User-Driven Ontology Evolution Management

Ljiljana Stojanovic[1], Alexander Maedche[1], Boris Motik[1], Nenad Stojanovic[2]

[1] FZI - Research Center for Information Technologies at the University of Karlsruhe,
Haid-und-Neu-Str. 10-14, D-76131 Karlsruhe, Germany
{Ljiljana.Stojanovic,Alexander.Maedche,Boris.Motik}@fzi.de
[2] Institute AIFB, University of Karlsruhe,
76128 Karlsruhe, Germany
nst@aifb.uni-karlsruhe.de

Abstract. With rising importance of knowledge interchange, many industrial and academic applications have adopted ontologies as their conceptual backbone. However, industrial and academic environments are very dynamic, thus inducing changes to application requirements. To fulfill these changes, often the underlying ontology must be evolved as well. As ontologies grow in size, the complexity of change management increases, thus requiring a well-structured ontology evolution process. In this paper we identify a possible six-phase evolution process and focus on providing the user with capabilities to control and customize it. We introduce the concept of an evolution strategy encapsulating policy for evolution with respect to user's requirements.

1 Introduction

With rising importance of knowledge interchange, many industrial and academic applications have adopted ontologies as their conceptual backbone. However, business dynamics and changes in the operating environment often give rise to continuous changes to application requirements, that may be fulfilled only by changing the underlying ontologies [16]. This is especially true for WWW and Semantic Web applications [2], that are based on heterogeneous and highly distributed information resources and therefore need efficient mechanisms to cope with changes in the environment.

Ontology evolution is the timely adaptation of an ontology to changed business requirements, to trends in ontology instances and patterns of usage of the ontology-based application, as well as the consistent management/propagation of these changes to dependent elements. A modification in one part of the ontology may generate subtle inconsistencies in other parts of the same ontology, in the ontology-based instances as well as in depending ontologies and applications [11]. This variety of causes and consequences of the ontology changes makes ontology evolution a very complex operation that should be considered as both, an organizational and a technical process [22]. It requires a careful analysis of the types of the ontology changes [13] that can trigger evolution as well as the environment in which the whole ontology evolution process is realized [25].

A. Gómez-Pérez and V.R. Benjamins (Eds.): EKAW 2002, LNAI 2473, pp. 285-300, 2002.

Although evolution over time is an essential requirement for successful application of ontologies [6], methods and tools to support this complex task completely are missing. This level of ontology management is necessary not only for the initial development [8] and maintenance of ontologies, but is essential during deployment, when scalability, availability, reliability and performance are absolutely critical [17].

In this paper we analyze ontology evolution requirements and present a novel, process-oriented approach that fulfils them. We specifically focus on the problem that the ontology has to remain consistent under complex changes during evolution. As for some changes there may be several different consistent states of the ontology, we introduce the notion of an evolution strategy allowing the user to customize the process according to her needs. Consequently, the user can transfer the ontology in the desired consistent state. Finally, we substantiate our discussion on ontology evolution by presenting its current implementation within KAON[1] framework.

The paper is organized as follows: Section 2 identifies the requirements for the ontology evolution and derives an ontology evolution process that fulfils them. Section 3 explores the complexity of the semantics of change problem and introduces different evolution strategies that allow user to control and to customize the evolution process. As a proof of the concept, section 4 contains a short description how ontology evolution been implemented. After a discussion of related work, concluding remarks outline some future work.

2 Ontology Evolution Requirements

Based on our experience in building ontologies and using them in several applications [25], we have formulated the following set of design requirements for ontology evolution:

- It has to enable resolving the given ontology changes [7] and to ensure the consistency of the underlying ontology and all dependent artifacts [24];
- It should be supervised allowing the user to manage changes more easily [28];
- It should offer advice to user for continual ontology refinement [19].

The first requirement is an essential one for any ontology evolution approach – after applying a change to a consistent ontology, the ontology should remain in consistent state. The second requirement complements the first one by presenting the user with information needed to control changes and make appropriate decisions. The last one states that potential changes improving the ontology may be discovered semi-automatically from ontology-based data and through analysis of user's behavior.

More careful analysis of these requirements (e.g. the changes have to be captured, analyzed, applied and validated by the user) implies the necessity to consider the ontology evolution problem as a composition of several subproblems realized in a determined sequence. This sequence of activities, resolving ontology changes in a composite way, is called the ontology evolution process. Consequently, the system, i.e. software, which copes with the ontology evolution problem has to be process-

[1] http://kaon.semanticweb.org

based, following currently the most popular programming paradigm in the business software development.

2.1 Resolving Changes While Keeping Consistency

Consistency requirement states that after applying and resolving changes in an ontology already in a consistent state[2], the ontology, its instances and dependent ontologies/applications must remain in (another) consistent state. This requirement encompasses two crucial aspects of the ontology evolution: enabling resolution of changes and maintenance the consistency of the system, and may be realized through following four phases as shown in Figure 1.

Change Representation. To resolve changes, they have to be identified and represented in a suitable format [13, 20]. Elementary changes in the ontology are derived from our ontology definition given in [25] specifying fine-grained changes that can be performed in the course of ontology evolution. However, this granularity of ontology evolution changes is not always appropriate. Often, intent of the changes may be expressed on a higher level. For example, the may need to generate a common superconcept sc of two concepts c_1 and c_2. He may bring the ontology into desired state through successive application of a list of elementary changes, such as 'Add_concept sc', 'Delete_SubConceptOf relation from c_1 to its current parent', 'Add_SubConceptOf relation from c_1 to sc', 'Delete_SubConceptOf relation from c_2 to its current parent' and 'Add_SubConceptOf relation from c_2 to sc'. However, this has significant drawbacks:

- There is an impedance mismatch between the intent of the request and the way the intent is achieved. It is required to create a superconcept of two concepts, but one needs to translate this operation into five separate steps, making the whole process error prone.
- A lot of unnecessary changes may be performed if each change is applied alone. For example, removing sub-concept-of relation from c_1 may introduce changes to property instantiations that should be reversed when assign sub-concept-of relation from c_1 to sc.

To avoid these drawbacks, it should be possible to express changes on a more coarse level, with the intent of change directly visible. Composite changes, representing a group of elementary changes applied together, are shown in the table 1.

Above mentioned changes are represented as instances of an evolution ontology – a special ontology which explicitly represents semantic information about ontology entities, changes in the ontology and mechanisms to discover and resolve changes. Detailed discussion of this ontology is out of scope of this paper and is given in [13].

[2] A consistent state of an ontology is the state in which all constraints, which are defined on the structure and content of an ontology are satisfied. An example of the structural constraints is the need to define the domain and the range for each relation in the ontology. Content constraints are related to the axioms in the ontology.

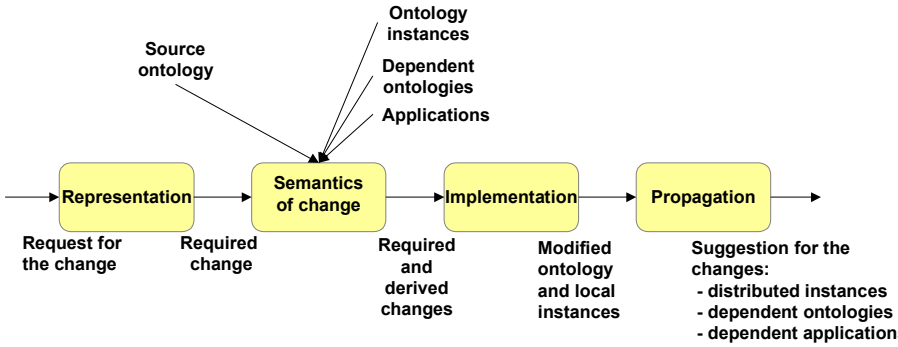

Figure 1. Four Elementary Phases of Ontology Evolution Process

Table 1. Composite changes in the ontology

Composite change	Description
Merge concepts	Replace several concepts with one and aggregate all instances.
Extract subconcepts	Split a concept into several subconcepts and distribute properties among them.
Extract superconcept	Create a common superconcept for a set of unrelated concepts and transfer common properties to it.
Extract related concept	Extract related information into a new concept and relate it to the original concept.
Shallow concept copy	Duplicate a concept with all its properties.
Deep concept copy	Recursively apply shallow copy to all subconcepts of a concept.
Pull up properties	Move properties from a subconcept to a superconcept.
Pull down properties	Move properties from a superconcept to a subconcept.
Move properties	Move properties from one concept to another concept.
Shallow property copy	Duplicate a property with same domain and range.
Deep property copy	Recursively apply shallow copy to all subproperties of a property.
Move Instance	Moves an instance from one concept to another.

Semantics of Change. Application of an elementary change in the ontology can induce inconsistencies in other parts of the ontology. We distinguish syntax and semantic inconsistency. Syntax inconsistency arises when undefined entities at the ontology or instance level are used or ontology model constraints are invalidated. Semantic inconsistency arises when meaning of an entity is changed due to changes performed in the ontology [29].

For example, removal of a concept which is the only element of domain set for some property results in syntax inconsistency [9]. Resolving that problem is treated as a request for a new change in the ontology, which can induce new problems that cause new changes and so on. Therefore, one change can potentially trigger other changes and so on. If an ontology is large, it may be difficult to fully comprehend the extent and meaning of each induced change. The task of 'semantics of change' phase is to enable resolution of induced changes in a systematic manner, ensuring consistency of the whole ontology. To help in better understanding of effects of each change, this phase should contribute maximum transparency providing detailed

insight into each change being performed. Some mechanisms used in this phase are described in the section 3.

In the course of evolution, actual meaning of concepts often shifts to better represent the structure of the real world. While some shifts of concept meaning are performed explicitly, a meaning of a concept can sometimes shift implicitly through changes in other parts of the ontology. For example, consider an ontology describing a relationship between jaguars and persons. In this ontology the meaning of the concept *Jaguar* is clear through the existence of the property *eats* that links *Jaguar*s and *Person*s – it is obvious that concept *Jaguar* stands for an animal from the feline family. For any reason one may delete the concept *Person*, which may result in the removal of the property *eats* as well. After the change is performed, the semantics of concept *Jaguar* is not clear any more – is it a Jaguar cat or a Jaguar car? These kinds of ambiguities can be eliminated in several ways. The simplest solution is by introducing a superconcept *Animal* before the change is performed. However, if the ontology is large, such issues may be easily overlooked because it is very hard to keep the complete ontology structure in mind at once.

This problem can be avoided using a richer description [13] determining semantic role of ontology entities. By attaching meta-information about e.g. essential properties of a concept [29], deeper knowledge about concept meaning is provided. Moreover, semantic ambiguities of ontology entities may be resolved through additional documentation, such as who is the author of an entity, what is the purpose of introducing an entity etc. Contrary to meta-information determining the semantic role of ontology entities, "documentation" meta-information cannot be used for formal consistency checking.

Change Implementation. In order to avoid performing undesired changes, before applying a change to the ontology, a list of all implications to the ontology should be generated and presented to the user [28]. He should be able to comprehend the list and approve or cancel the change. When the changes are approved, they are performed by successively resolving changes from the list. If changes are cancelled, the ontology should remain intact. This is more elaborated in description of implementation in section 4.

Change Propagation. When the ontology is modified, ontology instances need to be changed to preserve consistency with the ontology [9]. This can be performed in three steps. If the instances are on the Web they are collected in the knowledge base [14]. In the second step, modification of instances is performed according to the changes in the ontology [23]. In the last step "out-of-date" instances on the Web are replaced with corresponding "up-to-date" instances.

Ontologies often reuse and extend other ontologies. Therefore, an ontology update might also corrupt ontologies that depend on the modified ontology and consequently, all artifacts that are based on these ontologies. This problem can be solved by recursive applying the ontology evolution process on these ontologies.

When an ontology is changed, applications based on the changed ontology may not work correctly. An ontology evolution system has to recognize which change in the ontology can affect the functionality of dependent applications [10, 21] and to react

correspondingly. More information about possible problems in this phase and ways for solving them are given in [24].

2.2 User's Management of Changes

There are numerous circumstances where it may be desired to reverse the effects of ontology evolution, to name just a few:

- The ontology engineer may fail to understand the actual effect of the change and approve the change that shouldn't be performed.
- It may be desired to change the ontology for experimental purposes.
- When working on an ontology collaboratively, different ontology engineers may have different ideas about how the ontology should be changed.

In order to enable recovering from these situations, we introduce the validation phase in the ontology evolution process (see Figure 2). It enables validation of performed changes and undoing them at user's request. It is important to note that reversibility means undoing all effects of some change, which may not be the same as simply requesting an inverse change manually. For example, if a concept is deleted from a concept hierarchy, its subconcepts will need to be either deleted as well, attached to the root concept, or attached to the parent of the deleted concept. Reversing such a change is not equal to recreating the deleted concept – one needs, also, to revert the concept hierarchy into original state.

The problem of reversibility is typically solved by creating evolution logs. An evolution log tracks information about each change in the system, allowing to reconstruct the sequence of changes leading to current state of the ontology. With each change evolution logs additionally associate following information [13]:

- Meta-information such as change description, cost of change, time of change, cause of the change etc.,
- Identity of the change author.

2.3 Continual Improvement

In ontology evolution we may distinguish two types of changes: top-down and bottom-up, whose generation is part of the "capturing phase" in the ontology evolution process. Top-down changes are explicit changes, driven, for example, by top-manager who want to adapt the system to new requirements and can be easily realized by an ontology evolution system. However, some changes in the domain are implicit, reflected in the behavior of the system and can be discovered only through analysis of its behavior. For example, if a customer group doesn't contain members for a longer period of time, it may mean that it can be removed. This second type of change mined from the set of ontology instances are called bottom-up changes.

Another source of bottom-up changes is the structure of the ontology itself [19]. Indeed, the previously described "validation phase" results in an ontology which may be in a consistent state, but contains some redundant entities or can be better structured with respect to the domain. For example, multiple users may be working on

different parts of an ontology without enough communication. They may be deleting subconcepts of a common concepts at different points in time to fulfill their immediate needs. As a result, it may happen that only one subconcept is left. Since classification with only one subclass beats the original purpose of classification, we consider such ontology to have a suboptimal structure. Moreover, based on heuristics and/or data mining algorithms [12], suggestions for changes that refine ontology structure may be induced by analysis of patterns of ontology usage. By tracking when concept has last been retrieved by a query, it may be possible to discover that some concepts are out of date and should be deleted or updated. To aid users in detecting such situations, we investigated the possibilities of applying the self-adaptive systems principles and proactively make suggestions for *ontology refinements* – changes to the ontology with the goal of improving ontology structure, making the ontology easier to understand and cheaper to modify.

2.4 The Overall Ontology Evolution Process

Complete ontology process derived from our discussion of ontology evolution requirements is presented in Figure 2. It has a cyclic structure, since validation of realized changes may (automatically) induce new changes in order to obtain model consistency or to satisfy users' expectations. The first requirement from section 2 for ontology consistency results in phases 2 to 5, the second requirement for user supervision results in phase 6 and the third requirement for continual ontology refinement results in phase 1.

3 Evolution Strategy

As mentioned, the role of "semantics of change" phase in ontology evolution process if to figure out which elementary changes need to be performed for one change request, e.g. deletion of a concept. If this were left to the user, evolution process would be too error-prone and time consuming – it is unrealistic to expect that humans will be able to comprehend entire ontology and interdependencies in it [28]. This requirement is especially hard to fulfill if the rationale behind domain conceptualization is ambiguous or if the user does not have the experience. There are many ways to achieve consistency after a change request. For example, when a concept from the middle of the hierarchy is being deleted, all subconcepts may either be deleted or reconnected to other concepts. If subconcepts are preserved, then properties of the deleted concept may be propagated, its instances distributed, etc. Thus, for each change in the ontology, it is possible to generate different sets of additional changes, leading to different final consistent states. Most of existing systems for the ontology development [22] provide only one possibility for realizing a change and this is usually the simplest one. For example, the deletion of a concept always causes the deletion of all its subconcepts.

Thus, to resolve a change, the evolution process needs to determine answers at many *resolution points* – branch points during change resolution where taking a different path will produce different results. Each possible answer at each resolution

point is an *elementary evolution strategy*. Common policy consisting of a set of elementary evolution strategies, each giving an answer for one resolution point, is an *evolution strategy* and is used to customize the ontology evolution process. Thus, an evolution strategy unambiguously defines the way how elementary changes will be resolved [3]. Typically a particular evolution strategy is chosen by the user at the start of the ontology evolution process.

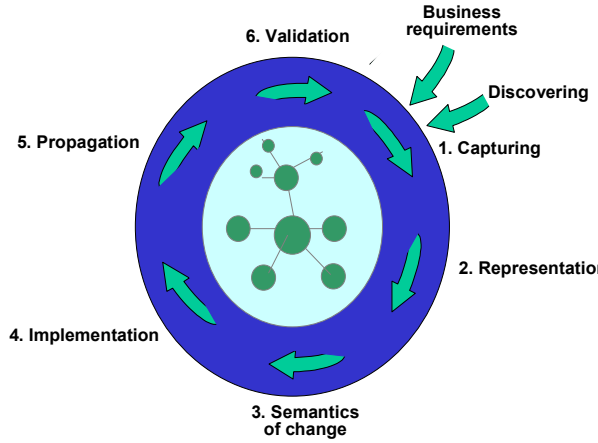

Figure 2. Ontology Evolution Process

To derive the set of resolution points within an evolution strategy, we started by considering types of changes that may be applied to an ontology. Next we analyzed what consequences can each change have on the ontology with respect to its definition [25] and dependencies between ontology entities. We isolated changes that can provoke syntax inconsistencies and, consequently, cannot be applied. For example, "Add_SubConceptOf" change is not allowed if it causes an inheritance hierarchy cycles. Further, we identified that some changes can generate the need for subsequent changes, some of them offering different ways of resolution. For each particular resolution way we defined an elementary evolution strategy. For each elementary change we defined an algorithm containing resolution points encountered during change resolution. Each resolution point represents a branching point, and each elementary evolution strategy represents one possible branch. The choice of exactly one elementary evolution strategy for each possible resolution point forms an evolution strategy.

3.1 Evolution Strategy Example

Let us explain our approach through an example of deleting a concept C embedded in a complex concept hierarchy. In order to keep the ontology in a consistent state, following resolution points may be observed:

- what to do with orphaned subconcepts of C;
- what to do with properties that subconcepts of C inherit from C's parents;

- what to do with all properties whose domain is C;
- what to do with the properties whose range is C;
- what to do with instances of C;
- what to do with instances of other concepts having relations with instances of C.

For each of these resolution points, there is a set of elementary evolution strategies defining possible options. E.g., in case of the first resolution point, as illustrated in Figure 3, orphaned subconcepts of C may be:

- connected to the parent concept(s) of C;
- connected to the root concept of the hierarchy;
- deleted as well.

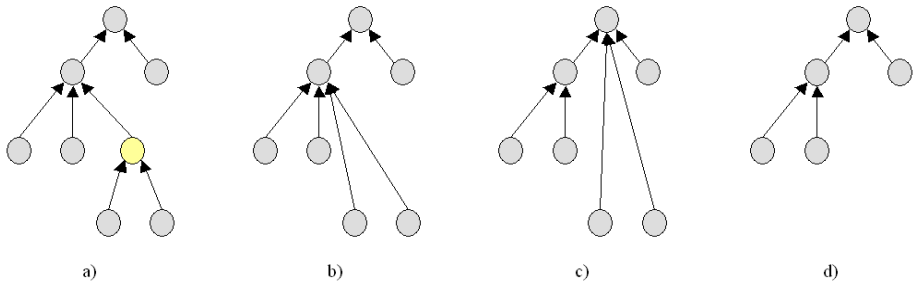

Figure 3. Resolution Points for Deleting Concept C: a) Original ontology; b) Connection to the parent concept; c) Connection to the root concept; d) Deletion of the subconcepts

Similarly we may elicit remaining elementary strategies for all mentioned resolution points. The part of the algorithm for deletion of a concept with corresponding resolution points and available elementary evolution strategies is given in Figure 4.

3.2 Advanced Evolution Strategies

In real business the choice of how a change (e.g. deletion of a concept) should be resolved may be based on characteristics of the final state of the ontology (e.g. make depth of hierarchy as small as possible) or on characteristics of the process for resolving changes itself (e.g. incur minimal cost of changes).

In order to enable such customization of the ontology evolution process, the user may choose an *advanced evolution strategy*. It represents a mechanism to priorities and arbitrate among different evolution strategies available in a particular situation, relieving the user of choosing elementary evolution strategies individually.

Advanced evolution strategy automatically combines available elementary evolution strategies to satisfy user's criteria. We have identified the following set of advanced evolution strategies:

- ***structure-driven strategy*** – resolves changes according to criteria based on the structure of the resulting ontology, e.g. the number of levels in concept

hierarchy. This strategy follows the requirements of the real-word ontology-based applications, e.g. MEDLINE[3]. MEDLINE requires a weekly update, usually involving only supplementary concept records. However, concept hierarchy is updated annually. This kind of changes is performed by keeping the hierarchy minimal, because it alleviates, according to the authors of MEDLINE, the understanding of the conceptualization.

- ***process-driven strategy*** – resolves changes according to process of changes itself, for example optimized per cost[4] of the process or per a number of steps involved[5]. Determining what has to be change and how to change it requires a deep understanding of how the ontology entities interact one with another. We cannot expect that the user spends time explaining the reasons for all performed changes and their ordering. One strategy enabling that the user can easily follow and understand sequences of the changes is to perform the minimal number of the updates.
- ***instance-driven strategy*** – resolves changes to achieve an explicitly given state of the instances. This relieves the user of the necessary to newly add or redistribute the instances, which can be time consuming and error prone task. An efficient instance-driven evolution strategy should analyze the difference between the initial and final state of instances and try to achieve final state in the most efficient manner. The process to achieve that is based on logical inference [5] and its description is out of scope for this paper.
- ***frequency-driven strategy*** – applies the most used or last recently used evolution strategy.

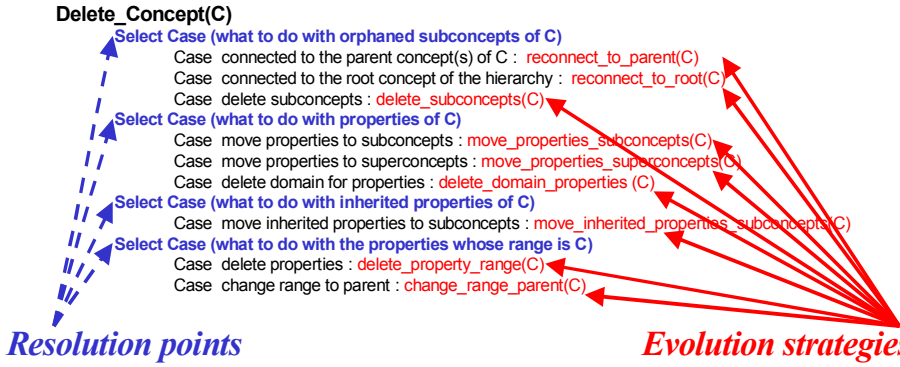

Figure 4. Algorithm for Concept Deletion

[3] http://www.nlm.nih.gov/pubs/factsheets/medline.html

[4] Cost may be defined by the number of instances that must be updated.

[5] The application of process-driven evolution with the minimal number of the changes for the ontology shown in the figure 3a) results in solution (d). Since instances are missing from the figure, a cost-based evolution strategy cannot be chosen.

4 Implementation

The Karlsruhe Ontology and Semantic Web framework (KAON) has been developed at the University of Karlsruhe. It is used as a basis for several ontology-enabled research and industry projects. It's primary goal is to establish a platform needed to apply Semantic Web technologies to e-commerce scenarios, knowledge management, automatic generation of Web portals, E-Learning etc. The simplified conceptual architecture of KAON emphasizing points of interest related to ontology evolution is presented in Figure 5.

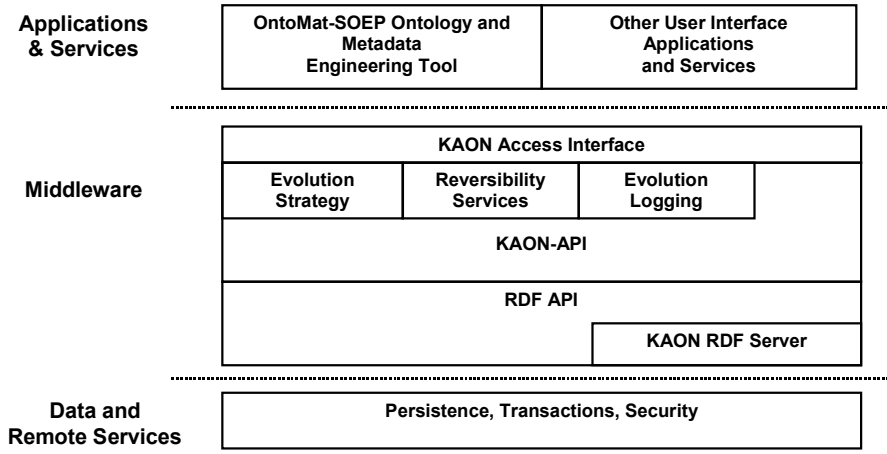

Figure 5. Conceptual KAON Architecture with Respect to Ontology Evolution

Roughly, KAON components can be divided into three layers:

- Applications and Services Layer realizes UI applications and provides interfaces to non-human agents. Among many applications realized, OntoMat-SOEP provides ontology and metadata engineering capabilities. It realizes many requirements related to ontology evolution and is described next in more detail.
- KAON API as part of the Middleware Layer is the focal point of KAON architecture since it realizes the model[6] of ontology based applications. The bulk of requirements related to ontology evolution is realized in this layer and is described in the next section.
- Data and Remote Services Layer provides data storage facilities. This layer also realizes concurrency and transactional atomicity of updates. Further elaboration of this layer is out of scope for this paper.

[6] The term model refers to the model component of the Model-View-Controller architectural pattern.

4.1 Ontology Evolution in KAON API

Before the ontology evolution process is started, a particular evolution strategy must be selected. Changes to the ontology are performed by assembling elementary and composite changes into a sequence. However, before the ontology is actually updated, this sequence is passed to the present evolution strategy to perform the steps described in section 3 in the "semantics of change" phase, resulting in an extended sequence of changes. To ensure atomicity of updates, either all or no change from the extended sequence of changes should succeed, so validity of change sequence is checked before any updates are actually performed. Transparency is realized by presenting the extended sequence of changed to the user for approval. To further aid the understanding of why some changes are performed, the evolution strategy may group related elementary actions and provide explanations why particular change is necessary, thus greatly increasing the chances that all side-effects of changes will be properly understood. After changes are reviewed by the user, they are passed to the ontology and executed, performing steps from the "change implementation" phase.

It is obvious that for each elementary change there is exactly one inverse change that, when applied, reverses the effect of the original change. With such infrastructure in place, it is not hard to realize the reversibility requirement: to reverse the effect of some extended sequence of changes, a new sequence of inverse changes in reverse order needs to be created and applied.

As mentioned in section 2, the evolution log needs to associate additional information with each change. Effectively, the log is treated as an instance of a special evolution ontology [13] consisting of concepts for each change, making it is easy to add meta-information to log entries. Structure of the log may be easily customized by editing the evolution ontology. Evolution logging and reversibility services are provided as special services of KAON API, allowing different applications reuse these powerful features. E.g., actions performed in one application may be easily reverted in another.

4.2 Ontology Evolution in KAON Applications

As mentioned in the previous section, ontology evolution is primarily realized through KAON API. However, UI applications provide human-computer interaction for evolution, whose primary role is to present change information in an orderly way, allowing easy spotting of potential problems. Also, any application that changes the ontology must realize the reversibility requirement in its user interface as well. Currently evolution requirements are realized within the OntoMat-SOEP ontology and metadata engineering tool, as follows:

- As shown in left part of Figure 6, users may set up the desired evolution strategy which consists of four resolution points. For each resolution point the user must choose appropriate elementary evolution strategy.
- Before changes are performed, their impact is reported to the user (the right part of Figure 6). Presentation of changes follows the progressive disclosure principle: related changes are grouped together and organized in a tree-like form. The user initially sees only the general description of changes. If he is

interested in details, he can expand the tree and view complete information. He may cancel the operation before it is actually performed.

- An unlimited undo-redo function is provided. Although is this function by large the responsibility of the KAON API, the user interface is responsible for restoring the visual context after an undo operation.

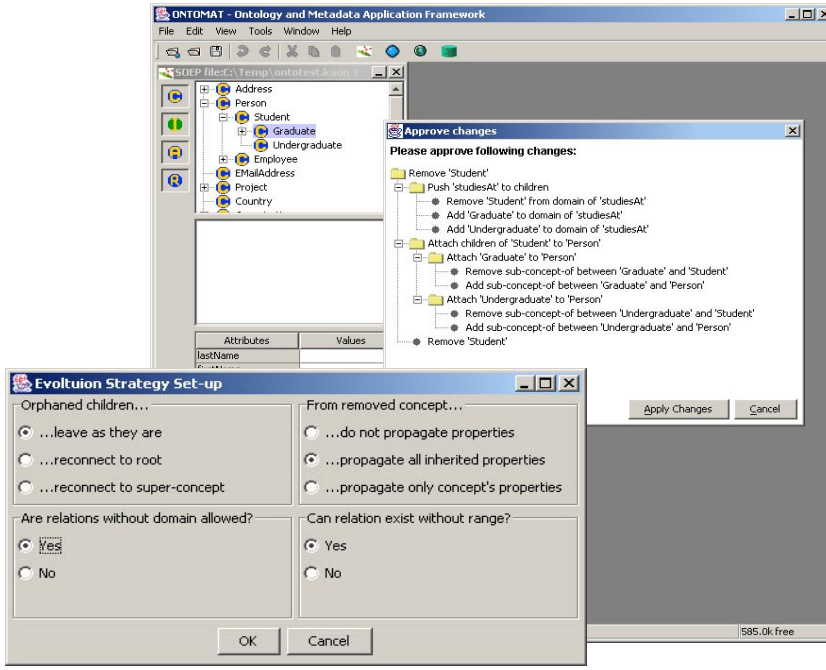

Figure 6. Ontology Evolution in KAON framework: Evolution Strategy Set-up and Ontology Evolution User Interface in OntoMat-SOEP

A sample screenshot of OntoMat-SOEP is given in Figure 6. In this scenario, the user requested to remove *Student* concept. The evolution strategy decided to push property *studiesAt* of that concept to children. By opening a node in the tree, the user can see what changes will actually be performed. Hence, the change information can be viewed at different levels of granularity. Similarly, the strategy decided that children of the concept will be attached to parent of the deleted concept. For each child a detailed list elementary changes needed to achieve that is presented.

5 Related Work

In the last decade there has been very active research in the area of ontology engineering. The majority of researches in this area are focused on construction issues. However, coping with the changes and providing maintenance facilities require a different approach. We cannot say that there exist commonly agreed

methodologies and guidelines for ontology evolution. Thus, there are very few approaches investigating the problems of changing in the ontologies.

Heflin [9] points out that ontologies on the Web will need to evolve and he provides a new formal definition of ontologies for the use in dynamic, distributed environments. Although good design may prevent many ontological errors, some errors will not be realized until the ontology is put into use. However, this problem as well as the problem of the change propagation are not treated in the work of Heflin. Moreover, the user cannot customize the way of performing the change and the problem of the identification of the change is not analysed.

In contrast to the ontology evolution that allows access to all data only through the newest ontology, the ontology versioning allows access to data through different versions of the ontology. Thus, the ontology evolution can be treated as a part of the ontology versioning mechanism that is analysed in [11]. Authors provide an overview of causes and consequences of the changes in the ontology. However, the most important flaw is the lack of a detailed analysis of the effect of specific changes on the interpretation of data which is a constituent part of our work.

Oliver et al. [20] discuss the kinds of changes that occur in medical ontologies and propose the CONCORDIA concept model to cope with these changes. The main aspects of CONCORDIA are that all concepts have a permanent unique identifier. Concepts are given a *retired* status instead of being physically deleted. Moreover special links are maintained to track the retired parents and children of each concept. However, this approach is insufficient for managing a change on the Semantic Web especially while there are no possibilities to control the whole process.

In [16] the author presents the guiding principles for building consistent and principled ontologies in order to alleviate their creation, the usage and the maintenance in the distributed environments. Authors analyse the requirements for the tool environments that enforces consistency. Many of these operational guidelines are included (and implemented) in our solution.

[29] presents an extended ontology knowledge model that represents semantic information about concepts explicitly. However, this enriched semantic is not used for supporting evolution problems, but to describe what is known by agents in a multi-agent system.

Other research communities also have influenced our work. The problem of schema evolution and schema versioning support has been extensively studied in relational and database papers ([1], [21]). In [18] authors discuss the differences that steam from different knowledge models and different usage paradigms. Moreover, research in ontology evolution can also benefit from the many years of research in knowledge-based system evolution [3, 15]. The script-based knowledge evolution [28] that identifies typical sequences of changes to knowledge base and represents them in a form of scripts, is similar to our approach. In contrast to the knowledge-scripts that allow the tool to understand the consequences of each change, we go step further by allowing the user to control how to complete the overall modification and by suggesting the changes that could improve the ontology.

6 Conclusion

In this paper we presented a novel approach for dealing with ontology changes. The approach is based on a six-phase evolution process, which systematically analyses the causes and the consequences of the changes and ensures the consistency of the ontology and depending artifacts after resolving these changes. In order to enable the user to obtain the ontology most suitable to her needs, we specifically focus on the possibilities to customize the ontology evolution process. We identify two means to do that: (i) to enable the user to set up one of predefined or advanced evolution strategies that are used for resolving the changes and (ii) to suggest the user to generate some change, implied by the analysis of the structure of the ontology, ontology instances or user behaviors in the underlying ontology-based applications.

Although our implementation is in an early phase and therefore the real evaluation is missing, we made some experiments with one of our ontology-based applications, particular the AIFB portal [25]. Comparison of time needed to resolve ”per-hand” initiated change, shows the real necessity for the methodological support for the ontology evolution, even for the very experienced ontology engineers. Moreover, the detailed analysis of the possibility to use our approach in the case of highly-distributed Web applications, such MEDLINE, shows many benefits of the presented approach for the large-scale ontologies and motivates further research in that direction.

References

1. J. Banerjee, W. Kim, H.J. Kim, H. Korth, Semantics and implementation of schema evolution in object-oriented databases, In proceedings of the Annual Conference on Management of Data, pp- 211-322, ACM SIGMOD, May 1997.
2. T. Berners-Lee, *XML 2000 – Semantic Web talk*, 2000, http//www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html, 2000.
3. P. Breche, M. Wörner, *How to remove a class in an ODBS*, In ADBS’95, 2nd International Conference on Application Database, Santa Clara, California, 1995
4. A. Bultman, J. Kuipers and F. van Harmelen, *Maintenance of KBS’s by domain experts: The Holy Grail in Practice*, Lecture Notes in AI, IEA/AIE’00, 2000.
5. S. Decker, M. Erdmann, D. Fensel and R. Studer, *Ontobroker: Ontology based access to distributed and semi-structured information*, Meersman, R. et al. (Eds.), Database Semantics: Semantic Issues in Multimedia Systems, pp. 351–369. Kluwer Academic Publisher, 1999.
6. D. Fensel, *Ontologies: Dynamics Networks of Meaning*, In Proceedings of the the 1st Semantic web working symposium, Stanford, CA, USA, July 30th-August 1st, 2001.
7. E. Franconi, F. Grandi, and F. Mandreoli, *A semantic approach for schema evolution and versioning in object-oriented databases*, Proc. CL2000, 2000.
8. A. Gomez-Perez, *Ontological engineering: A state of the art*, Expert Update, 2(3):33-43, Autumn 1999.
9. J. Heflin, *Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment*, Ph.D. Thesis, University of Maryland, College Park. 2001.
10. W. Hürsch, *Maintaining consistency and behaviour of object-oriented systems during evolution*, In Proc. of the ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '97), Vol.32 No. 10, pp1-21, 1997.

11. M. Klein and D. Fensel, *Ontology versioning for the Semantic Web*, Proc. International Semantic Web Working Symposium, USA, July 30 - August 1, 2001.
12. A. Maedche and S. Staab, *Ontology Learning for the Semantic Web*, IEEE Intelligent Systems, 16(2), March/April 2001. Special Issue on Semantic Web, 2001.
13. A. Maedche, L. Stojanovic, R. Studer, R. Volz: *Managing Multiple Ontologies and Ontology Evolution in Ontologging*, In Proceedings of the Conference on Intelligent Information Processing, World Computer Congress 2002, Montreal, Canada, 2002.
14. A. Maedche, M. Ehrig, S. Handschuh, L. Stojanovic, R. Volz, *Ontology-Focused Crawling on Documents and Relational Metadata*, In Proceedings of the Eleventh International World Wide Web Conference WWW-2002, (Poster), Hawaii, 2002.
15. T. Menzis, Knowledge maintenance: The state of the art. The Knowledge Engineering Review, 10(2), 1998.
16. D. McGuinness, Conceptual Modeling for Distributed Ontology Environments, In the Proceedings of the ICCS 2000, August 14-18, Darmstadt, Germany , 2000.
17. D. McGuinness, R. Fikes, J. Rice, and S. Wilder, *An environment for merging and testing large ontologies*, In Proceedings of KR-2000. principle of Knowledge Representation and Reasoning. Morgan-Kaufman, 2000.
18. N. F. Noy, M. Klein, Ontology Evolution: Not the Same as Schema Evolution, SMI technical report SMI-2002-0926, 2002.
19. N. F. Noy, D. McGuinness, *Ontology Development 101: A Guide to creating your first Ontology*, Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001
20. D. E. Oliver, Y. Shahar, M. A. Musen, and E. H. Shortliffe, *Representation of change in controlled medical terminologies*, AI in Medicine,15(1):53–76, 1999.
21. J.F. Roddick, *A Survey of Schema Versioning Issues for Database Systems*, Information and Software Technology, 37(7):383-393, 1996.
22. S. Staab, H.-P. Schnurr, R. Studer and Y. Sure, *Knowledge Processes and Ontologies*, IEEE Intelligent Systems. 16(1), Jan./Feb. 2001. Special Issue on Knowledge Management, 2001.
23. L. Stojanovic, N. Stojanovic and R. Volz, *Migrating data-intensive Web Sites into the Semantic Web*, In Proceedings of the ACM Symposium on Applied Computing SAC, 2002.
24. L. Stojanovic, N. Stojanovic, S. Handschuh, *Evolution of the Metadata in the Ontology-based Knowledge Management Systems*, In Proceedings of Experience Management 2002, Berlin, 2002.
25. N. Stojanovic, A. Maedche, S. Staab, R. Studer and Y. Sure, *SEAL — A Framework for Developing SEmantic PortALs*, *ACM K-CAP 2001*. October, Vancouver, 2001.
26. N. Stojanovic, L. Stojanovic*, Searching for the Knowledge in the Semantic Web*, The 15th International FLAIRS Conference, Pensacola, Florida, May 14-16, 2002.
27. N. Stojanovic, L. Stojanovic: *Evolution in the ontology-based knowledge management system*. In Proceedings of the European Conference on Information Systems - ECIS 2002, Gdañsk, Poland, 2002.
28. M. Tallis, Y. Gil, *Designing Scripts to Guide Users in Modifying Knowledge-based Systems*, AAAI/IAAI 1999: 242-249
29. V.A.M. Tamma, T.J.M Bench-Capon, A conceptual model to facilitate knowledge sharing in multi-agent systems, In Proceedings of the OAS 2001. Montreal, pp. 69-76, 2001.