) *local update*, Fig. 1. The process starts by having *domain experts*, *users*, *knowledge engineers* and *ontology engineers build*ing an initial ontology. 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. At this point, it is not required to arrive at an initial ontology that covers the complete domain.

Once the initial ontology is made available, users can start using it and *locally adapting* it for their own purposes. Typically, due to new business requirements or user and organization changes, their local ontologies evolve.

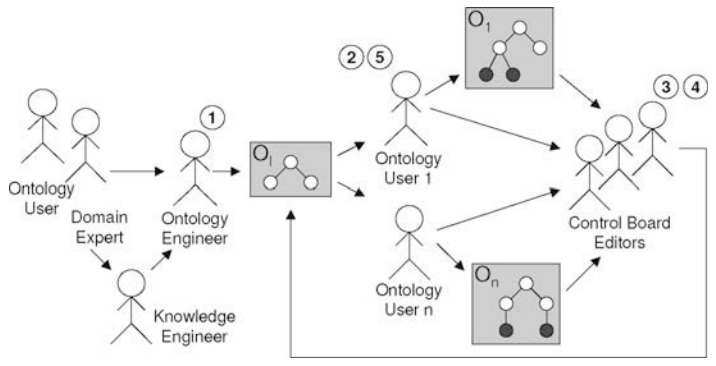

Fig. 1. Distributed, loosely controlled, evolving Ontology Engineering

In DILIGENT there are two kinds of ontologies: The *shared ontology* and *local ontologies*. The shared ontology is available to all users and cannot be changed directly except by the board. Users are free to change, in their local environments, a copy of the shared ontology. The ontology resulting from the changes of a user is the user local ontology.

A board of ontology stakeholders *analyzes* the local ontologies and the users' requests and tries to identify similarities in their ontologies. At this point it is not intended to merge all local ontologies. Instead, changes to local ontologies will be analyzed by the board in order to decide which changes introduced or requested by the users should be introduced in the shared ontology. Therefore, a crucial activity of the board is deciding which changes are going to be introduced in the next version. A balanced decision that takes into account the different needs of user's evolving requirements has to be found.

The board should regularly *revise* the shared ontology, so that the parts of the local ontologies overlapping the domain of the shared ontology do not diverge too far from it. Therefore, the board should have a well-balanced and representative participation of the different kinds of participants involved in the process, which includes *ontology engineers*, *domain experts* and *users*. Of course, these are roles that may overlap.

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. The last four stages of the process are performed in a cyclic manner: when a new common ontology is available a new round starts again.

There are evidences that this process template can be used in different areas and therefore understanding and better supporting it is important. For instance, the taxonomy of life on earth has been evolving since 1735 following a DILIGENT like 5-step process. It was initially proposed by Linnaeus (*build*) based on phenetics (observable features). Considering the "most general" level, initially, two kingdoms were proposed: animals and plants. As more and more detailed knowledge about them was discovered, new kingdoms were proposed by its users and introduced by the boards controlling them once some consensus was reached. For instance, when microorganisms were discovered the moving ones were classified in the animals kingdom and the colored (non-moving) ones in the plants kingdom (*local adaptation*). A few of them were classified in both kingdoms. Users were locally adapting the taxonomy for their own purposes. To more easily identify organisms in both classes, Haeckel (1894) proposed a new kingdom to more easily identify them, the Protista kingdom. This change was introduced by the board (*analysis and revision*). This kingdom still exists today (*locally update*) and is used to gather all organisms that do not belong to one of the other kingdoms. The major force driving the reorganization of the taxonomy over time has been the identification of important classifying features and gathering all beings sharing a given value for that feature into that class. The parallel between DILIGENT template process and the development of the taxonomy of life on earth is far more deep than described here. For other examples see [12].

3.2 DILIGENT Process Stages

In order to facilitate the application of DILIGENT ontology engineering processes and provide guidance to its participants in real settings, DILIGENT had to be more detailed. For this purpose, we have analyzed the different process stages in detail. For each stage we have identified (1) major roles, (2) input, (3) decisions, (4) actions, (5) available tools, and (6) output information. One should stress that this elaboration is rather a recipe or check list than an algorithm or integrated tool set. In different contexts it may have to be adapted or further refined to fit particular needs and settings. Tools may need to be integrated or customized to match requirements of the application context. In Fig. 2 we sketch our results, which are presented in the following. For the sake of brevity we refer the reader to [20] that includes an even more detailed process description.

Build

As mentioned before, DILIGENT focuses on distributed ontology development and ontology evolution, but borrows from established methodologies (chapter "Ontology Engineering Methodology" and [4]). This is particularly true at this stage. The goal is to quickly build an ontology that is going to be used in an application. At this stage one can follow different approaches and even approaches inspired from software engineering methodologies, such as rapid prototyping, extreme programming and open source guidelines. The motto is: get something small and useful and give it to the users, as early as possible. Therefore, there is no need for completeness, although usability and usefulness are crucial.

Roles: Usually, there are three roles: knowledge engineer, ontology engineer and domain expert. The domain expert provides both knowledge and ontology engineers with the required domain knowledge and knowledge sources.

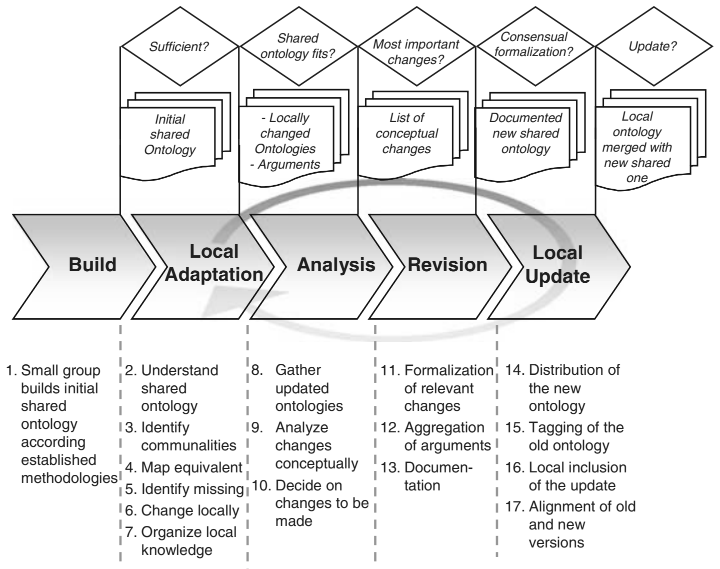
Sufficient?
Shared ontology fits?
Most important changes?
Consensual formalization?
Update?
Initial shared Ontology
- Locally changed Ontologies - Arguments
List of conceptual changes
Documented new shared ontology
Local ontology merged with new shared one
Build
Local Adaptation
Analysis
Revision
Local Update
1. Small group builds initial shared ontology according established methodologies
2. Understand shared ontology
3. Identify communalities
4. Map equivalent
5. Identify missing
6. Change locally
7. Organize local knowledge
8. Gather updated ontologies
9. Analyze changes conceptually
10. Decide on changes to be made
11. Formalization of relevant changes
12. Aggregation of arguments
13. Documentation
14. Distribution of the new ontology
15. Tagging of the old ontology
16. Local inclusion of the update
17. Alignment of old and new versions

Fig. 2. Process stages (1–5), actions (1–17) and structures

The knowledge engineer creates a conceptual model of the domain from the knowledge extracted from these sources. The ontology engineer generates a machine readable ontology from the conceptual model. Quite often the knowledge engineer and ontology engineer are roles performed by the same person. Additionally to these classical roles we also propose the involvement of users. At this stage, usually the actors involved as users are also involved in the process in one of the other more classical roles. Most of those involved in the build stage are initial board members.

Input: Since this stage borrows from traditional OE the usual pre-development activities are performed. Given our analysis of existing methodologies [21] we recommend the adoption of the OTK methodology (chapter "Ontology Engineering Methodology", since it is the one providing more guidance and has a more detailed and complete set of activities. However, the use of other methodologies is not excluded.

Decisions: The usual decisions of a classical OE process need to be taken. In contrast to common OE methodologies we do not require completeness of the ontology at this stage. It is particularly important that the ontology is clear and easily understandable by possible users.

Actions: As in classical OE development, common core activities are conceptualization, formalization, and implementation.[6] Integral activities like knowledge acquisition, evaluation, reuse (comprising fusion and composition), and documentation are complemented in DILIGENT with a recommendation for *Argument provision*.

[6] Maintenance is supported by later stages of DILIGENT.

Output: The result is an ontology with the main concepts of the domain. Once an initial ontology is (1) *built* and released, users will start to adapt it locally for their own purposes.

Local Adaptation

This is a use and personalization stage, therefore users use and adapt the released ontology to their own needs. The idea is for users to understand the shared ontology, use it in the context of their applications, eventually find some problems in the shared ontology for their particular applications that require customization on their local ontologies, and accordingly modify these to best suit their needs. All changes should be justified with arguments. Their changes will only apply to their local copies and not to the shared ontology that was made available to all users. In ideal settings, users can also have access to other users' ontologies, when customizing the shared ontology (either under the same framework or from external sources) therefore reuse of ontologies may also be performed.[7] One should stress that all traditional OE activities are usually performed by the users at this stage, such as knowledge acquisition, conceptualization, formalization, evaluation, integration, etc. Once in a while a new shared ontology is made available to users.

Roles: The actors involved in the local adaptation step are users of the ontology. However, they usually do not have an OE background. They use the ontology, e.g. to retrieve documents which are related to certain topics modeled in the ontology or more structured data like the projects an employee was involved in.

Input: Besides the common shared ontology, in the local adaptation step the information available in the local information space is used. This can be existing databases, ontologies or folder structures and documents. Moreover, external knowledge sources or ontologies can also be reused as well as other user's ontologies.

Decisions: The users decide which changes they want to make to their local ontology, hence they must decide if and where new concepts are needed and which relations a concept should have. They should provide reasons for their changes.

Actions: To achieve the desired output the user performs different groups of actions namely: Analyze the shared ontology; Change and integrate the shared and local ontologies; and Use the shared ontology. The last two actions of the process step are performed in a cyclic manner until a new shared ontology is available and the entire process step starts again.

One important issue is the fact that this stage can either be performed immediately after a build or after a local update stages. In both cases, the shared ontology is available: in the first case, it is the only ontology users have had so far, in the second they have already their own local ontologies

[7] With naive users this usually does not occur often.

that were somewhat connected (or not) to the shared ontology. Users then start adapting the shared ontology to their own purposes. Although these two situations are not different from a conceptual point of view, from a practical point of view they are different since in the second case users usually are not going to simply discard their local ontologies and build them again so that they can be connected to the new version of the shared ontology. Therefore, it is important to assure that there can be a smooth transition.

We now describe in detail each one of the proposed actions:

The Analysis of shared ontology usually involves (2) *Understand shared ontology* and (3) *Identify similarities between own and shared conceptualization.* An ontology should represent a shared conceptualization of part of the world. At this point the analysis is mainly the identification of similarities and mismatches between the available shared ontology and either the conceptual model of the domain users have in their minds or the local ontologies they already developed in previous iterations of the process.

(2) Understand the shared ontology The user must learn where the different concepts are located in the ontology and how they are interrelated. The ontology can be very complex, thus understanding the ontology depends mainly on its visualization, and good naming conventions.

(3) Identify similarities between own and shared conceptualization Following the comprehension of the ontology, the user can realize the similarities and differences between the own and shared conceptualizations

Change and integration of shared and local ontologies usually involves (4) *Map equivalent conceptualizations of different actors* (5) *Identify mismatches in conceptualizations*, and (6) *Change conceptualization.*

(4) Map equivalent conceptualizations of different actors: After the identification of similarities they should be made explicit, otherwise the system will not be able to make use of these findings in later stages. This is particularly important when the user is identifying similarities between his local concepts and the new concepts in the shared ontology. Different implementations may add specialized adds-on. Mappings have the advantage, that they leave the original local structures unchanged. Of course users may also decide to change their local structures in favor of the common structure. In this case the changes must be traceable, so that the user can retain his old version, whenever he wants.

(5) Identify mismatches in conceptualizations: The techniques to identify similarities can also be applied in the subsequent step to support the user in identifying missing conceptualizations. Depending on the scenario, the user might have access to other users' ontologies and use their local adaptations as further input to identify missing concepts in his own conceptual model.

(6) Change conceptualization: After identifying missing or unwanted conceptualizations the user must be enabled to introduce them. This is a customization phase and of course, evaluation is also performed here. Users should assure that their changes are adequate both from a domain and a representation point of view. Since later on the board analyzes the changes performed

or requested by the users, users must provide reasons for each change and/or request for change, so that the board can understand them. To support the user in providing reasons, the argumentation framework focuses the user on the relevant arguments, [19].

Ontology use typically involves that users (7) *Organize local knowledge according to the new conceptualization.* At this point the local ontology should reflect the user's conceptualization. Now he can use the ontology in his application. In our case studies ontologies were used in information retrieval scenarios therefore, ontology use typically involved the organization of local knowledge according to the local conceptualization. Therefore, the user instantiated the ontology with the information available locally and hence contributed to the collective knowledge.

Output: One output of the process step is a locally changed ontology which better reflects the user's needs. Each change is supported by arguments explaining the reasons for a change. At this point changes are not propagated to the shared ontology. Moreover, users can send requests for changes directly to the control board, which should also be duly justified. Only in the *analysis step* the board gathers all ontology changes and requests and their corresponding arguments to be able to evolve the common shared ontology in a user driven *revision step.*

Analysis

In this stage, the board analyzes incoming requests and observations of changes.[8] The idea is for the board to identify which changes should be made to the ontology based on the changes made or requested by the users. The frequency of this analysis is determined based on the frequency and volume of changes to the local ontologies. The board analyzes and decides which changes would the users most benefit from and would most like to see implemented. At this stage the new requirements for the future version of the shared ontology are identified. At this stage, work is conducted at a conceptual level. This activity borrows from classical ontology reuse processes, but is simpler since local ontologies are available in the same environment and language.

Roles: In the analysis stage we can distinguish three roles played by board members: (1) The domain expert decides which changes to the common ontology are relevant from the domain point of view and which are relevant for smaller communities only. (2) Representatives from the users explain different requirements from the usability perspective. (3) The ontology engineers analyze the proposed changes from a knowledge representation point of view foreseeing whether the requested changes could later be formalized and implemented.[9]

[8] Ideally the board should have access to all users' ontologies. However, in some settings it may only have access to requests for changes.

[9] In the revision stage.

Input: The analysis stage takes as input the ontology changes requested and/or made by the participating actors. To be able to understand their changes and requests, users should have provided their reasons. Both manual and automated methods can be used in the previous stages, therefore besides of arguments by ontology stakeholders, one may here consider rationales generated by automated methods, e.g. ontology learning. The arguments underlying the proposed changes constitute important input for the board to achieve a well balanced decision about which changes to adopt.

Decisions: The board must decide which changes to introduce into the new shared ontology at the conceptual level. Metrics to support this decision are (1) the number of users who introduced a change in proportion to all users who made changes. (2) The number of queries including certain concepts. (3) The number of concepts adapted by the users from previous rounds.

Actions: To achieve the desired output the board takes different actions namely (8) *Gather locally updated ontologies and corresponding arguments*, (9) *Analyze the introduced changes* and (10) *Decide on changes to be made.*

We now describe in detail each one of the proposed actions:

(8) Gather locally updated ontologies and corresponding arguments: Depending on the deployed application the gathering of the locally updated ontologies can be more or less difficult. It is important that the board has access to the local changes from users and their corresponding arguments to be able to analyze them. It may also be interesting not only to analyze the current users' ontologies, but also its evolution. However, with an increasing number of participants this in-depth analysis may be unfeasible. Since usually analysis takes place at the conceptual level, reverse engineering is usually an important technique to get the conceptual model from the formalized model [4]. To support users providing their reasons, the argumentation framework focuses the users on the relevant arguments, [19].

(9) Analyze introduced changes: In this action the board tries to identify the parts of the shared ontology which should be modified. As the number of change requests may be large and also contradictory, first the board must identify the different areas in which changes took place. Within analysis the board should bear in mind that changes of concepts should be analyzed before changes of relations and these before changes of axioms. Good indicators for changes relevant to the users are (1) overlapping changes and (2) their frequency. Furthermore, the board should analyze (3) the queries made to the ontology. This should help to find out which parts of the ontology are more often used. Since actors instantiate the ontology locally, (4) the number of instances for the different proposed changes can also be used to determine the relevance of certain adaptations.

(10) Decide on changes to be made: Having analyzed the changes and having grouped them according to the different parts of the ontology they belong to, the board has to identify the most relevant changes, that is identify changes presumably relevant for a significant share of all actors. Based on the provided arguments the board must decide which changes should be

introduced. Depending on the quality of the arguments the board itself might argue about different changes. For instance, the board may decide to introduce a new concept that better abstracts several specific concepts introduced by users, and connect it to the several specific ones. Therefore, the final decisions entail some form of evaluation from a domain and a usage point of view.

Output: The outcome of this action is a reduced and structured list of changes that are to be implemented in the shared ontology that were agreed by the board. Arguments should be provided for each one of them. All changes which should not be introduced into the shared ontology are filtered. Arguments justifying the decisions to leave them out should also be provided. At this stage it is not required to decide on the final modeling of the shared ontology.

Revision

The revision and analysis stages are closely related. While in the previous stage the new requirements for the shared ontology are identified, in this stage they are formalized and implemented. In the end the new version of the shared ontology is distributed to its users.

Roles: The ontology engineers from the board judge the changes from an ontological perspective more exactly at a formalization level. Some changes may be relevant for the common ontology, but may not be correctly formulated by the users. The domain experts from the board should judge and decide wether new concepts/relations should be introduced into the common ontology even though they were not requested by the users

Input: The input for the revision phase is a list of changes at a conceptual level which should be included into the ontology and the arguments underlying them.

Decisions: The main decisions in the revision phase are formal ones. All intended changes identified during the analysis phase should be included into the common ontology. In the revision phase the ontology engineer decides how the requested changes should be formalized. Evaluation of the decisions is performed by comparing the changes on the conceptual level with the final formal decisions. The differences between the original formalization by the users and the final formalization in the shared ontology should be kept to a minimal basis.

Actions: To achieve the desired output the members of the board, mainly its ontology engineers, perform different actions namely (11) *Formalization of the decided changes*, (12) *Aggregation of corresponding arguments*, (13) *Documentation*, and (14) *Distribution of the new ontology to all actors*.

We now describe in detail each one of the proposed actions:

(11) Formalization of the decided changes: As in classical OE development, the requested changes must be formalized with respect to the expressivity of the ontology representation language. Before their actual implementation, the agreed changes should be analyzed from a knowledge representation point of

view. This evaluation is somehow similar to the one performed when reusing an ontology according to classical reuse methodologies. The goal is to determine how the changes identified in the previous step should be formalized. Once this is done, the actual changes are formalized and the quality of the resulting ontology is again assured through evaluation. All required activities are addressed by classical OE methodologies.

(12) Aggregation of arguments: As arguments play a major roll in the decision process we expect that the changes which are eventually included into the common ontology are supported by good arguments. One of the reasons for keeping track of the arguments is to enable users to better understand why certain decisions have been made. Therefore, the board should summarize and aggregate understandable, pedagogical and the most convincing arguments underlying each change. The user should be able to retrieve them.

(13) Documentation: With the help of the arguments, the introduced changes are already well documented. However, we assume that some arguments may only be understandable by the domain experts and not users. Hence, we expect that the changes should be documented to a certain level.

(14) Distribution of the ontology to all actors: Analogously to stage (1) the shared ontology must be distributed to the different participants. Depending on the overall system architecture different methods can be applied here. Moreover, the board should assure version and release management.

Output: The new version of the shared ontology together with its arguments and documentation is the result of this stage. This documentation is essential for users to understand the new shared ontology when a new cycle begins.

Local Update

In the local update stage the new shared ontology is released and put to use by its users. They decide which changes they will adopt. Part of this stage is similar to local adaptation: users must get familiar with the new version and identify which parts of their local ontologies they will discard in favor of the new shared ontology and which ones they will retain.

Roles: The local update phase involves only users. They perform different actions to include the new common ontology into their local system before they start a new round of local adaptation.

Input: The new formalized shared ontology is the input for this step. We also require as input the documentation and arguments justifying those changes. For a better understanding the user should be allowed to request a delta to the original version.

Decisions: The user must decide which changes he will introduce locally. This depends on the differences between the own and the new shared conceptualization. The user does not need to update his entire ontology. This stage interferes a lot with the next local adaptation stage. We do not exclude the possibility of conflicts and/or ambiguity between local and shared ontologies, which may entail reduced precision if the ontology is being used in IR applications.[10]

Actions: To achieve the desired output the user takes different actions namely *Analysis of the new shared ontology*; and *Integration of new shared version with current user's local one*.

After the local update, the iteration continues with local adaptation. During the next analysis step the board reviews which changes were actually accepted by the users.

We now describe in detail each one of the proposed actions:

Analysis of the new shared ontology: The goal is to understand the new shared ontology. The user scans for the changes introduced by the board that are relevant for his use, and controls whether his change proposals were implemented. He must further identify wether the benefits of updating to the new version outweight its effort. Issues to be analyzed include: concepts introduced by other users, consistency of new shared version with local version, maintenance of interoperability with other users.

Integration of new shared version with current user's local one: In this action the user reuses or not the new version of the shared ontology. If the new shared ontology is not of use the system should allow the user to retain the outdated version. In this case the user will have to perform (15) *Tagging of the outdated version*. In case the user finds the new shared version of use two further subactions can be performed: (16) *Inclusion of the updated version* and (17) *update local adaptations not included in the common ontology*.

(15) Tagging of the outdated version: To ensure user satisfaction, the system must enable the user to retain his old version of the ontology or parts of it. The user may later realize that the new updated version of the common ontology does not represent his needs anymore and thus want to leave the update cycle out and return to the old version. To reach a better acceptance this must be possible and is foreseen in the methodology. The user can always balance between the advantages of using a shared ontology or using his own conceptual model. Therefore, the old version should be stored for possible later reuse.

(16) Inclusion of the updated version: The system must support the user to easily integrate the new version into his local system. It must be guaranteed that all annotations made for the old version of the ontology are available in the new version. It may require restructuring and adaptation of instantiations to stay in line with the new model.

(17) Update of local adaptations which are not included in the common ontology: The update of the local ontology can lead to different kinds of conflict. Changes proposed by the user may indeed have found their way into the common ontology. Hence, the user should be enabled to use from now on

[10] Ideally one should be able to blacken out the ambiguous parts like in multilevel databases. This has not been transferred to OE yet.

the shared model instead of his own identical model. Furthermore, the board might have included a change based on arguments the user was bringing forward, but has drawn different conclusions. Here the user can decide wether he prefers the shared interpretation.

Other options may emerge in the course of further case studies.

Output: Ideally the output of the local update phase is an updated local ontology which includes all changes made to the shared ontology. However, since not all users may want to completely change to the new version, we do not require the users to adopt all changes proposed by the board. So, the output is not mandatory since the actors could change the new ontology back to the old one in the local adaptation stage.

4 Applying DILIGENT in Case Studies

In this section we describe briefly how we specifically investigated how a distributed, loosely controlled and evolving ontology engineering process following DILIGENTcould be implemented. For more detailed descriptions refer to the relevant bibliography referred in each subsection.

4.1 The IBIT Case Study

The first running case study took place under the SWAP project. In this project, the challenges were how the process template could be implemented in a multi-organizational setting with non-expert ontology engineering users, and which finer grained support could be provided to these users.

In the SWAP project, the IBIT case study was in the tourism domain of the Balearic Islands. The needs of the tourism industry there, which accounts 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,[11] 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.

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

Due to the different working areas and goals of the collaborating organizations, it proved impossible to build a centralized knowledge management system or even a centralized ontology satisfying all user requirements. The users emphasized the need for local control over their ontologies. They asked explicitly for a system without a central server, where knowledge sharing was integrated into the normal work, but where different kinds of information, like files, emails, bookmarks and addresses could be shared with others. To this end the SWAP consortium – including us at University of Karlsruhe, IBIT, Free Univ. Amsterdam, Meta4, and Empolis – developed the SWAP generic P2P platform and built a concrete application on top that allowed the satisfaction of the information sharing needs just elaborated using local ontologies, which were linked to a shared ontology. A case study was set up. The main goals were the evaluation of the DILIGENT process and the developed peer-to-peer platform. The case study lasted for 3 months. Moreover, a set of tools were also specifically developed [18] to support the participants in the case study. However, most of the tools were being developed at the same time as the process was taking place. Therefore, the administrator had a major role in bridging the gap between our real users and the weaknesses of the tools, for instance by doing local adaptations for the users since the tools were not error-proof.

Regarding the methodology we had four hypothesis: (1) DILIGENT supports collaborative development of a shared ontology; (2) ontologies in use need to evolve; (3) non-ontology engineering experts can participate in ontology engineering processes, and (4) the organizational structure DILIGENT suggests fits the organizational setting found in the IBIT case study, a peer-to-peer setting.

The first round of our OE process started with the distribution of the three modules of the common ontology to all users. In both rounds, users – during the local adaptation stage – and the board – in the revision stage – could perform ontology change operations (concepts/relations/instances). Most frequently the concept hierarchy was changed.

The first month of the case study, corresponded to the first round of the DILIGENT process. One organization with seven peers participated. This organization can be classified as a rather loose one. In the first round we had seven users, six of which had no OE background. In general, the users added concepts to the shared ontology to represent the topics of their core working area. They did not share all their local information, but selected the documents which they thought would be interesting for the group. In the interviews they commented, that they would share more files at a later stage, when they would feel more confident with the system. In this organization most of the users were very active and did local adaptations to best serve their own needs. They also add access to other user's ontologies. Moreover, the board received by e-mail requests to modify the shared ontology. The first round of the process resulted in seven adapted ontologies.

In Analysis, the board consisted of two ontology engineers and two domain experts/users, the same that were involved in the build stage. The local adaptations from seven users were collected. Additionally the board had access to their folder structures. All changes introduced were motivated by the users' requests and changes. They all made sense and were not contradictory on the conceptual level. Then, the new shared ontology was distributed.

In Local update all users decided to use the new shared ontology as it covered more domain knowledge and they found their requests integrated to it. As a result of this stage the new shared ontology was commonly used and the users' folders were aligned with the new shared ontology.

In the second round the case study was extended to four organizations with 21 peers. The users participating in the first round had more experience and were still active. One of the new organizations was very hierarchical. None of the new 14 users had OE experience. The experienced users started with the result of the local update stage, while the new users received only the new shared ontology. All users shared the local information which they thought relevant for the group. The new users behaved in a similar way as the users in the first stage and did not share many folders, as they wanted to gain confidence in the system first. The experienced users, however, published more information, and adapted the local ontologies accordingly. The second local adaptation stage resulted in 14 adapted ontologies. The rest of the users did not make changes. Although, some did not change the shared ontology directly, they submitted change requests to their supervisor, thus they delegated the modeling task. The supervisor then communicated the requests to the board.

In Analysis, in this round the board consisted of one domain expert and two ontology engineers. Additionally two users were invited to answer questions to clarify the changes they introduced. The 21 local ontologies of the users were the input to the second round. This time the board had to perform reverse engineering on the formal local ontologies from users in order to get their conceptual models. As in the first round the board included all change requests from users. Again, as in the first round, only very few concepts in the common ontology were never used. All conceptual requests could be modelled in the ontology, providing the next version.

The case study ended after the distribution of the new shared ontology. We collected feedback from the users w.r.t. to their impressions on the new version. They emphasized that the new version represented their requirements at that time. The users commented that they appreciated being involved in the development process, although they recognized that they were not experienced in ontology engineering. They did not object to the modeling decisions of the board and understood the reasons for the differences between their change requests and the final modeling.

However, updating to the new version was still a problem, since some instances of the ontology might have to be newly annotated to the new concepts of the shared ontology. In our case, documents needed a new classification. This problem can be partly overcome with the help of technology [7].

For more detailed descriptions on this project refer to [20, 21].

4.2 The Judges Case Study

The Judges case study took place under the SEKT project. It aimed at providing an intelligent Frequently Asked Questions system, Iuriservice, that offers help to newly appointed judges in Spain. Although judges had a strong and thorough education and became experts in their domain, they still often seek the help of senior judges or tutors regarding procedural questions. The system focuses on such procedural knowledge, which is often neglected, as it is very hard to externalize. Examples for procedural questions are: How should I organize a round of recognition of suspects if there are no people available? Which are the actual functions and competences of the judge as compared to those of the secretaries?

In this regard, the design of legal ontologies requires not only to represent the legal, normative language of written documents (decisions, judgments, rulings, partitions, etc.) but also those chunks of professional knowledge from the daily practice at courts. One of the main features of this professional legal knowledge is that it is context-sensitive. In this sense, it implies: (1) the ability to discriminate among related but different situations; (2) the practical attitude or disposition to rule, judge or make a decision; (3) the ability to relate new and past experiences of cases; (4) the ability to share and discuss these experiences with the group of peers.

In this case, the argumentation framework developed under the DILIGENTmethodology, together with a wiki system proved an invaluable tool that promoted discussion and allowed finding good solutions for the problems newly appointed judges faced.

For more detailed descriptions on this project refer to [20, 22].

5 Related Work

In the past, there have been OE case studies involving dispersed teams, such as $(KA)^2$ ontology [1] or [13]. However, they usually involved tight control of the ontology, of its development process, and of a small team of ontology engineering experts that could cope with the lack of precise guidelines.

Established methodologies for ontology engineering summarized in [4, 17, 24], focus on the centralized development of static ontologies, i.e. they do not consider iteration between construction/modification and use. *METHONTOLOGY* [4] and the *OTK methodology* [17] are good examples for this approach. They offer guidance for building ontologies either from scratch, reusing other ontologies as they are, or re-engineering them. They divide OE processes into several stages which produce an evaluated ontology for a specific domain.

Holsapple et al. [5] focus their methodology on the collaborative aspects of ontology engineering but still aim at a static ontology. A knowledge engineer defines an initial ontology which is extended and modified based on the feedback from a panel of domain experts. *HCOME* is a methodology which integrates argumentation and ontology engineering in a distributed setting [6]. It supports the development of ontologies in a decentralized setting and allows for ontology evolution. It introduces three different spaces in which ontologies can be stored: In the *Personal Space* users can create and merge ontologies, control ontology versions, map terms and word senses to concepts and consult the top ontology. The evolving personal ontologies can be shared in the *Shared Space*. The *Shared Space* can be accessed by all participants. In the shared space users can discuss ontological decisions. After some discussion and agreement, the ontology is moved into the *Agreed space*. However, they have neither reported that their methodology had been applied in a case study nor do they provide any detailed description of the defined process stages.

There are a number of technical solutions to tackle problems of remote collaboration, e.g. ontology editing with mutual exclusion [3], inconsistency detection with a voting mechanism [9], collaborative ontology editing [8, 16] or evolution of ontologies by different means [7]. 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.

6 Conclusion

Decentralization can take different forms. One can have more loose or more hierarchical organizations. We observed and supported both kinds of organizations. Therefore, the first finding is the fact that this process can be adapted both to hierarchical and to more loose organizations. DILIGENT processes cover both traditional OE processes and more Semantic Web-oriented OE processes, that is with strong decentralization and partial autonomy requirements.

The process helped non-OE-expert users to conceptualize, specialize and refine their domain. The agreement met with the formalized ontologies was high, as shown by people willing to change their folder structures to better use the improved domain conceptualization. In spite of the technical challenges, user feedback was very positive.

The DILIGENT process proved to be a natural way to have different people from different organizations collaborate and change the shared ontology. The set-up phase for DILIGENT was rather fast, and users could profit from their own proposals (local adaptations) immediately. The result was much closer to the user's own requirements. Moreover, other users profited from them in a longer term. Finally, the case studies clearly have shown the need for evolution. Users performed changes and adaptations.

The development of ontologies in centralized settings is well studied and there are established methodologies. However, current experiences from projects suggest that ontology engineering should be subject to continuous improvement rather than a one-time effort and that ontologies promise the most benefits in decentralized rather than centralized systems. To this end we have conceived the DILIGENT methodology. DILIGENT supports domain experts, users, knowledge engineers and ontology engineers in collaboratively building a shared ontology in a distributed setting. Moreover, the methodology guides the participants in a fine grained way through the ontology evolution process, allowing for personalization. We have demonstrated the applicability of our process model in a cross-organizational case study in the realm of tourism industry and another in the judicial domain. Real users were using the ontology to satisfy their information needs for an extended period of time.

References

1. V. R. Benjamins, D. Fensel, S. Decker, and A. Gómez-Pérez. $(KA)^2$: Building ontologies for the internet. *International Journal of Human-Computer Studies (IJHCS)*, 51(1):687–712, 1999.
2. K. Dellschaft, H. Engelbrecht, J. M. Barreto, S. Rutenbeck, and S. Staab. Cicero: Tracking design rationale in collaborative ontology engineering. In *ESWC*, volume 5021 of *Lecture Notes in Computer Science*, pages 782–786. Springer, Berlin, 2008.
3. A. Farquhar et al. The ontolingua server: A tool for collaborative ontology construction. Technical report KSL 96–26, Stanford, 1996.
4. A. Gómez-Pérez, M. Fernández-López, and O. Corcho. *Ontological Engineering*. Advanced Information and Knowlege Processing. Springer, Berlin, 2003.
5. C. W. Holsapple and K. D. Joshi. A collaborative approach to ontology design. *Communications of the ACM*, 45(2):42–47, 2002.
6. K. Kotis, G. A. Vouros, and Jerónimo Padilla Alonso. HCOME: Tool-supported methodology for collaboratively devising living ontologies. In *SWDB'04: Second International Workshop on Semantic Web and Databases 29–30 August 2004 Co-located with VLDB*. Springer, Berlin, 2004.
7. A. Maedche, B. Motik, and L. Stojanovic. Managing multiple and distributed ontologies on the semantic web. *The VLDB Journal*, 12(4):286–302, 2003.
8. N. Noy, A. Chugh, W. Liu, and M. A. Musen. A framework for ontology evolution in collaborative environments. In *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 544–558. Springer, Berlin, 2006.
9. A. Pease and J. Li. Agent-mediated knowledge engineering collaboration. In L. van Elst, V. Dignum, and A. Abecker, editors, *Agent-Mediated Knowledge Management International Symposium AMKM 2003 Stanford, CA, USA*, Lecture Notes in Artificial Intelligence (LNAI) 2926, pages 405–415. Springer, Berlin, 2003.
10. H. S. Pinto and J. P. Martins. A methodology for ontology integration. In *Proceedings of the First International Conference on Knowledge Capture (K-CAP2001)*, pages 131–138. ACM Press, New York, 2001.

11. H. S. Pinto, S. Staab, Y. Sure, and C. Tempich. OntoEdit empowering SWAP: A case study in supporting DIstributed, Loosely-controlled and evolvInG Engineering of oNTologies (DILIGENT). In C. Bussler, J. Davies, D. Fensel, and R. Studer, editors, *First European Semantic Web Symposium, ESWS 2004*, volume 3053 of *LNCS*, pages 16–30, Heraklion, Crete, Greece, May. Springer, Berlin, 2004.
12. H. S. Pinto, S. Staab, and C. Tempich. DILIGENT: Towards a fine-grained methodology for DIstributed, Loosely-controlled and evolvInG Engineering of oNTologies. In R. L. de Mántaras and L. Saitta, editors, *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004)*, pages 393–397, Valencia, Spain, August 2004. IOS Press, Amsterdam, 2004.
13. H. Sofia Pinto and J. P. Martins. Evolving Ontologies in Distributed and Dynamic Settings. In D. Fensel, F. Giunchiglia, D. L. McGuiness, and M.-A. Williams, editors, *KR2002 Proceedings*. Morgan Kaufmann, San Fransisco, CA, 2002.
14. T. Pirlein and R. Studer. An environment for reusing ontologies within a knowledge engineering approach. *International Journal of Human-Computer Studies*, 43(5):945–965, 1995.
15. M. I. Sucasas, C. Caracciolo, C. Baldassarre, and Y. Jaques. Revised specifications of user requirements for the Fisheries case study. NeOn deliverable 7.1.2, Food and Agriculture Organization of the United Nations (FAO), 2008.
16. Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke. Ontoedit: Collaborative ontology development for the semantic web. In *International Semantic Web Conference*, volume 2342 of *Lecture Notes in Computer Science*, pages 221–235. Springer, Berlin, 2002.
17. Y. Sure, S. Staab, and R. Studer. On-to-knowledge methodology. In S. Staab and R. Studer, editors, *Handbook on Ontologies in Information Systems*. Springer, Berlin, 2004.
18. C. Tempich, H. S. Pinto, S. Staab, and Y. Sure. A case study in supporting DIstributed, Loosely-controlled and evolvInG Engineering of oNTologies (DILIGENT). In K. Tochtermann and H. Maurer, editors, *Proceedings of the 4th International Conference on Knowledge Management (I-KNOW'04)*, pages 225–232, Graz, Austria, June 30–July 02 2004. Journal of Universal Computer Science (JUCS).
19. C. Tempich, H. S. Pinto, Y. Sure, and S. Staab. An argumentation ontology for DIstributed, Loosely-controlled and evolvInG Engineering processes of oNTologies (DILIGENT). In C. Bussler, J. Davies, D. Fensel, and R. Studer, editors, *Second European Semantic Web Conference, ESWC 2005*, LNCS, Heraklion, Crete, Greece, May. Springer, Berlin, 2005.
20. C. Tempich. *Ontology Engineering and Routing in Distributed Knowledge Management Applications*. PhD thesis, Karlsruhe University, 2006.
21. C. Tempich, H. S. Pinto, and S. Staab. Ontology engineering revisited: An iterative case study. In *Proceedings of the 3rd European Semantic Web Conference*, 2006.
22. C. Tempich, H. S. Pinto, Y. Sure, D. Vrandecic, N. Casellas, and P. Casanovas. Evaluating DILIGENT Ontology Engineering in a Legal Case Study. In P. Casanovas, P. Noriega, D. Bourcier, and V. R. Benjamins, editors, *IVR 22nd World Congress – Law and Justice in a Global Society*. International Association for Philosophy of Law and Social Philosophy, 2005.

23. C. Tempich, E. Simperl, H. S. Pinto, M. Luczak, and R. Studer. Argumentation-based ontology engineering. *IEEE Intelligent Systems*, 22:52–29, 2007.
24. 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.
25. D. Vrandečić, H. S. Pinto, Y. Sure, and C. Tempich. The diligent knowledge processes. *Journal of Knowledge Management*, 9(5):85–96, 2005.