Chapter 11
Ontology Evolution

Raúl Palma, Fouad Zablith, Peter Haase, and Oscar Corcho

Abstract Ontologies are dynamic entities that evolve over time. There are several challenges associated with the management of ontology dynamics, from the adequate control of ontology changes to the identification and administration of ontology versions. Moreover, ontologies are increasingly becoming part of a network of complex relationships and dependencies, where they reuse and extend other ontologies, have associated metadata in order to ease sharing and reuse, are used to integrate heterogeneous knowledge bases, etc. Under these circumstances, a change in an ontology does not only affect the ontology itself but may also have consequences in all its related artifacts. In this chapter, we propose methodological guidelines for carrying out the ontology evolution activity. We target different scenarios, supporting users in the process of ontology evolution from a generic perspective and on how to use tools that semiautomatically assist them in discovering, evaluating, and integrating domain changes to evolve ontologies. To illustrate their applicability, we describe how such guidelines have been used in real example applications.

R. Palma (✉)
Poznan Supercomputing and Networking Center, ul Dabrowskiego 79a, 60-529 Poznan, Poland
e-mail: rpalma@man.poznan.pl

F. Zablith
Knowledge Media Institute (KMi), The Open University, Walton Hall, Milton Keynes MK7 6AA, UK
e-mail: f.zablith@open.ac.uk

P. Haase
fluid Operations AG, Altrottstr. 31, 69190 Walldorf, Germany
e-mail: peter.haase@fluidops.com

O. Corcho
Ontology Engineering Group, Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo sn, 28660 Boadilla del Monte, Madrid, Spain
e-mail: ocorcho@fi.upm.es

11.1 Motivation

Ontologies are fundamental building blocks of the Semantic Web and are often used as the knowledge backbones of advanced information systems. As such, the growing use and application of ontologies in many different areas during the last years has led to an increasing interest of both researchers and industry in the construction of ontologies and the reuse of existing ones. Reusing existing ontologies instead of creating new ones from scratch has many benefits: it lowers the time and cost of developing new ontologies, avoids duplicate efforts, eases interoperability, etc. As a consequence, complex networks of ontologies are being created where each ontology may depend on several others and may also be related to other artifacts (e.g., individuals, mappings, applications, and metadata).

Nevertheless, this situation also brings about new issues. Ontologies (like many other system components) are dynamic entities. An ontology, defined as a formal, explicit specification of a shared conceptualization (Studer et al. 1998), may change whenever any of the elements of this definition changes. For instance, domains are not static or fixed: they may evolve when non-existing elements become part of the domain or when some elements become obsolete, among others. Additionally, ontologies need to be kept up to date in order to reflect the changes that affect the life cycle of the underlying systems (e.g., changes in the underlying data sets, need for new functionalities, etc.). A similar situation occurs with shared conceptualizations, which may change, for example, when the domain experts involved in modeling acquire additional knowledge about the domain. Finally, the formal specification may change because new ontology languages or new versions of the existing ones become available, for example.

The management of ontology dynamics raises many challenges such as the identification and administration of different ontology versions or the flow control of ontology changes (i.e., when and how an ontology can change). Moreover, dealing with ontology changes involves the execution of many related tasks. Most of these tasks are already identified in the context of the ontology evolution process, defined in (Stojanovic 2004) as the timely adaptation of an ontology to the arisen changes and the consistent management of these changes. For example, among these tasks are the capturing and formal representation of ontology changes, the verification of the ontology consistency after the changes are performed, and the propagation of those changes to the ontology related entities. The distributed nature of a network of ontologies where complex relations can exist between ontologies and other artifacts demands the necessity to propagate ontology changes to the distributed ontology-dependent artifacts (e.g., related ontologies, ontology individuals, mappings, and metadata). For instance, a change in a wine ontology (e.g., add a new class for a type of wine) may require one or more updates in its related metadata (e.g., increase the number of classes by one, add an additional key class, add an additional contributor, and update the date of the last modification) or its mappings to other similar ontologies (e.g., create a new correspondence between the new class and another class representing the same type of wine in another ontology). Moreover, the ontology and its related artifacts may be distributed in different places across the web.

While it seems necessary to apply the ontology evolution activity consistently for most ontology-based systems, it is often a time-consuming and knowledge-intensive activity, as it requires a knowledge engineer to identify the need for change, perform appropriate changes on the base ontology, and manage its various versions. While existing evolution frameworks normally include a description of the life cycle, this description is neither meant nor suited to replace guidelines. Therefore, we propose here methodological guidelines for supporting ontology developers during the evolution of the ontologies and for supporting them in exploiting tools to facilitate the evolution of their ontologies.

It is worth noting that both ontology evolution and ontology versioning deal with the management of ontology changes. However, they differ in their focus: ontology evolution focuses on the modification of an ontology, possibly preserving its consistency, whereas ontology versioning focuses on creating and managing different versions of the ontology.

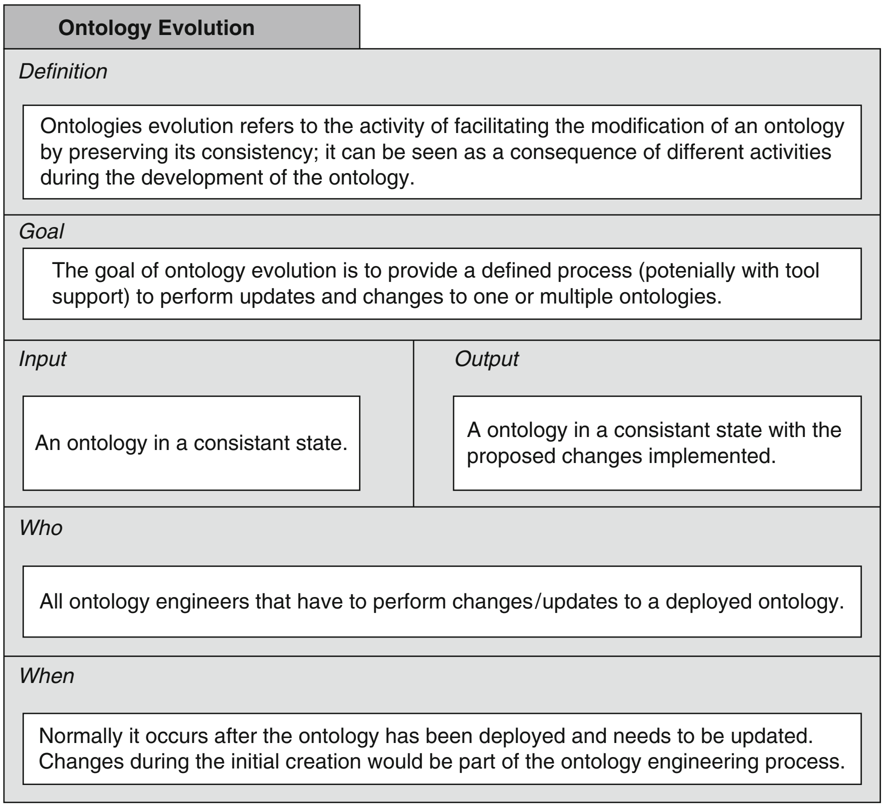
We argue that in order to provide a comprehensive support for ontology evolution, targeted at users and ontology engineers, we need two types of guidelines: one that guides users in the process of ontology evolution from a generic perspective and another that provides guidelines on how to use tools that semiautomatically assist users in discovering, evaluating, and integrating domain changes to evolve ontologies.

The remainder of this chapter is organized as follows: First, we introduce high-level guidelines for carrying out the ontology evolution activity. This would give an overall picture of the required tasks and possible options to handle each one, supported by an example in the fishery domain of FAO. Second, we provide guidelines for how to support users in exploiting and customizing tools that support users in evolving ontologies from external domain data using semiautomatic techniques. This is also supported by an applied example in the academic domain.

11.2 Guidelines for Ontology Evolution

In this section, we present the guidelines set out to help ontology developers in the ontology evolution activity. Such guidelines have been created in the context of the NeOn Methodology for building ontology networks. This methodology takes into account the existence of multiple ontologies in ontology networks, the collaborative ontology development, the dynamic dimension, and the reuse and reengineering of knowledge-aware resources.

According to the NeOn Glossary of Processes and Activities (Suárez-Figueroa and Gómez-Pérez 2008), ontology evolution refers to the activity of facilitating the modification of an ontology by preserving its consistency; it can be seen as a consequence of different activities during the development of the ontology.



Ontology Evolution

Definition

Ontologies evolution refers to the activity of facilitating the modification of an ontology by preserving its consistency; it can be seen as a consequence of different activities during the development of the ontology.

Goal

The goal of ontology evolution is to provide a defined process (potenially with tool support) to perform updates and changes to one or multiple ontologies.

Input

An ontology in a consistant state.

Output

A ontology in a consistant state with the proposed changes implemented.

Who

All ontology engineers that have to perform changes/updates to a deployed ontology.

When

Normally it occurs after the ontology has been deployed and needs to be updated. Changes during the initial creation would be part of the ontology engineering process.

Fig. 11.1 Ontology evolution filling card

Thus, in the framework of the NeOn Methodology we propose the filling card for the ontology evolution, presented in Fig. 11.1, which includes the definition, goal, input, output, who carries out the activity, and when the activity should be carried out.

11.2.1 Ontology Evolution Tasks

Figure 11.2 illustrates the methodological guidelines for carrying out the ontology evolution activity, showing the main tasks involved, their inputs, outputs, and actors. The tasks shown in the figure are explained below. They are based on the generic activities discussed in (Bennett and Rajlich 2000) for the process of making changes to any type of artifact that is subject to changes, customized to the case of ontology evolution (e.g., Leenheer and Mens 2007) in the context of ontology networks.

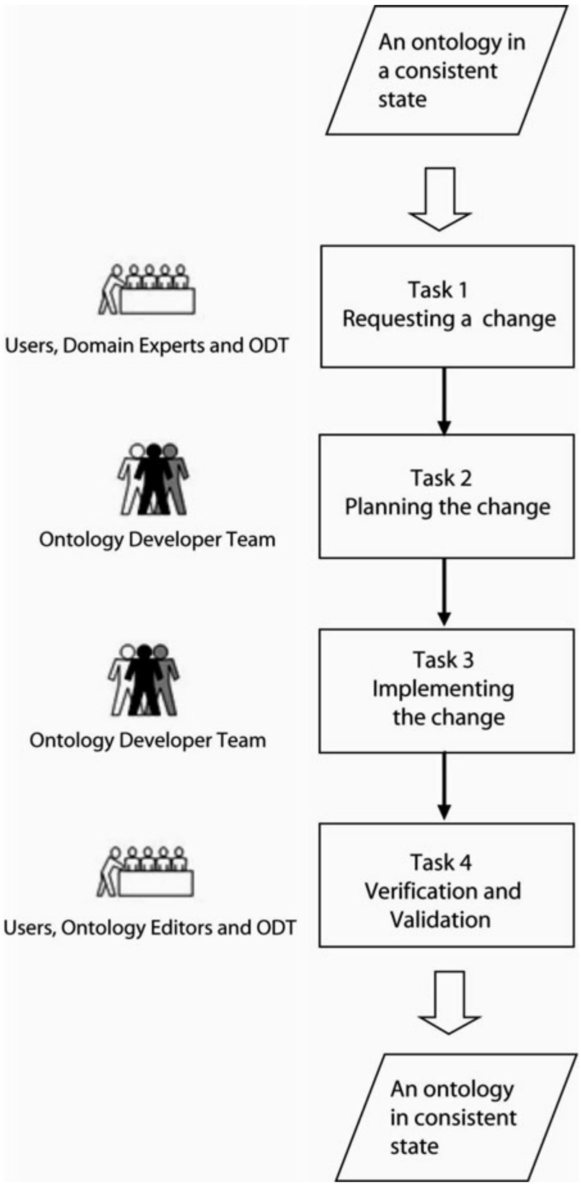

An ontology in a consistent state
Task 1 Requesting a change
Users, Domain Experts and ODT
Task 2 Planning the change
Ontology Developer Team
Task 3 Implementing the change
Ontology Developer Team
Task 4 Verification and Validation
Users, Ontology Editors and ODT
An ontology in consistent state

Fig. 11.2 Tasks for ontology evolution activity

Task 1 Requesting a Change

This is the initial task in the evolution of an ontology. In order for ontology evolution to have the desired outcome, it is important that the input ontology is in a consistent state. If the ontology is not in a consistent state, it has to be repaired first, using one of the different ontology diagnosis and repair tools (e.g., RaDON, see Chap. 17) or techniques before starting the evolution process. Note that we require the input ontology to be in a consistent state because dealing with an inconsistent ontology may produce unexpected results. For instance, the propagation of changes may produce inconsistencies in related artifacts. This requirement is also in accordance to existing ontology evolution approaches (e.g., Stojanovic (2004)). Besides, the main goal of ontology evolution is to adapt an ontology to arisen needs (e.g., changes in the domain, changes in the experts knowledge, etc.), not to repair an inconsistent ontology. Therefore, the input of the evolution process is an ontology that correctly models a particular domain/task, before new needs arise. However, the repairing of an inconsistent ontology before starting the ontology evolution process can be seen as a preprocessing task. The first step of this task is basically initiating the change process. Changes can either be requested from users or developers, who feel that the ontology is not adequate in its current form, or changes can be discovered. In literature (Stojanovic 2004), change discovery is distinguished into top-down and bottom-up change discovery. Top-down (deductive/explicit) changes are often the results from knowledge elicitation techniques that are used to acquire knowledge directly from human experts (e.g., domain experts or end users). Bottom-up changes are typically the result from machine learning techniques, which use different methods to infer patterns from the sets of examples (e.g., structure/data/usage-driven change discovery).

Once changes are discovered or requested, they have to be represented in a formal and explicit way. Typically, a change ontology is used to model proposed/requested changes (e.g., Stojanovic 2004; Klein and Noy 2003; Noy et al. 2006; Palma et al. 2009). This formal representation of ontology changes makes them machine-understandable, which supports and facilitates many evolution activities: their propagation to ontology related entities, the synchronization of distributed copies of the same ontology, their integration with information related to the process of the ontology development (e.g., accept/reject changes), the identification of conflicts, etc. Moreover, having changes formally represented makes them usable by other ontology evolution systems as well as exploitable for supplementary functionality of an ontology evolution system such as learnability. Finally, it allows to keep track of the ontology changes by generating a log that maintains the history (and order) of applied changes as a sequence of individuals of the proposed model.

In contrast to previous approaches in the literature, in NeOn, a layered approach for the representation of ontology changes was proposed (Palma et al. 2007, 2009), which consists of a generic ontology, independent of the underlying ontology model that models generic operations in a taxonomy of changes that are expected to be supported by any ontology language. Furthermore, the model can be specialized for different ontology languages, allowing the reuse and refinement for specific needs. Also, the model extends previous taxonomies of changes with a more granular classification that considers the actual atomic changes that can be performed in an ontology.

In case there are multiple change requests for an ontology, the requested changes have to be prioritized. In order to determine which change should be implemented first, one can rely on the status of the person requesting the change, or have an ontology engineer review the requested changes and rank them according to urgency. It is also important that dependencies are considered when ranking the requested changes. It could be that changes are dependent on each other or even contradict each other.

Finally, this task may include the use of a well-defined process (a workflow) for coordinating change proposals (see Palma et al. 2008a, b). This process is responsible for determining who (depending on the user permissions) can do what (what kind of actions) and when (depending on the state of the ontology element (e.g., classes, properties and individuals), and the permissions of the user).

Tool Support in the NeOn Toolkit

- RaDON plugin is an ontology diagnosis and repair tool that can be used before starting the evolution activity, i.e., before applying changes.
- Tools supporting the request/discovery of changes:
  - The workflow feature supports the process that coordinates the proposal of changes in a collaborative environment. It supports a top-down/explicit discovery method, i.e., when changes are requested by users/developers.
  - The Evolva plugin supports the discovery of changes from external data sources (e.g., text, folksonomies, or RSS feeds). Changes are integrated and evaluated by relying on background knowledge such as online ontologies. In the next section, we present in details the guidelines for how to exploit such tool to apply the identified changes on the ontology and produce a new ontology version.

Task 2 Planning the Change

In this task, the change request is analyzed, and it is determined why the change needs to be made and which part of the ontology is affected by the change.

For that purpose, one uses impact analysis, where all potential consequences (side effects) of a change are identified along with an estimation of what needs to be modified to accomplish a change (Arnold 1996; Bohner 1996). As we noted in the introduction, ontologies may depend on several others and may also be related to other artifacts (e.g., individuals, mappings, applications, metadata, etc.). Hence, for the analysis of the impact of a change, a complete list of all implications to the ontology and its dependent artifacts should be presented to the ontology engineer (Plessers 2006).

The previous analysis is also helpful to estimate the cost of evolution. Based on this cost, the ontology engineer can decide whether or not to propagate a change to a dependent artifact (Plessers 2006).

As a result of the analysis performed during this task, the ontology engineer may decide to implement the change, or if the change has many side effects or if the cost of implementation is too high, he may defer the change request to a later time or not implement it at all.

Once the ontology developer team has decided which changes will be implemented and how they have to be implemented, the next phase of ontology evolution, namely change implementation, is entered.

Tool Support in the NeOn Toolkit

- The NeOn Toolkit provides simple support when deciding whether to make a change or not. In particular, when a user wants to delete an ontology element, the list of related axioms (the side effect) is shown to the editor, which permits him to verify the cost of implementing the change.

Task 3 Implementing the Change

Implementing the changes is of varying difficulty, depending on the impact of the requested change. While some change can be as easy as adding or removing a subclass, other changes can require complex operations and restructuring of the ontology.

One of the first and foremost important features is change logging, which allows to track which changes have been made, and also allows for an easy undo, in case something goes wrong. The change log can also be published to inform people using the ontology on the updates.

If the requested change turns out to be too difficult to be implemented, the ontology may need to be restructured first, before the actual desired change can be implemented (Chikofsky and Cross 1990). Depending on the complexity of the task, an ontology engineer can be chosen to perform the restructuring and the subsequent implementation of the changes. For instance, in (Proper and Halpin 2004), the authors distinguish three reasons to apply transformation: (1) to select an alternative conceptual schema which is regarded as a better representation of the domain, (2) to enrich the schema with derivable parts creating diverse alternative views on the same conceptual schema as a part of the original schema, and (3) to optimize a finished conceptual schema before mapping it to a logical design.

One important issue to take into consideration when implementing a change is the management of inconsistencies that this change may introduce in the ontology. In case an inconsistency occurs, it has to be decided how to address it. While some approaches try to keep the ontology in consistent state at all cost by even disallowing changes introducing inconsistencies, others claim that the inconsistencies are inevitable and hence we have to deal with them. Regardless of the approach, the inconsistencies have to be identified and resolved, possibly using some tools as it was mentioned in the introduction. In the literature, this activity has been introduced in (Stojanovic 2004) as the semantics of the change (originally proposed in the area of data schema evolution in Banerjee et al. 1987) and includes the computation of additional changes that guarantee the transition of the ontology into another consistent state. It enables the resolution of induced changes in a systematic manner, ensuring the consistency of the whole ontology. In particular, the author focuses on the structural inconsistencies that arise when the ontology model constraints are invalidated after a change request. Additionally, the author introduces evolution strategies to choose how a change should be resolved based on the structure of the ontology, the complexity of the process, the frequency of the strategy use, or on an explicitly given state of the instances to be achieved (given by the ontology engineer).

Furthermore, another important issue that has to be addressed during the implementation of the change(s) is the management of the ontology version. After the ontology changes, the ontology engineer should decide whether the resulting ontology constitutes a new version of the ontology and hence it should have a different version information. Some recommendations on the use of URIs can be found. For instance, in (Klein and Fensel 2001), the authors propose to use an URI for ontology identification with a two-level numbering scheme: major and minor. Minor numbers for backward compatible modifications (an ontology-URI ending with a minor number identifies a specific ontology). Major numbers for incompatible changes (an ontology-URI ending with a major number identifies a line of backward compatible ontologies). In practice, however, it is common that ontologies do not include any version information at all. As a consequence, usually it is not easy to identify different versions of an ontology. The problem of identifying ontologies in the Semantic Web is not a trivial issue (see Klein and Fensel 2001). For instance, in (Palma et al. 2008c), a composite identification consisting of the URI plus version (if available) plus the location of the ontology is used to identify an ontology.

Finally, as aforementioned, the change(s) have to be propagated to all the ontology related artifacts (if the ontology engineer decided to do it in the previous task based on the analysis of the cost and impact). In (Stojanovic 2004), the author discusses the propagation of changes to dependent ontologies, individuals, and applications and elaborates on the propagation to dependent ontologies using a combination of push and pull mechanism. For the propagation to ontology individuals, several mechanisms can be applied from the research in the area of databases. For instance, in (Parsia et al. 2005), the authors discuss how changes can be propagated to the individuals of the database by using four possible mechanisms: immediate conversion (propagate changes as they happen), deferred conversion (propagate changes at specific points in time), explicit deletion (when referenced concepts are dropped), or filtering (for using different versions of the schema). In NeOn, the propagation of changes has also been considered to (1) distributed copies of the same ontology and (2) ontology metadata (Palma 2009; Palma et al. 2007, 2008b).

Tool Support in the NeOn Toolkit

- NeOn Toolkit ontology editor allows the manual application of changes to ontologies.
- The change capturing plugin supports the logging of changes automatically from the NeOn ontology editor. It also supports the application of logs generated by other systems. Additionally, it is also in charge of propagating changes to the distributed copies of the same ontology.
- RaDON plugin can be used for the management of inconsistencies.

Task 4 Verification and Validation

Before the ontology is considered evolved completely, the last step deals with assessing questions whether the right ontology is built and whether it is built in the right way. During this assessment, usually not only the ontology originally modified is verified in isolation, but in general, this activity can include the verification of other artifacts related to the ontology (as mentioned above) to ensure that they were not changed in a wrong way or they have an unexpected behavior. The verification and validation step can include the following activities:

- Formal verification, such as state machines and temporal logics, to derive useful properties of the system under study
- Testing by users or automatically to verify whether the system behaves as expected
- Debugging for localizing and repairing errors found during the verification or testing (usually performed by an ontology engineer) (for example Haase et al. (2006))
- Quality assurance, which typically concerns non-functional qualities, like reusability, adaptability, interoperability, etc.
- Justification of the changes, (for example Stojanovic 2004)
- Relevance of the changes with respect to the ontology under evolution (Zablith et al. 2010)

In case problems are detected, these have to be fixed by moving back into Task 3, and then returning to Task 4 to verify the corrected outcome.

Additionally, this task may include curation activities (e.g., approve/reject) derived from the well-defined process (e.g., workflow) that coordinates the change proposals (see Palma et al. 2008a, b). In this case, ontology engineers usually have different roles, and only those with the required authority can accept or reject the change proposals. If a change is rejected, the original author can modify the change and start all over again since Task 1 or he can decide to discard it completely.

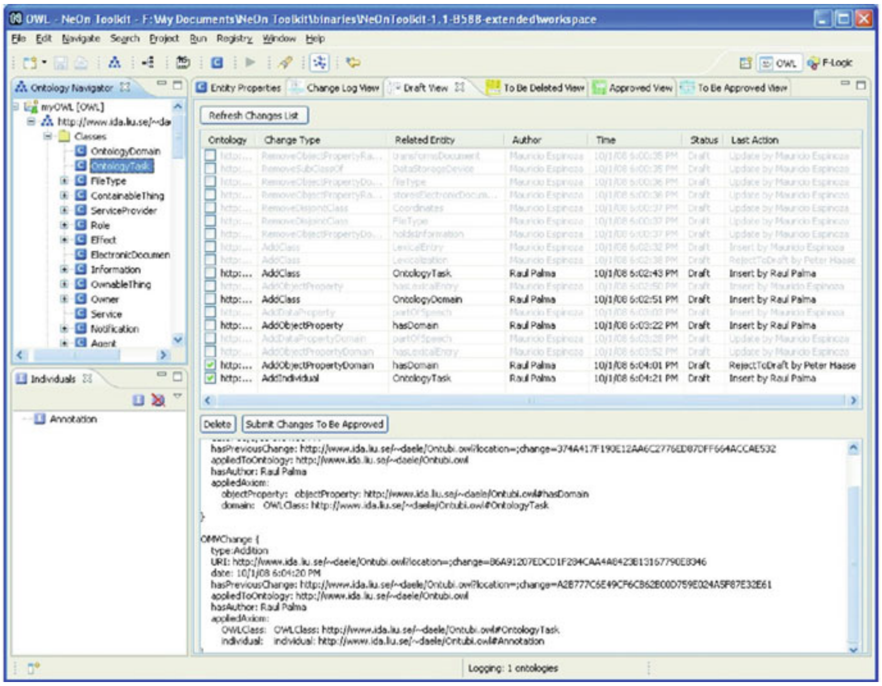
Tool Support in the NeOn Toolkit

- The Cicero plugin supports the justification of changes.
- The workflow feature supports the refining of activities (see Fig. 11.3).
- The Evolva plugin checks the relevance of a change with respect to an ontology by relying on the analysis of ontological contexts and a set of identified relevance patterns supported by a confidence-based ranking (Zablith et al. 2010).

Working with Networked Ontologies

The NeOn project deals with networks of ontologies and networked ontologies (Haase et al. 2006), defined as a collection of ontologies related together via a variety of different relationships such as mapping, modularization, version, and dependency relationships.

Hence, it is worth remarking that the process described above can be applied to networked ontologies since such a process takes into account the existing ontology dependencies with other related artifacts, such as individuals, mappings, applications, and metadata, as we noted in each step. In a nutshell, such dependencies are first considered during the analysis of the impact and cost in Task 2. Furthermore, during the propagation of the changes in Task 3, all the ontology-related artifacts are updated (if necessary), ensuring the consistency of the networked ontologies. Finally, when assessing the correctness of the evolved ontology in Task 4, the verification also takes into consideration the ontology-related artifacts to ensure that



	Ontology	Change Type	Related Entity	Author	Time	Status	Last Action
☐	http:...	RemoveObjectPropertyRa...	transformsDocument	Mauricio Espinoza	10/1/08 6:00:35 PM	Draft	Update by Mauricio Espinoza
☐	http:...	RemoveSubClassOf	DataStorageDevice	Mauricio Espinoza	10/1/08 6:00:35 PM	Draft	Update by Mauricio Espinoza
☐	http:...	RemoveObjectPropertyDo...	fileType	Mauricio Espinoza	10/1/08 6:00:36 PM	Draft	Update by Mauricio Espinoza
☐	http:...	RemoveObjectPropertyRa...	storesElectronicDocum...	Mauricio Espinoza	10/1/08 6:00:36 PM	Draft	Update by Mauricio Espinoza
☐	http:...	RemoveDisjointClass	Coordinates	Mauricio Espinoza	10/1/08 6:00:37 PM	Draft	Update by Mauricio Espinoza
☐	http:...	RemoveDisjointClass	FileType	Mauricio Espinoza	10/1/08 6:00:37 PM	Draft	Update by Mauricio Espinoza
☐	http:...	RemoveObjectPropertyDo...	holdsInformation	Mauricio Espinoza	10/1/08 6:00:37 PM	Draft	Update by Mauricio Espinoza
☐	http:...	AddClass	LexicalEntry	Mauricio Espinoza	10/1/08 6:02:32 PM	Draft	Insert by Mauricio Espinoza
☐	http:...	AddClass	Lexicalization	Mauricio Espinoza	10/1/08 6:02:38 PM	Draft	RejectToDraft by Peter Haase
☐	http:...	AddClass	OntologyTask	Raul Palma	10/1/08 6:02:43 PM	Draft	Insert by Raul Palma
☐	http:...	AddObjectProperty	hasLexicalEntry	Mauricio Espinoza	10/1/08 6:02:50 PM	Draft	Insert by Mauricio Espinoza
☐	http:...	AddClass	OntologyDomain	Raul Palma	10/1/08 6:02:51 PM	Draft	Insert by Raul Palma
☐	http:...	AddDataProperty	partOfSpeech	Mauricio Espinoza	10/1/08 6:03:03 PM	Draft	Insert by Mauricio Espinoza
☐	http:...	AddObjectProperty	hasDomain	Raul Palma	10/1/08 6:03:22 PM	Draft	Insert by Raul Palma
☐	http:...	AddDataPropertyDomain	partOfSpeech	Mauricio Espinoza	10/1/08 6:03:28 PM	Draft	Update by Mauricio Espinoza
☐	http:...	AddObjectPropertyDomain	hasLexicalEntry	Mauricio Espinoza	10/1/08 6:03:52 PM	Draft	Update by Mauricio Espinoza
✓	http:...	AddObjectPropertyDomain	hasDomain	Raul Palma	10/1/08 6:04:01 PM	Draft	RejectToDraft by Peter Haase
✓	http:...	AddIndividual	OntologyTask	Raul Palma	10/1/08 6:04:21 PM	Draft	Insert by Raul Palma

```
   hasPreviousChange: http://www.ida.liu.se/~daele/Ontubi.owl?location=;change=374A417F190E12AA6C2776ED87DFF664ACCAE532
   appliedToOntology: http://www.ida.liu.se/~daele/Ontubi.owl
   hasAuthor: Raul Palma
   appliedAxiom:
      objectProperty:   objectProperty: http://www.ida.liu.se/~daele/Ontubi.owl#hasDomain
      domain:   OWLClass: http://www.ida.liu.se/~daele/Ontubi.owl#OntologyTask
}

OMVChange {
   type:Addition
   URI: http://www.ida.liu.se/~daele/Ontubi.owl?location=;change=B6A91207EDCD1F284CAA4A8423B13167790E8346
   date: 10/1/08 6:04:20 PM
   hasPreviousChange: http://www.ida.liu.se/~daele/Ontubi.owl?location=;change=A2B777C6E49CF6CB62B00D759E024A5F87E32E61
   appliedToOntology: http://www.ida.liu.se/~daele/Ontubi.owl
   hasAuthor: Raul Palma
   appliedAxiom:
      OWLClass:   OWLClass: http://www.ida.liu.se/~daele/Ontubi.owl#OntologyTask
      individual:   individual: http://www.ida.liu.se/~daele/Ontubi.owl#Annotation
```

Fig. 11.3 Collaborative editorial workflow support in NeOn Toolkit

the whole network of ontologies is behaving as expected, i.e., it is consistent. So, any conflict that may arise can be caught at an earlier stage of the ontology evolution process, affecting, for instance, the decision of whether or not a change should be implemented.

11.2.2 Example

To describe the proposed guidelines for the ontology evolution activity in a more practical way, in this section we illustrate how to perform this activity by describing an experiment conducted in collaboration with a team of FAO ontology editors in charge of the maintenance of ontologies in the fishery domain. The editors performed collaboratively a set of typical changes and actions to a stable version of one fishery ontology in order to reach a new stable version. In this scenario, a central server kept a shared copy of the ontology and the related changes. In the remainder of this section, we describe only the most relevant points. A detailed and complete description of the experiment is presented in Palma (2009).

Task 1 Requesting a Change

Initially, FAO experts in the fishery domain requested a set of changes to be applied to the current version of the species ontology[1] (v1.0 at the time the experiment was conducted) – the ontology models a taxonomic classification of biological entities, including classes such as Family, Group, Order, and Species. In this case, changes were discovered using a top-down/explicit method as the knowledge came directly from human experts. A total of 34 changes were requested using real information according to the experts (see Palma 2009). Examples of those changes are: to add Individual 31005–10001 (Species); to add Individual 31005–10001 DataProperty hasNameScientific, value: Pterodroma wrong macroptera, type: string; to add Root Class Speciation; and to add ObjectProperty hasScientificNameAuthor.

In this scenario, different ontology editors, with different roles (Subject Experts, and Validator), were working collaboratively in the implementation of the changes and hence it was not necessary to prioritize them (prioritization of multiple changes).

Each of the proposed changes was represented as an individual of the change ontology proposed in Palma et al. (2009) – representation of changes. For this experiment, ontology editors were using the NeOn Toolkit with the collaborative infrastructure. Hence, the representation of the changes was performed automatically whenever a new change was captured by the change capturing plugin of NeOn Toolkit.

Furthermore, in this scenario, the ontology editors were following a well-defined process (workflow) for the coordination of the change proposals. As a consequence, during this task the system created for any new change proposal, the appropriate workflow action automatically (insert, update, delete).

Task 2 Planning the Change

For this experiment, it was necessary to implement the requested changes regardless of the side effects. Therefore, it did not perform any analysis of the impact or cost. In fact, the idea of the experiment was to assess the efficiency of the system to support the development of an ontology in a collaborative scenario, not the time or cost of implementing a change.

Task 3 Implementing the Change

For this task, no restructuring of the change(s) was necessary, because on the one hand the changes were not too difficult to implement due to the ontology structure, and on the other hand, the cost of implementing was not an issue.

Additionally, for this task, the system (change capturing plugin of NeOn Toolkit) took care of logging automatically all of the proposed changes (change logging), maintaining the chronological history of the events.

[1] Available at http://aims.fao.org/en/website/Fisheries-ontologies-/sub2#species

In this experiment, the change(s) did not introduce any inconsistencies in the ontology. However, in case it would be necessary to manage inconsistencies, the RaDON plugin for NeOn Toolkit could have been used to detect and fix them.

As we introduced at the beginning of this section, for this experiment, the ontology and related changes were centralized in a server. Furthermore, the ontology used for the experiment was not related to other artifacts at the moment. Hence, it was not necessary any propagation of changes.

Task 4 Verification and Validation

During this task, the ontology editors analyzed every change to ensure that the resulting ontology was as expected using the visualization plugins of the NeOn Toolkit.

Additionally, this task was one of the most important of the experiment as it included all the curation activities derived from the workflow that coordinates the proposal of changes. Hence, in this task, an ontology validator was in charge of accepting and rejecting changes as necessary by using the appropriate workflow plugins of the NeOn Toolkit. Finally, at the moment of the experiment, there was no support for the justification of changes.

11.3 Guidelines for Exploiting Tools in Ontology Evolution

In this section, we propose a methodological guideline for supporting users in identifying new and relevant domain changes from external data sources. Such guidelines aim to facilitate the process on evolving ontologies to reflect the latest changes in certain domains by analyzing various data sources. This guideline complements the tool-based support provided by the Evolva ontology evolution framework (discussed next), with concrete guidance on how to realize the various tasks of the evolution activity, using semiautomatic techniques in an efficient way.

11.3.1 The Evolva Framework

The Evolva ontology evolution framework (Zablith 2009) relies on the hypothesis that various forms of data corpus (texts, folksonomies, RSS feeds, etc.) can be used to detect the need for an evolution and initiate it (see Fig. 11.4). Evolva also relies on the idea that, in order to integrate new pieces of information extracted from the exploited sources into the current ontology, evolution systems can rely on the automated use of external background knowledge sources, which can be supplied by online ontologies, lexical resources (e.g., WordNet, Fellbaum 1998), or the web. An additional use of background knowledge comes at the level of online ontologies used to assess the relevance of statements with respect to the ontology in focus.

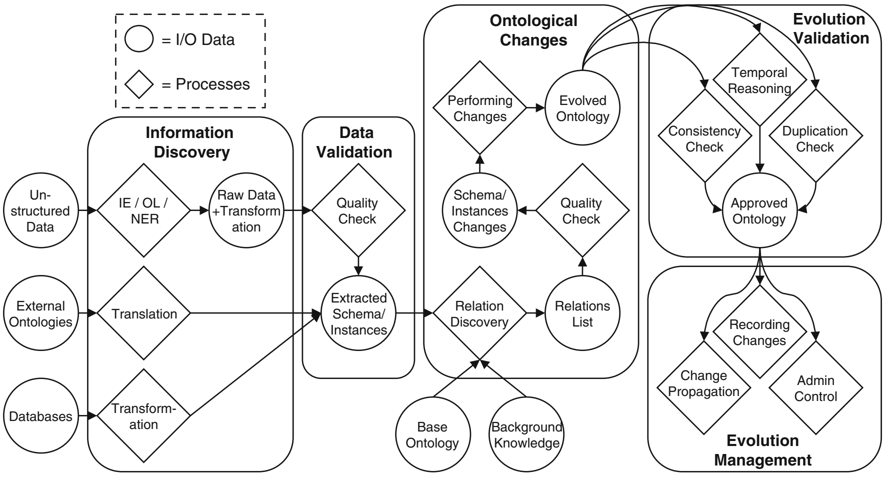


Fig. 11.4 Evolva's ontology evolution framework

While the goal of the Evolva framework is to reduce, as much as possible, human intervention within the evolution process, user input is required at the level of evolution management and for fine-tuning of various parts of the framework. The role of the user is needed to properly parameterize the components, select the right sources of information and of background knowledge, validate the results of various steps, and, generally, guide the evolution process to obtain high-quality results. These tasks are not trivial, as they depend a lot on the particular ontology to be evolved, the domain covered, the applications relying on the ontology, and the reasons for its evolution. The experience of the knowledge engineer and his/her knowledge of the ontology and of the exploitable sources of information are therefore essential.

11.3.2 Tasks

The tasks for performing a semiautomatic ontology evolution can be seen in the workflow shown in Fig. 11.5. In this context, the starting point is an existing ontology (depicted as V1 in Fig. 11.5 and *base ontology* in Fig. 11.4, see Sect. 11.3.1), which the user aims to evolve based on available domain data sources. The selection of the appropriate sources from which new ontology entities are identified depends on the evolution use case and the availability of such sources in the domain in focus. In the rest of this section, we present the details of the tasks involved in semiautomatically evolving the ontology.

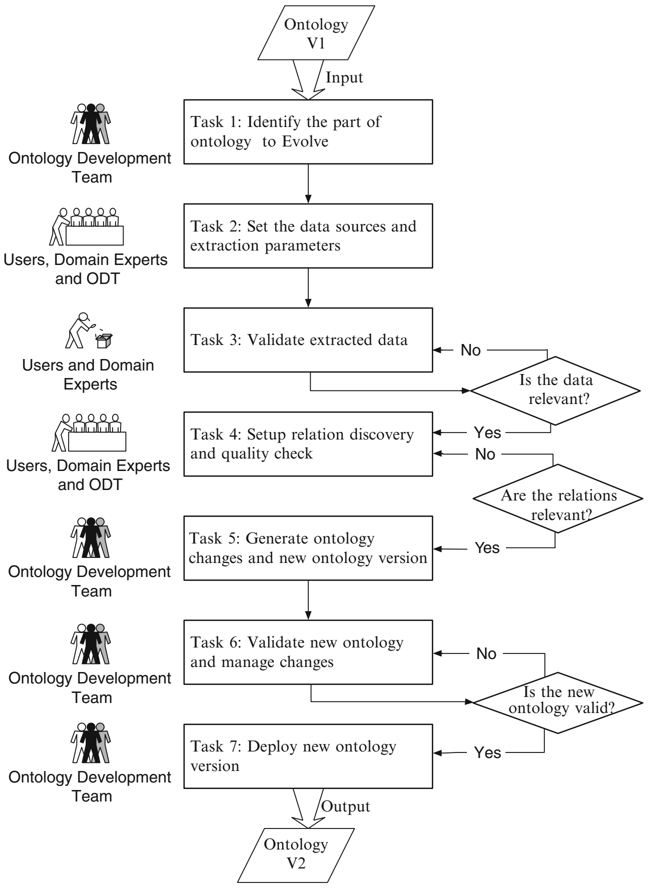

Ontology V1
Input
Ontology Development Team
Task 1: Identify the part of ontology to Evolve
Users, Domain Experts and ODT
Task 2: Set the data sources and extraction parameters
Users and Domain Experts
Task 3: Validate extracted data
No
Is the data relevant?
Yes
Users, Domain Experts and ODT
Task 4: Setup relation discovery and quality check
No
Are the relations relevant?
Yes
Ontology Development Team
Task 5: Generate ontology changes and new ontology version
Ontology Development Team
Task 6: Validate new ontology and manage changes
No
Is the new ontology valid?
Yes
Ontology Development Team
Task 7: Deploy new ontology version
Output
Ontology V2

Fig. 11.5 Tasks for ontology evolution supported by semiautomatic tools

Task 1 Identify the Part of Ontology to Evolve

The first task required by the ontology development team is to select the part of the ontology to evolve. The evolution can be applied either on the entire ontology or on a certain part of it. In many cases, ontologies may include a significant amount of statements, causing the evolution to take a long processing time. In such cases, after specifying the evolution purpose, the user may choose the part of the ontology to evolve through selecting the set of concepts to be handled by the process.

Task 2 Set the Data Sources and Extraction Parameters

Depending on the domain, domain experts should prepare the data sources that contain relevant information to the ontology context. Such data sources could be in the form of text documents, folksonomies, databases, or even other ontologies. Based on the decision of the ontology development team to evolve the ontology either in terms of schema, individuals, or both, the extraction should be customized accordingly. For example, in the case of schema evolution, the user may choose to extract concepts for the data sources, without dealing with individuals. While in the case of individuals, the evolution process could omit the extraction of schema elements. Choosing between schema and individuals evolution could be biased by the ontology functionalities and domain nature, i.e., when many ontology-dependent components exist (e.g., various applications or other aligned ontologies), evolving the ontology schema may be costly, and the ontology development team may choose to perform this operation less frequently. While in environments where ontology components are easily controllable and where a lot of new information is generated leading to a frequent generation of new concepts, schema evolution would be required.

Task 3 Validate Extracted Data

After extracting knowledge elements from the data sources, noise and irrelevant entities should be excluded. The user is supported by manual and automated validation techniques with customizable parameters. For the manual validation, the domain expert would serve as one of the best quality checkers as he/she is the most knowledgeable about the ontology context. This task is completed after checking that all the data are valid to be processed further by the system.

Task 4 Setup Relation Discovery and Quality Check

The role of the user, after the data validation task, is to prepare the automated relation discovery process. The relation discovery process links the validated data to the ontology. This requires the user to select the various types of background knowledge sources to be used. The choice of background knowledge is directly dependent on the type of domain the ontology represents. If the domain were specialized, the user would choose domain-specific background knowledge sources (e.g., specialized thesauri). This would improve the quality of relations and increase the system precision. While if the domain is generic, using online ontologies or generic thesauri would perform well. In addition to the selection of sources, the user should fine-tune the parameters of the relation discovery process, such as the settings related to the automatic relevance checking, or specify the maximum depth to explore. In addition to the supplied automatic quality checking methods, for example, in terms or relevance, domain experts should additionally check the quality of relations, before using them later in the system.

Task 5 Generate Ontology Changes and New Ontology Version

Based on the approved relations in the previous task, ontology changes are generated and applied on the new ontology version. Users should specify where to apply the changes, i.e., directly on the initial ontology or on a detached copy. The choice of where to perform changes depends on the environment and the ontology development team approach. The team should be aware that applying changes on the initial ontology would directly affect the dependent components. If this is not feasible, or designers prefer to keep the initial ontology intact while reviewing the changes, creating a detached ontology version would be more appropriate.

Task 6 Validate New Ontology and Manage Changes

The user should control the changes performed on the new ontology version. With the new evolved ontology, problems such as inconsistencies and duplication are likely to emerge. Users in this task specify the checking methods to be applied on the new ontology version using reasoners, for example, in addition to manually control the recorded ontology changes.

Task 7 Deploy New Ontology Version

Once the new version is approved, users should control the propagation of the new ontology version to the dependent components. Links to the previous ontology version should be checked and whether the new ontology has been successfully saved and accessible.

11.3.3 Example

In this part, we highlight an example of ontology evolution scenario using the Evolva plugin for the NeOn Toolkit, following the guidelines presented in the previous section. We run our example in an environment where the NeOn Toolkit and Evolva plugin[2] are operational.

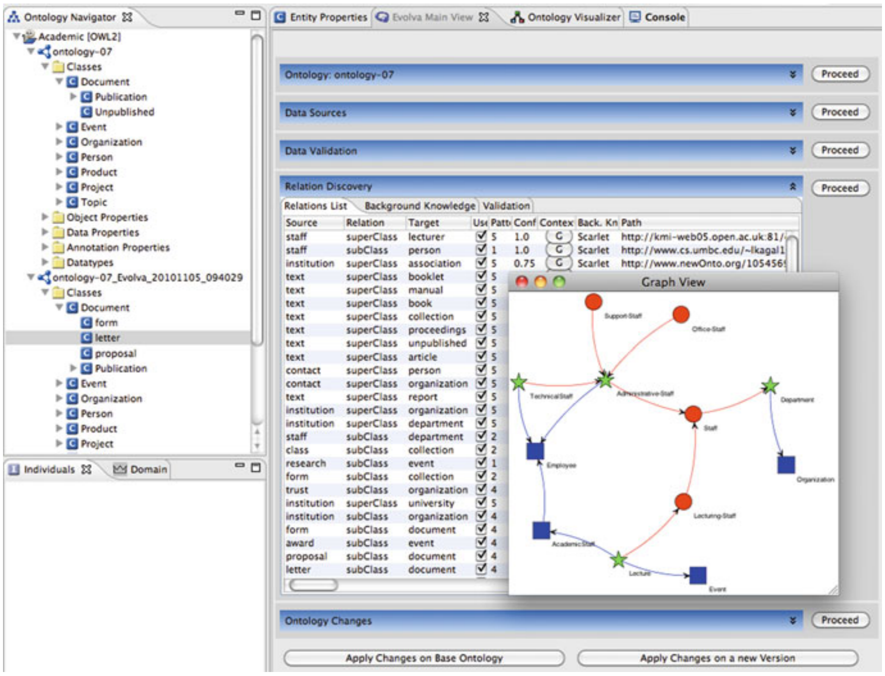
Consider the case of evolving the latest version of the SWRC ontology[3] in the academic context. We first load the ontology in the NeOn Toolkit and start Evolva. In our simple case where the ontology has a limited number of concepts and time is not an issue, we choose to evolve all the ontology (Task 1). This choice can be specified in the first step of the process (called *Ontology*) in Fig. 11.6.

After preparing the ontology and identifying the part of ontology to evolve, we move to select the data sources containing relevant information with a potentially added value to our ontology (Task 2). This is implemented in *Data Sources* step of Fig. 11.6. A relevant source of information we found was on the Leverhulme website[4] that contains text documents about research project and information about people in the academic domain. We locate and download the relevant text documents, then select the sources in Evolva for performing data extraction and validation. Having no ontology-dependent components, a schema evolution would not have any side effects on applications or other dependent elements. Thus, as ontology developers in this use case, we test the extraction of concepts from the data sources and integrate them in the ontology. Evolva includes extraction of

[2] Details on how to install and run Evolva can be found at: http://evolva.kmi.open.ac.uk/

[3] The SWRC ontology can be downloaded from: http://ontoware.org/swrc/swrc/SWRCOWL/swrc_updated_v0.7.1.owl

[4] http://www.leverhulme.ac.uk/



Source	Relation	Target	Use	Patt	Conf	Contex	Back. Kn	Path
staff	superClass	lecturer	✓	5	1.0	G	Scarlet	http://kmi-web05.open.ac.uk:81/
staff	subClass	person	✓	1	1.0	G	Scarlet	http://www.cs.umbc.edu/~lkagal1
institution	superClass	association	✓	5	0.75	G	Scarlet	http://www.newOnto.org/105456
text	superClass	booklet	✓	5				
text	superClass	manual	✓	5				
text	superClass	book	✓	5				
text	superClass	collection	✓	5				
text	superClass	proceedings	✓	5				
text	superClass	unpublished	✓	5				
text	superClass	article	✓	5				
contact	superClass	person	✓	5				
contact	superClass	organization	✓	5				
text	superClass	report	✓	5				
institution	superClass	organization	✓	5				
institution	superClass	department	✓	5				
staff	subClass	department	✓	2				
class	subClass	collection	✓	2				
research	subClass	event	✓	1				
form	subClass	collection	✓	2				
trust	subClass	organization	✓	4				
institution	superClass	university	✓	5				
institution	subClass	organization	✓	4				
form	subClass	document	✓	4				
award	subClass	event	✓	4				
proposal	subClass	document	✓	4				
letter	subClass	document	✓	4				

Fig. 11.6 Screenshot of the Evolva plugin

concepts from text documents and RSS feeds, as well as a list of raw terms. The validation parameters incorporate term existence checking feature (based on a similarity value) and a term length checker for removing terms under a specified length.

We load the Leverhulme text documents and run the extraction and validation process. A list of extracted concepts is returned, with Evolva automatically identifying existing terms in the ontology and terms that fall below a length threshold. If the automatic validation performs poorly overall, it is possible for users to fine-tune the parameters and rerun the validation process again. In addition to the automatic validation, users have the ability to go through the list of concepts and manually select terms they find irrelevant (Task 3), implemented in the *Data Validation* step in Fig. 11.6. Domain experts would play here a major role as they are the most aware of the relevance of concepts with respect to the ontology.

After the data validation process and approving relevant data, we move to Task 5 of setting up the relation discovery process with the right background knowledge, sources, and parameters. The SWRC ontology domain is, to some extent, a generic academic purpose ontology. Thus, related information can be easily found through online ontologies in which a lot of academic domain ontologies can be found, as well as through WordNet, the generic thesaurus. Thus, we choose to perform the relation discovery process through exploiting online ontologies using Scarlet (Sabou et al. 2008) (a Semantic Web–based relation discovery engine) and WordNet.

Evolva automatically harvests the chosen background knowledge sources and identifies how extracted concepts should be integrated in the ontology. If needed, Evolva also provides the option to discover relations between new concepts, before being integrated in the ontology. This has been implemented in the *Relation Discovery* step in Fig. 11.6. To illustrate how background knowledge sources integrate new concepts, *Applicant* and *Website* are two concepts extracted from the Leverhulme text document. WordNet links *Applicant* as a *subclass* of *Person*, an existing concept in the SWRC ontology, while online ontologies link *Website* to *Organization* through a *hasWebsite* relation. The length of relations to discover is customizable. Thus, if the users find that the process is taking long, or lengthy relations prove to be overall irrelevant, they can decrease the relation length threshold and rerun the process again.

After the relations linking new concepts to existing concepts in the ontology, Evolva relies on online ontologies from where the relation is identified to assess the relevance of the relation with respect to the ontology. Using identified relevance patterns, with pattern-specific confidence, relations are returned ranked to the user with the highly relevant relations placed on top (Zablith et al. 2010). The user is supplied with a customizable graphical visualization of the ontological contexts (shown on top of Fig. 11.6), as a validation of the relevance calculation. In addition to the visualization parameters, it is possible to change the weight of relevance patterns, hence affecting the overall ranking of relations.

Once all relations are approved and relevant, they are used to generate the ontology changes (last step in Fig. 11.6). If the user spots any unwanted changes, it is possible to go back to the relation validation, remove the source relations, and regenerate the ontology changes. Based on the ontology changes provided, it is possible to apply the changes on the source ontology, or a new version with the evolution date appended to the name of the new ontology version. Changes are applied automatically within the NeOn Toolkit, and the user will instantly see the updates in the ontology navigator of the toolkit (on the left of Fig. 11.6).

Our next task is to validate the new ontology version and manage the new changes that the ontology has been subject to (Task 6). Evolva relies on the change logging plugin (Palma et al. 2008b) based on the NeOn Toolkit. The user is given all the functionalities to review changes after being applied on the ontology. Inappropriate changes can be rolled back, or sent for further review, until reaching a reliable new ontology version.

After approving the final ontology version, we deploy it by double-checking the links to the previous ontology version that are automatically created by Evolva (Task 7). We also check that the ontology has been saved correctly, and that it is still accessible by doing some checks such as running queries and validating the results.

11.4 Conclusion

Ontology evolution is a tedious and time-consuming task. To successfully keep the ontology up to date with domain changes, ontology engineers should be supplied with the right guidelines and tool usage to make this task easier. For that, we presented in this chapter guidelines for ontology evolution covering two aspects: a high-level ontology evolution process and tool-oriented guidelines to semiautomatically identify, evaluate, and apply domain changes to ontologies.

The first aspect describes the tasks involved in the ontology evolution process from a generic perspective and discusses guidelines in possible ways to achieve each task. The second aspect aims to facilitate the process of identifying ontology changes from external domain data, checking their quality, and integrating them in the ontology, by using semiautomatic techniques. The guidelines in this case include how to use and parameterize the involved tools to achieve the optimal new ontology version.

The two aspects work together to enable ontology engineers to understand the complete picture and tasks involved in ontology evolution, to successfully move from an existing ontology state to a new one with the appropriate representation of domain changes that arise.

References

Arnold RS (1996) Software change impact analysis. IEEE Computer Society Press, Los Alamitos

Banerjee J, Kim W, Kim HJ, Korth HF (1987) Semantics and implementation of schema evolution in object-oriented databases. SIGMOD Rec 16(3):311–322

Bennett KH, Rajlich V (2000) Software maintenance and evolution: a roadmap. In: ICSE - future of SE track, ACM, New York, pp. 73–87

Berners-Lee T, Fielding R, Masinter L (2005) RFC 3986, Uniform Resource Identifier (URI): Generic syntax. Available at http://tools.ietf.org/html/rfc3986

Bohner SA (1996) Software change impact analysis for design evolution. In: Software change impact analysis. IEEE Computer Society Press, Los Alamitos, pp 67–81

Chikofsky EJ, Cross JG (1990) Reverse engineering and design recovery: a taxonomy. IEEE Softw 7(1):13–17

Fellbaum C (1998) Wordnet: an electronic lexical database. MIT Press, Cambridge

Haase P, Rudolph S, Wang Y, Brockmans S, Palma R, Euzenat J, d'Aquin M (2006) NeOn deliverable D1.1.1. Networked ontology model. Available at http://www.neon-project.org/

Klein M, Fensel D (2001) Ontology versioning for the semantic web. In: Proceedings of the international semantic web working symposium (SWWS'01), Stanford University, Stanford, CA, USA

Klein M, Noy N (2003) A component-based framework for ontology evolution. In: Proceedings of the IJCAI'03 workshop: ontologies and distributed systems, Acapulco, Mexico

Leenheer PD, Mens T (2007) Ontology management. Semantic web, semantic web services, and business applications. In: Ontology evolution. State-of the-art and future directions. Springer, New York/London

Noy N, Chugh A, Liu W, Musen M (2006) A framework for ontology evolution in collaborative environments. In: International semantic web conference, Athens, pp 544–558

Palma R (2009) Ontology metadata management in distributed environments. PhD thesis, Departamento de Inteligencia Artificial, Facultad de Informatica, Universidad Politecnica de Madrid

Palma R, Haase P, Wang Y, d'Aquin M (2007) D1.3.1 propagation models and strategies. Technical report D1.3.1, UPM; NeOn deliverable. Available at http://www.neon-project.org/

Palma R, Haase P, Corcho O, Gómez-Pérez A (2008a) An editorial workflow approach for collaborative ontology development. In: ASWC'08. Springer, Berlin

Palma R, Haase P, Jiu Q (2008b) D1.3.2 Evaluation of propagation models and strategies. Technical report D1.3.2; NeOn deliverable

Palma R, Hartmann J, Haase P (2008c) OMV – ontology metadata vocabulary for the semantic web. v. 2.4. Available at http://omv.ontoware.org/

Palma R, Haase P, Corcho O, Gómez-Pérez A (2009) Change representation for OWL 2 ontologies. In: Proceedings of the fifth international workshop OWL: experiences and directions. In ISWC09, Chantilly, VA, USA

Parsia B, Sirin E, Kalyanpur A (2005) Debugging OWL ontologies. In: Proceedings of the 14th international conference on world wide web. ACM Press, New York, pp 633–640

Plessers P (2006) An Approach to Web-based Ontology Evolution. PhD thesis, Department of Computer Science, Vrije Universiteit Brussel, Brussel

Proper HA, Halpin TA (2004) Conceptual schema optimisation – database optimization before sliding down the waterfall. Technical report, Department of Computer Science, University of Queensland

Sabou M, d'Aquin M, Motta E (2008) Exploring the semantic web as background knowledge for ontology matching. J Data Semant XI:156–190

Stojanovic L (2004) Methods and tools for ontology evolution. PhD thesis, University of Karlsruhe (TH)

Studer R, Benjamins VR, Fensel D (1998) Knowledge engineering: principles and methods. Data Knowl Eng 25(1–2):161–197

Suárez-Figueroa MC, Gómez-Pérez A (2008) First attempt towards a standard glossary of ontology engineering terminology. In: Proceedings of 8th international conference on terminology and knowledge engineering (TKE'08) Copenhagen, DENMARK, pp 1–15

Zablith F (2009) Evolva: a comprehensive approach to ontology evolution. In: Proceedings of ESWC 2009: the semantic web: research and applications – PhD symposium, Heraklion, pp 944–948

Zablith F, Sabou M, d'Aquin M, Motta E (2010) Using ontological contexts to assess the relevance of statements in ontology evolution. In: Proceedings of the 17th conference on knowledge engineering and knowledge management by the masses (EKAW), Lisbon, Portugal. Springer, Berlin