KAON – An Overview

Karlsruhe Ontology Management Infrastructure

Thomas Gabel, York Sure and Johanna Voelker
(Institute AIFB, University of Karlsruhe)

with Acknowledgements to our colleagues at
FZI – Research Center for Information Technologies, Karlsruhe,
and at Institute AIFB, University of Karlsruhe,
for their support.

April 7th, 2004

Summary

This document is an informal deliverable provided to SEKT partners. The main aim of this document is to get partners quickly started with using the KAON Open Source ontology management infrastructure. KAON consists of a number of different modules providing a broad bandwidth of functionalities centered around creation, storage, retrieval, maintenance and application of ontologies. It was and currently is being further developed in a joint effort mainly by members of the Institute AIFB at University of Karlsruhe and the FZI – Research Center for Information Technologies, Karlsruhe.

We will introduce the following components of and related to KAON:

- **OI-Modeler** (ontology editor)
- **KAON API** (programming interface for developers)
- **KAON Engineering Server** (server for distributed ontology engineering, to be used in combination with OI-Modeler as front-end)
- **TextToOnto** (workbench for ontology learning from texts, feeds learned ontologies into the OI-Modeler)

We will conclude with an outlook of future development trends. Finally, an overview on **download and installation** information is presented in Appendix A. For interested readers we provide numerous references to further existing publications.

Contents

Chapter 1

Introduction

1.1 Motivation

This document provides an informal overview of the components and tools related to KAON, the Karlsruhe ontology management infrastructure. The main aim of this document is to get interested users and developers quickly started with using the Open Source ontology management infrastructure KAON.

KAON consists of a number of different modules providing a broad bandwidth of functionalities centered around **creation, storage, retrieval, maintenance and application of ontologies**. It was and currently is being further developed in a joint effort mainly by members of the Institute AIFB at University of Karlsruhe and the FZI – Research Center for Information Technologies, Karlsruhe.

Before presenting an outline of this document we will clarify in the next section the overall picture on what kind of KAON components exist currently. If you are not yet confused by the plethora of tools or if you have only an interest in a special tool you can leave out the next section and continue with section 1.3.

Note: Please be aware that we here present a snapshot of currently available versions. Future versions of the tools might have additional and/or different functionalities etc. In appendix A we will provide detailed information about download and installation including a table describing the version numbers of the here described tools.

1.2 KAON Overview

The *KArlsruhe ONtology and Semantic Web tool suite* a.k.a. **KAON ToolSuite** is, as mentioned before, an Open Source ontology management infrastructure. However, there exists also external components which support functionalities such as e.g. ontology learning from texts. An overview of the KAON ToolSuite and its main components - *KAON*, *KAON Extensions* and *TextToOnto* - is presented by figure 1.1.

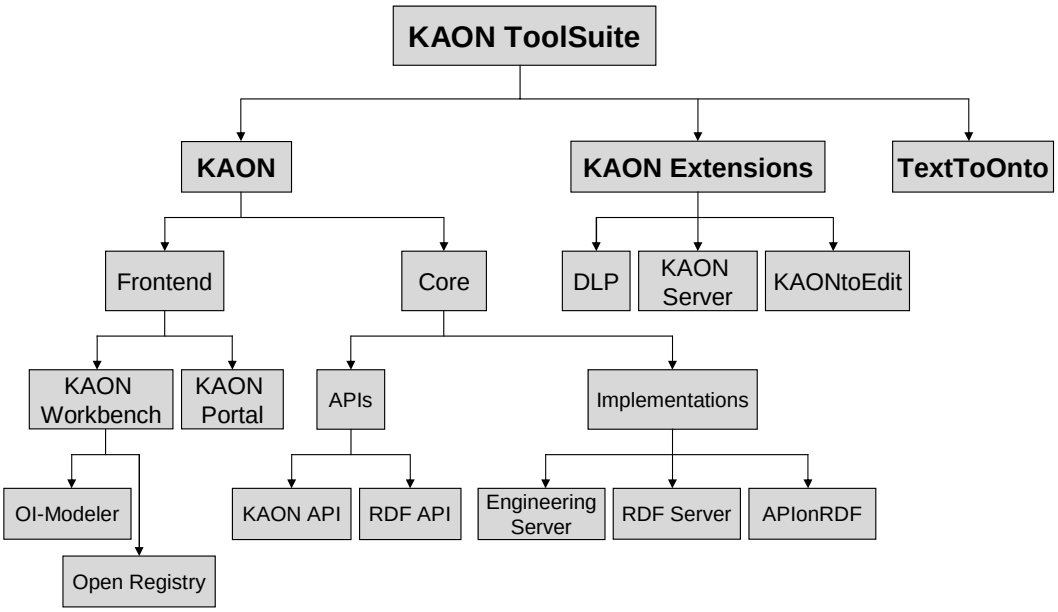

Figure 1.1: KAON Tool Overview

- **KAON** consisting of *KAON Frontend* and *KAON Core* includes a variety of different modules for ontology creation and management.

  The **Frontend** is represented by two applications developed in order to be used particularly by human users:

  - **KAON Workbench** provides a graphical environment for ontology-based applications. It includes the **OI-Modeler** (cf. chapter 2) - a graphical ontology editor - and the **Open Registry** (a.k.a. *Ontology Registry*), which provides mechanisms for registering and searching ontologies in a distributed context.

  - **KAON Portal** is a simple tool for multi-lingual, ontology-based Web portals.

  The **Core** of KAON supports programmatic access to ontologies by including both *APIs* and *implementations* for managing local and remote ontology repositories:

  - An abstract interface for accessing various types of ontologies independently of the regarding storage mechanisms is provided by the **KAON API** and the **RDF API** (cf. chapter 3).
  - Currently three different **implementations** of the KAON API and the RDF API are available: Whereas the **Engineering Server** (cf. chapter 4) is an ontology server using a scalable database representation of ontologies, the **RDF Server** can be used for storing and accessing RDF models. **APIonRDF** (cf. chapter 3) is a main-memory implementation of the KAON API on the RDF API.

- **The KAON Extensions** are a collection of optional components not included in the standard distribution of KAON.

  - **DLP** (*Description Logic Programs*) supports efficient ontology reasoning by mapping Description Logic into Logic Programs.
  - **KAON Server** can be considered as Application Server for the Semantic Web, which provides a generic infrastructure to facilitate plug'n'play engineering of ontology-based applications.
  - **KAONtoEdit** is a plug-in for OntoEdit [oG03], which allows to work directly on implementations of the KAON API in order to load, modify and store KAON ontology models.

- **TextToOnto** (cf. chapter 5) is a KAON-based tool suite supporting the ontology engineering process by providing a collection of independent tools for ontology learning and maintenance.

KAON Architecture While we have so far provided an overview on the components and tools that are part of or related to KAON, we now want to focus on the rather technical interplay of some of those components, i.e. we intend to give a coarse outline of KAON's functional architecture distinguishing between APIs, implementations of those APIs and data sources to be accessed.

Figure 1.2 illustrates some of the interactions between KAON's APIs and reference implementations and highlights the central role of the KAON API. Each

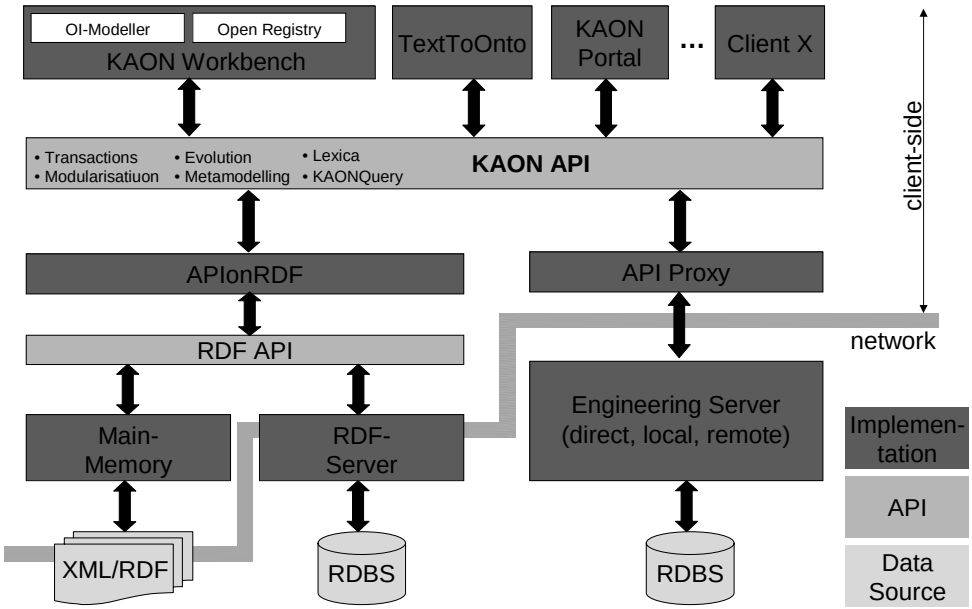

Figure 1.2: KAON Architecture Overview

client, e.g. KAON's ontology editor OI-Modeler, accesses KAON ontologies and instances independently of the storage mechanism via KAON API. In so doing, the OI-Modeler for example employs KAON's ontology evolution facilities integrated into KAON API.

As already mentioned , APIonRDF represents an in-memory implementation of the KAON API to access RDF-based data sources via the RDF API. For the RDF API in turn exist two reference implementations: On the one hand, a simple main memory implementation including RDF parser and serializer. On the other hand, the RDF Server which implements the RDF API remotely and allows for persistently storing RDF ontology models in relational databases and hence enables transactional ontology modification.

Sketched on the right-hand side, the API Proxy depicts an implementation of the KAON API that acts as a client-side proxy for various types of the KAON Engineering Server (cf. Chapter 4). That Engineering Server, being accessed remotely via an API Proxy, features mechanisms to store KAON ontologies in relational databases, to distribute change notifications (thus allowing for multi-user ontology engineering), and to bulk-load ontology elements.

For a more thorough and detailed depiction of KAON's architecture the interested reader is referred to [Vol04, MMV02].

1.3 Reader's Guide

We will introduce in this document the following components of and related to KAON:

- **OI-Modeler** (ontology editor) in chapter 2
- **KAON API** (programming interface for developers) in secion 3
- **KAON Engineering Server** (server for distributed ontology engineering, to be used in combination with OI-Modeler as front-end) in chapter 4
- **TextToOnto** (workbench for ontology learning from texts, feeds learned ontologies into the OI-Modeler) in chapter 5

We will conclude with an outlook of future development trends in chapter 6. Finally, an overview on download and installation information is presented in Appendix A.

Chapter 2

Ontology Editor OI-Modeler

OI-Modeler is KAON's tool for ontology creation and ontology maintenance[1]. The OI-Modeler's main goal is to allow scalability for editing large ontologies and to incorporate some basic usability issues related to ontology management and evolution.

The goal of this chapter is to introduce the reader to the OI-Modeler's main features. For that purpose we use a running example about the construction of a tiny ontology, presenting OI-Modeler's basic functionalities. For further details on how to work with the OI-Modeler we refer to "OI-Modeler User's Guide" [Kar02].

This chapter is mainly for end-users of the OI-Modeler. Programmers may want to proceed to the following Chapter 3 which describes the KAON API and how to access it.

2.1 Create New OI-Model

After having launched the KAON Workbench (by invoking `kaongui.bat`, cf. Section A.2) the user may work with the OI-Modeler, KAON's ontology editor. To work with an OI-Model, you can create a new ontology or open an existing one. When choosing "Create new OI-Model" from the "File" menu, a dialog box appears asking for the specifics of the ontology instance to be created (see Figure 2.1).

When speaking about working with an OI-Model, it is important to emphasize

[1]The "OI" here refers to "Ontology Instance". Hence, in the following we refer by the term "OI-Model" to an instance of an ontology.

that the ontology may be stored in different ways depending on the intended application or usage scenario. If a *really* voluminous ontology with lots of concepts and instances shall be created, the usage of the Engineering Server is advisable, since it employs a relational database system to store all entities involved (see Chapter 4). In general, however, it is fully sufficient to store the OI-Model in main memory, in particular when intending to get started with the OI-Modeler. Here, tab "RDF Models" has to be chosen from the dialog shown in Figure 2.1 and so the resulting ontology will be stored locally on the user's hard disc drive.

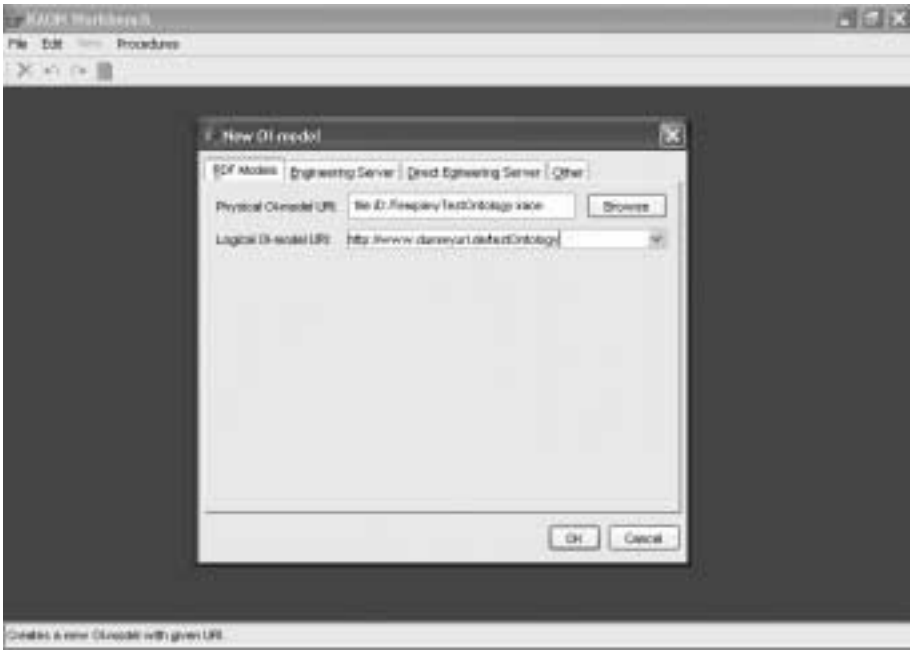

Figure 2.1: Creation of a new OI-Model

For our running example, however, we do not want to employ the Engineering Server, instead our ontology shall be stored in main memory. For that reason, we only have to provide a physical as well as a logical[2] URI for our ontology.

As illustrated in Figure 2.2, the OI-Modeler provides different views on the

[2]Each OI-Model has two URIs that uniquely identify it. The *physical* URI is the URI used to access the model. For example, if the model is located in file `D:\Temp\myRunningExampleOntology.kaon`, then the physical URI of the model will be `file:/D:/Temp/myRunningExampleOntology.kaon`. Each OI-Model also has a logical URI, which is independent from the physical one. E.g., a model may have the logical URI `http://kaon.semanticweb.org/myModel.kaon`, although it is not loaded from the web. A model's logical URI should be globally unique, whereas its physical URI typically is not globally unique, and is often relative to the system which processes the model.

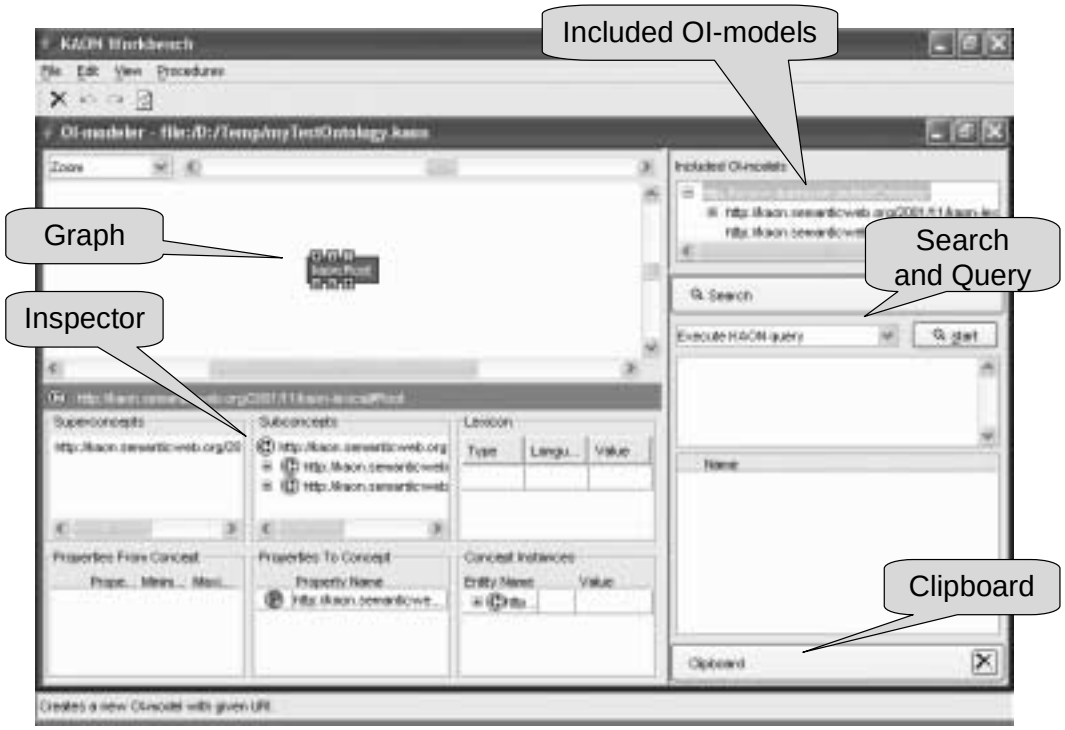

Figure 2.2: Main Window of the OI-Modeler

Ontology and allows to inspect its components (concepts, instances, properties and lexicon).

Graph The graph in the upper section of the window shows the ontology entities and the connections between them. The graph layout algorithms in OI-Modeler are based on an open-source TouchGraph[3] library.

Each graph node features up to six little arrows (see Figure 2.3). By clicking on those arrows related entities can be expanded, so that the user can successively browse through the ontology. For example, for a concept the user may expand that concept's sub- and super-concepts, properties to and from this concept, the concept's instances as well as its spanning instances. Regarding the notion of spanning instances please refer to [MMV02].

Inspector In the inspector you can find all information about the ontology entity that is currently selected in the graph. Thus, the inspector's appearance adapts to the type of entity (concept, property, or instance) currently

[3] `http://www.touchgraph.com`

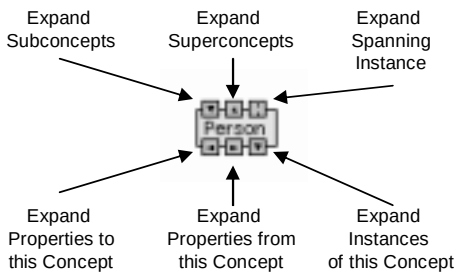

Figure 2.3: Characteristics of Nodes in OI-Modeler's Graph Visualization

selected. If, for example, a concept is selected information about that concepts and its super- and subconcepts are displayed, also the properties to and from the concept and the concept instances. Furthermore, the inspector may be used to directly create new (sub-)concepts (see below).

Included OI-Models The OI-Modeler allows for including ontologies. This means that the user is able to combine two (or more) ontologies to one ontology. An OI-Model always consists of two basic or system ontologies: The `kaon-root` and the `kaon-lexical`. To include an OI-Model one can choose "Open and include OI-Model" in the "Edit" menu. Then, a new window opens and the source of the OI-Model to be included can be selected. Please refer to Section 2.8.

Search and Query With the search function, one can easily find different named nodes. It is possible to search for concepts, instances, and properties and to perform a keyword-based search for any matching item.

Furthermore, KAON provides a query language KAON Query suited for posing queries to the ontology.

The search and query facilities integrated into OI-Modeler are described in more detail in Section 2.9.

Clipboard The Clipboard is for copy and paste use. It allows to copy entities to the clipboard, store them there, and later use them by pasting into the ontology. Please refer to Section 2.10 for further details.

2.2 Add New Concept

OI-Modeler provides three ways to create a concept. You can add new concepts by

1. using the “Edit” menu and choosing “New Concept...”,
2. opening the context-menu (right mouse-button) in the graph window and choosing “New Concept...”,
3. using the Inspector and opening the context-menu there.

Figure 2.4 illustrates the first of the three above-mentioned ways.

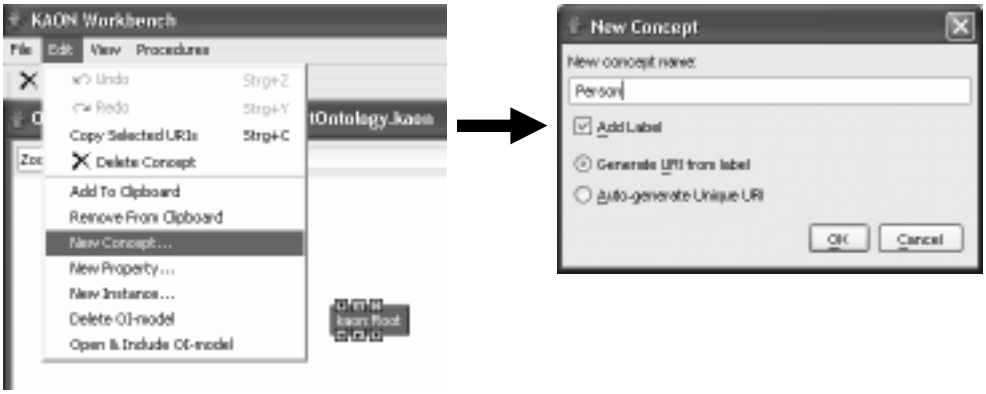

Figure 2.4: Adding a Single Concept Person

Sub-concepts are thematic refinements of concepts. When intending to add a sub-concept c_s to an existing concept c, first concept c has to be selected – as a consequence, details regarding that concept are displayed in the Inspector. Now, the sub-concept can be added to c with one of the three possible ways mentioned before. In Figure 2.5 the third alternative (using the context menu in the Inspector) is shown.

2.3 Add New Property

The procedure to add a property to an ontology model is almost the same as creating a concept, i.e. the user can choose between three ways just as in the case of adding concepts as described in Section 2.2. However, the OI-Modeler differs between two kinds of properties: properties from and to a concept.

Furthermore, when speaking about OI-Modeler’s facilities to edit *properties* we ought to clarify what we mean with that term and with terms often used additionally or synonymously such as attribute and relation.

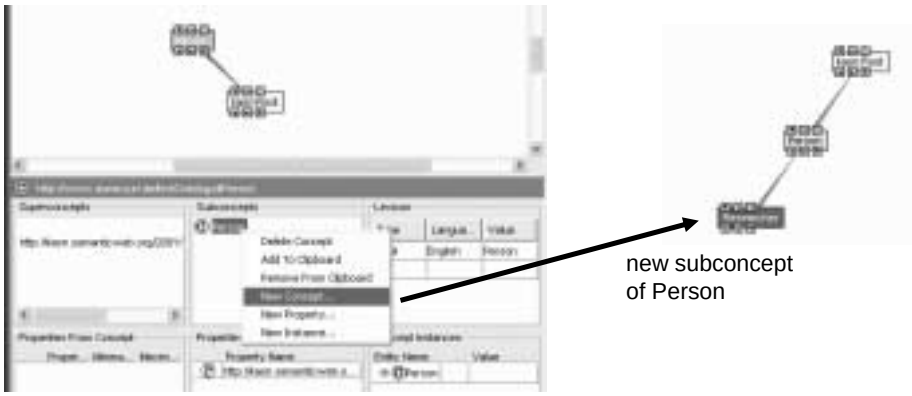

Figure 2.5: Adding Sub-Concept Researcher to Concept Person

Relation/Relationship is used as a generic term to refer to any kind of property that interlinks concepts.

Properties *from* a Concept are relations to other concepts (instances). In the graph view they are displayed in the same way as *attributes* of a concept.

Properties *to* a Concept are relations from other concepts to the concept under consideration.

Attributes do not connect two or more concepts, but they are rather used to express a certain characteristic of a concept, e.g. a description, long name, or URL. Attributes are often typed as XML Schema data types.

We will use mainly the terms property and attribute. Please note that both are inherited equally from super-concepts to sub-concepts.

In Figure 2.6 a later development stage of the ontology is shown. There, you can see what is meant by those different types of properties, how they relate to concepts, and how they are displayed in OI-Modeler's graph view. In the meantime additional concepts (e.g. Paper and Researcher) have been added. Moreover, there are also two new properties: First, there is the HASWRITTEN property which, on the one hand, represents a property from concept Researcher and, on the other hand, a property to concept Paper.

Then, there is the AGE property from concept Researcher. This property represents an attribute of concept Researcher. Note, that when selecting a specific property in the graph view, the Inspector displays the characteristics of that property. Among those, there are also checkboxes that may be used to declare that property to be

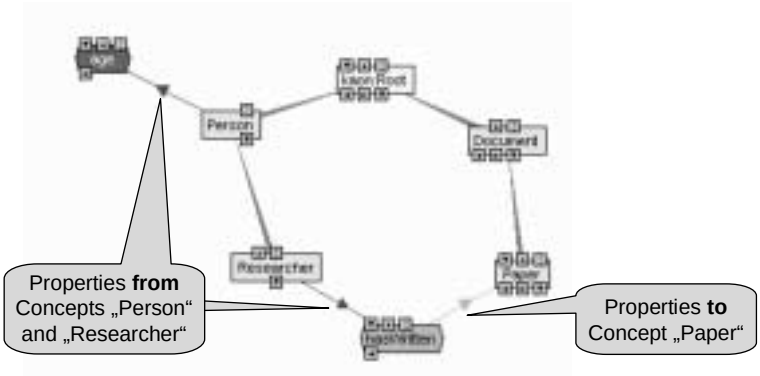

Figure 2.6: Properties from and to a Concept

- an attribute
- a symmetric property
- a transitive property

So, for the AGE property, the attribute flag has been set.

Moreover, it is possible to specify an inverse property to another one. For example, the inverse property for HASWRITTEN may be called HASAUTHOR being a property *from* concept Paper *to* concept Researcher.

Just as sub-concepts are thematic refinements of concepts, sub-properties represent thematic refinements of properties. OI-Modeler also support the refining of properties by creating sub-properties, whose creation we do not describe in more detail here.

2.4 Add New Instance

Just as in the case of concepts and properties, users can add a new instance by three different ways. In each case, however, you first have to choose the concept you want to add the instance to. Then, you may

1. use the “Edit” menu’s entry “New Instance...”
2. use a concept’s context menu (right mouse button) and choosing “New Instance...”

3. use the Inspector's table "Concept Instances" in the right by making a right-click on the respective concept name to which an instance shall be added and choosing "New Instance..." from the context menu opened.

In any way a new window appears asking the user to provide a name for the instance: Figure 2.4 illustrates the third way to create an instance and shows the mentioned window asking for the name. The resulting ontology, after having added two further instance, is sketched in the right part of that figure.

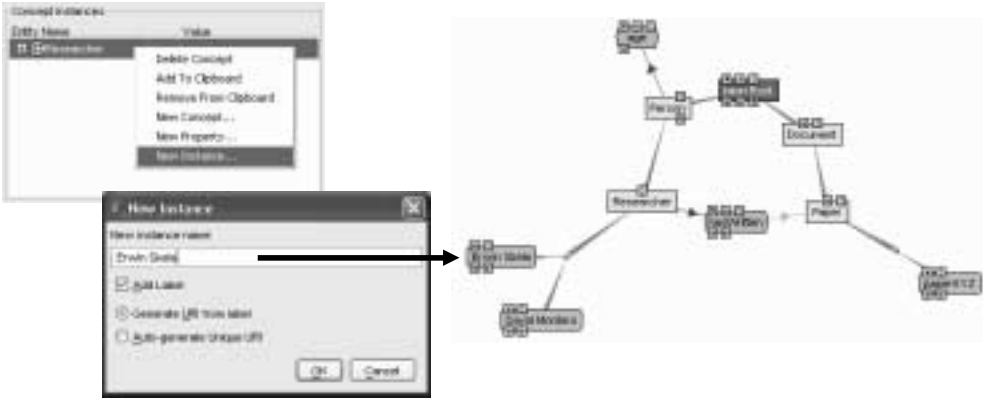

Figure 2.7: Creation of a New Instance

2.5 Instantiate Properties

The properties between the instances are the relations between these instances.

First, we want to add an attribute instance AGE to the instance `Erwin Skela`. To do so, there are the three usual ways (via "Edit" menu, via context menu for that instance's graph node, and via context menu for that instance in the inspector). In any way, the menu item "Add Attribute Instance..." has to be chosen. OI-Modeler then present a sub-menu containing all attributes that are defined for the instance's corresponding concept. In our case (cf. Figure 2.8) there is of course only the attribute AGE listed. After having performed that step, an attribute instance is created which initially does not contain a concrete value yet.

To assign a specific value (bottom-right part of Figure 2.8) you may edit the input field in the Inspector corresponding to that newly created attribute instance.

Figure 2.8: Instantiating an Attribute

Next, we want to connect to instances using the property HASWRITTEN. To be exact, we want to link the instance `David Montero` with instance `paperXYZ` via the HASWRITTEN property.

There are two different ways to connect instances through a property:

1. Use the graph view and select the (source) instance you want to connect. Then do a right-click on the instance that ought to be connected (target instance). From the opening menu choose “Connect Instances Using”, and from its sub-menu the respective property (here: HASWRITTEN).

2. It is also possible to press and hold the left mouse button (on the source instance) and then to drag the cursor to the (target) instance you want to connect. A line appears, as shown in the picture and if you disengage the mouse button, a menu appears and you can choose the property (here: HASWRITTEN).

Both options are visualized in Figure 2.9.

2.6 Delete Concept

To remove a concept you can use one of the three following options. First, select the concept to be deleted, then

1. do a right-click on it and choose “Delete Concept” from the context menu opening,

2. select the concept’s name in the Inspector, do a right-click on it and choose “Delete Concept” from the context menu opening.

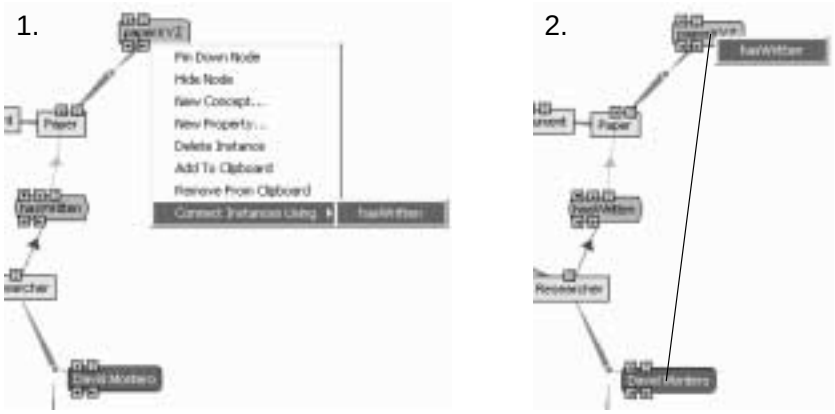

Figure 2.9: Instantiating a Property

Deleting of properties and instances works similar to the here described deletion of concepts.

While adding new elements to an ontology in general does not induce needed follow-on operations, the removal of an entity from the ontology may easily do so. For example, when deleting a certain concept, one upcoming question is how to handle the concept's instances. As questions like that are of high importance for ontology evolution, the following section is devoted to KAON's current features concerning ontology evolution.

2.7 Evolution Features

Industrial and academic environments are very dynamic, inducing changes to application requirements. Using an ontology-based system, often the underlying ontology must be evolved in order to adapt to those changes. As ontologies grow in size, the complexity of change management increases, thus requiring a well-structured ontology evolution process.

In KAON the user is provided with capabilities to customize and control the process of ontology evolution. It employs so-called evolution strategies that encapsulate certain policies for evolution with respect to the user's demands (see [SMMS02] and [SSH02]). As those evolution features are an integrated element of KAON, their usage fully available from within the OI-Modeler.

Note, that evolution reversibility services are provided as special service of KAON API, allowing different applications to reuse these powerful features.

2.7.1 Using Evolution Features

Potentially, an ontology change might corrupt the instances, dependent ontologies as well as application programs running against the ontology and/or the data base. With option “Set-up Evolution Parameters” from the “Procedures” menu the user is allowed to define the strategy how the OI-Modeler handles changes in the ontology, e.g. the deletion of concepts.

The window shown in Figure 2.10 shows the parameters by which users of the OI-Modeler may decide for a specific ontology evolution strategy. So, for example, problems are addressed like the handling of orphaned concepts that come into existence after a (parent) concept has been deleted, or the handling of properties that do not have a domain concept any more. The evolution strategies shown are rather self-explanatory.

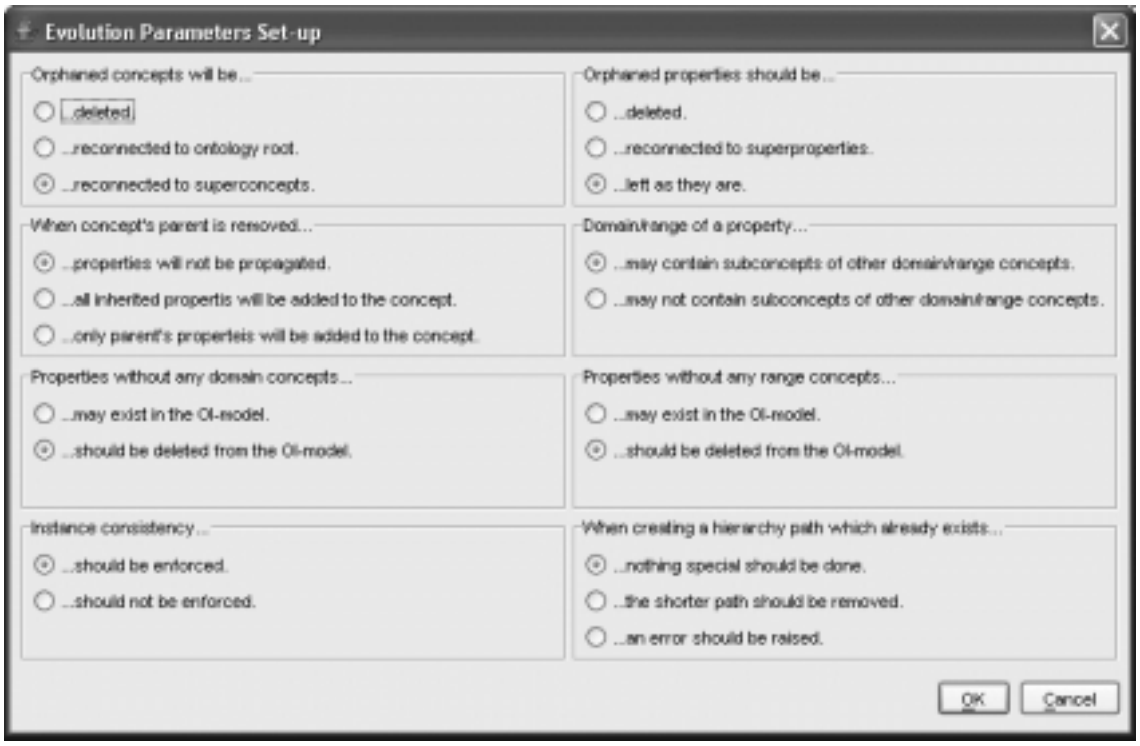

Figure 2.10: Setting Up Parameters for Ontology Evolution Strategies

Changes to the ontology are performed by assembling elementary and composite changes into a sequence which is based upon the respective evolution strategy. To ensure atomicity of updates to the ontology (and thus to allow for do/undo

functionality), either no or all changes from that sequence have to be processed.

In the "Procedures" menu you find the option "Show Evolution Details". If that option is not checked, changes (e.g. concept deletion) are performed immediately. By checking that option the mentioned extended sequence of changes is presented to the user for approval.

From our current version of the ontology (compare Figures 2.7 and 2.9) we now intend to delete concept Paper. Then, the dialog shown in Figure 2.11 appears and displays the sequence of all (atomic) changes that have to be performed in accordance to the evolution strategy chosen. Obviously, the removal of concept Paper induces the deletion of property HASWRITTEN and hence of its property instantiation (`David Montero` HASWRITTEN `paperXYZ`), the deletion of instance `paperXYZ` and of course the desired deletion of concept Paper.

To further aid the understanding why certain changes have to be performed, related elementary change actions are grouped together in a tree-like structure increasing the understandability of why some changes/side effects have to be executed. After those changes have been reviewed and approved by the user, they are passed to the ontology and performed.

2.7.2 Undo / Redo Functionality

There are various circumstances under which it may be desirable to reverse the effects of ontology evolution, e.g.

- The ontology engineer may fail to understand the actual effects of his/her changes and may approve a change that actually should not have been performed.
- Sometimes it is helpful to change the ontology for experimental purposes.
- When working collaboratively on an ontology, several ontology engineers may have different ideas on how the ontology ought to evolve.

It is obvious that for each elementary change there is exactly one inverse change that, when applied, reverses the effect of the original change. Based on the infrastructure described in the previous section, it is not hard to realize the requirement for reversibility of ontology engineering actions and to provide an appropriate undo/redo functionality: 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.

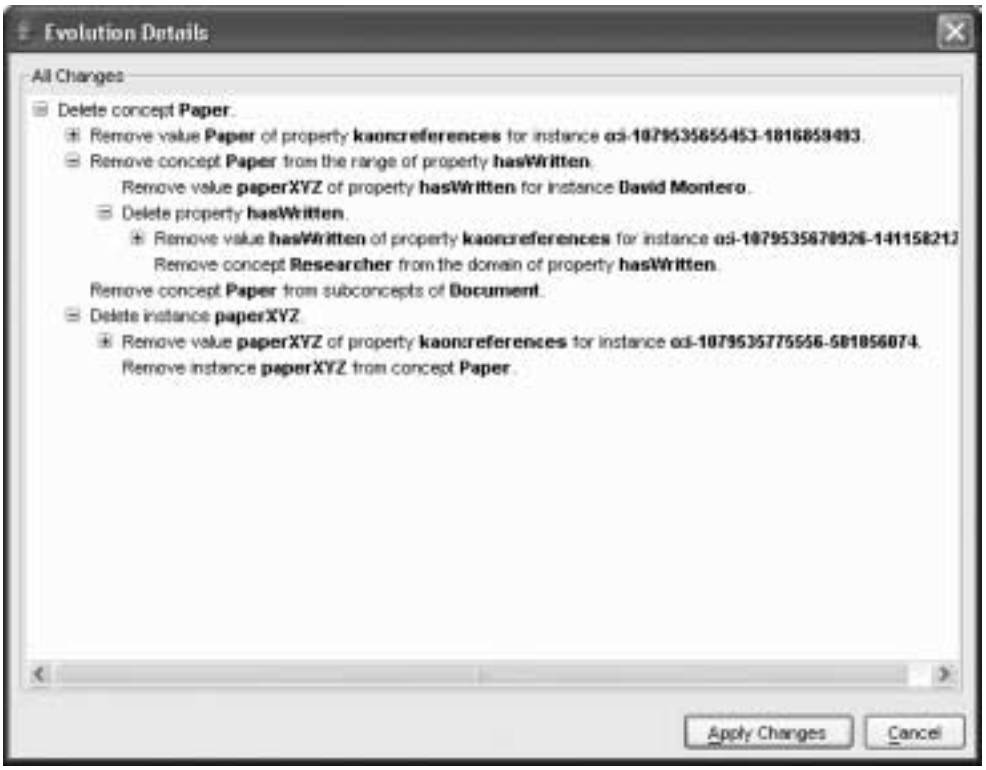

Figure 2.11: Presenting Evolution Details to the User for Approval

In other words, reversibility means undoing *all* effects of some change, which is in general not the same as requesting an inverse change manually. For example, if a concept is deleted from a concept hierarchy, its subconcepts will need to be deleted as well, attached to the root concept, or attached to a parent of the deleted concept. Reversing such a change is of course not equal to recreating the deleted concept – one needs, also to revert the concept hierarchy into its original state.

In OI-Modeler the undo and redo features are provided via the “Edit” menu as shown in Figure 2.12.

Ontology Evolution Log File The problem of reversibility is typically solved by creating evolution logs. An evolution log stores information about each change in the system, allowing to reconstruct the sequence of changes. With each change applied to the ontology the evolution log additionally associates further information [MSSV02], like meta-information such as change description, cost of change, time required to perform the change, cause of the change, or identity of the change’s author.

Figure 2.12: Undoing and Redoing Changes

The following excerpt from a log file illustrates some of the information put into that tracking facility. It refers to adding of concept Manual as sub-concept of Document to the ontology.

```
<a:AddEntity rdf:ID="i-1079545047999-1104860243"
    a:inOIModel="http://www.dummyurl.de/testOntology"
    a:version="85"  >
    <a:has_previousChange rdf:resource="#i-1079545047999-24213731"/>
    <a:has_previousHistoryChange rdf:resource="#i-1079545047999-24213731"/>
    <a:has_referenceInstance>
      http://www.dummyurl.de/testOntology#i-1079545046317-1018711561
    </a:has_referenceInstance>
</a:AddEntity>
<a:AddEntity rdf:ID="i-1079545047999-1343051864"
    a:firstChangeInAGroup="true"
    a:has_referenceInstance="http://www.dummyurl.de/testOntology#Manual"
    a:inOIModel="http://www.dummyurl.de/testOntology"
    a:version="85">
    <a:has_previousChange rdf:resource="#i-1079545041129-1524299176"/>
    <a:has_previousHistoryChange rdf:resource="#i-1079545041129-1524299176"/>
</a:AddEntity>
[...]
<a:AddInstanceOf rdf:ID="i-1079545047999-24213731"
    a:has_referenceConcept="http://www.dummyurl.de/testOntology#Document"
    a:has_referenceInstance="http://www.dummyurl.de/testOntology#Manual"
    a:inOIModel="http://www.dummyurl.de/testOntology"
    a:version="85">
    <a:has_previousChange rdf:resource="#i-1079545047999-1343051864"/>
    <a:has_previousHistoryChange rdf:resource="#i-1079545047999-1343051864"/>
</a:AddInstanceOf>
<a:AddPropertyInstance rdf:ID="i-1079545047999-345568492"
    a:has_referenceProperty=
      "http://kaon.semanticweb.org/2001/11/kaon-lexical#references"
    a:has_referenceTargetInstance="http://www.dummyurl.de/testOntology#Manual"
    a:inOIModel="http://www.dummyurl.de/testOntology"
    a:version="85">
    <a:has_previousChange rdf:resource="#i-1079545047999-2048209500"/>
    <a:has_previousHistoryChange rdf:resource="#i-1079545047999-2048209500"/>
    <a:has_referenceSourceInstance>
```

```
      http://www.dummyurl.de/testOntology#i-1079545046317-1018711561
    </a:has_referenceSourceInstance>
</a:AddPropertyInstance>
<a:AddPropertyInstance rdf:ID="i-1079545047999-359720445"
    a:has_referenceProperty="http://kaon.semanticweb.org/2001/11/kaon-lexical#value"
    a:has_referenceTargetObject="Manual"
    a:inOIModel="http://www.dummyurl.de/testOntology"
    a:version="85">
    <a:has_previousChange rdf:resource="#i-1079545047999-345568492"/>
    <a:has_previousHistoryChange rdf:resource="#i-1079545047999-345568492"/>
    <a:has_referenceSourceInstance>
      http://www.dummyurl.de/testOntology#i-1079545046317-1018711561
    </a:has_referenceSourceInstance>
</a:AddPropertyInstance>
```

2.8 Inclusion of other OI-Models

As mentioned before OI-Modeler is capable of including managing several ontologies in parallel an of including entire ontologies into another one. The semantics of the inclusion are described in detail in [MMS+03].

As depicted in Figure 2.2 OI-Modeler's main window. Each OI-Model consists per default of two so-called system ontologies, `kaon-lexical` and `kaon-root`. Note, that it is possible to mask these system ontologies (in the graph view) by deselecting the option "System Objects" from the "View" menu.

To include an OI-Model choose "Open and Include OI-Model" in the "Edit" menu. A new window opens and you can select the source of the OI-Model you want to include (see Figure 2.13).

Figure 2.13: Inclusion of other OI-Models

2.9 Querying and Searching

Queries in KAON (and thus in the OI-Modeler) are an experimental feature that from the perspective of the KAON development team is far from being finished. The primary role of the current support for querying in KAON is to gather feedback in order to improve these features in next versions of KAON. It is quite likely that the query syntax and/or the API will change significantly in the future.

KAON provides an experimental conceptual query language (KAON Query) that allows easy and efficient locating of elements in KAON OI-Models. However, as already mentioned queries in KAON are under development, so the interested reader is referred to [KK04].

Keyword-Based Searching With the search function, the user can easily find different named nodes. It is possible to search for

- anything: Every matching entity in the ontology will be displayed.
- concepts: Matching concepts will be displayed.
- instances: Matching instances will be displayed.
- properties: Matching properties will be displayed.

Figure 2.14 shows how to search for the keyword Person in our running example – that search returns two results: The concept Person as well as a spanning instance `Person`. For the notion of a spanning instance please refer to [MMV02].

Figure 2.14: OI-Modeler's Searching Facility

The results are presented as a list matching the search string. In particular, the user can also paste selected results into the "Graph window" via drag & drop.

2.10 Using the Clipboard

The clipboard is a convenient way to employ copy & paste functionality when working with an ontology. By opening an entity's context menu via right-clicking on it (e.g. in the graph view or in the Inspector) and choosing "Add to Clipboard", or by choosing "Add to Clipboard" from the "Edit" menu, the respective entity is copied to the clipboard (see Figure 2.15).

Figure 2.15: OI-Modeler's Clipboard

After having been added to the clipboard, the respective entity may be used via drag & drop and can that way be integrated into the graph view or into the Inspector.

For example, the copied concept Researcher may be dragged onto another concept Scientific Staff and would thus be made a sub-concept of that concept.

2.11 Other Features

In this section we give a very brief overview of some of OI-Modeler's other features.

Loading/Saving the Workspace The entries "Load Workspace" and "Save Workspace" from the "File" menu allow the user to load/save a workspace containing all windows of the previous/current work session.

Open/Save OI-Model In analogy to creating a new OI-Model (cf. Section 2.1) it is possible to load a previously saved OI-Model by choosing the corresponding entry from the "File" menu.

Duplicate OI-Model This option from the "File" saves the current OI-Model under another name and hence duplicated it.

Copy to new OI-Model With this option from the "File" menu you can copy an existing OI-Model into a new one. This replica has the reference to the original OI-Model. If you choose this function, the same window as in the function "Open OI-Model" appears, and you can choose where to save the backup.

View Specifics The view onto the user interface can be customized (via the "View" menu) to, e.g.

- refresh the graph representation (shortcut F5),
- show only selected nodes (shortcut F4),
- use an "Incremental Search Graph": With that function you can search the graph for e.g. a keyword incrementally,
- show/hide the entire graph view, Inspector, and clipboard,
- hide system objects: An OI-Model consists of three objects: `kaon-lexical#Root`, `kaon-lexical#language`, and `kaon-lexical#LexicalEntry`. By switching of the "System Objects" you only visualize the `kaon-root` and so the ontology gets more clear because less concepts and instances are shown in the graph and the inspector.

Language Parameters Language parameters are to be found in the "View" menu. The user can choose between English, German, French, Spanish, Arabic, and Chinese.

Context Menus As mentioned most entities in OI-Modeler feature context menus whose appearance varies from entity to entity. For a detailed description of context menus we refer to [Kar02].

Entity Icons To easily distinguish concepts, properties, and instances in the Inspector OI-Modeler utilizes several specific icons as shown in Figure 2.16.

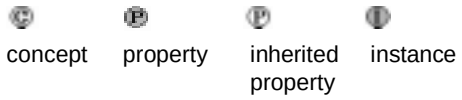

Figure 2.16: OI-Modeler's Icons for Entities

Lexical Layer All ontological entities are considered as language neutral in KAON. On the lexical layer, lexical descriptions referring to different entities in the KAON representation vocabulary may be defined. The lexicon is always accessible within the Inspector. A lexical entry is a lexicalization of a concept, attribute, relation, and instance. Several types of lexical entries are defined. The standard lexical description are multilingual labels that may be used for the user's interface. A label is a specific kind of a lexical entry, describing a primary descriptor of an ontological or knowledge base entity. Another kind of lexical entries are morphologically reduced word stems that may be used by a natural language processing system. shows lexical elements available in OI-Model.

A synonym is a specific kind of a lexical entry, describing synonymous words for an ontological or knowledge base entity. The documentation allows you to enter a text description of the ontological entity. The lexical

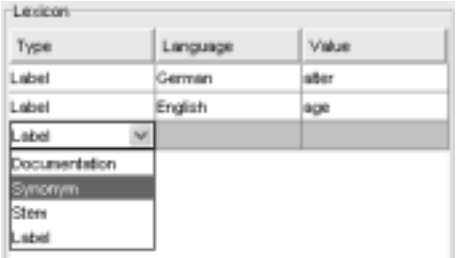

Figure 2.17: OI-Modeler's Lexicon

layer also allows you to create multilingual ontologies. As shown in the picture it is possible to label a concept (in this case the property AGE) in different languages.

Chapter 3

KAON API Description

3.1 Overview

The KAON API is a set of interfaces developed in order to offer programmatic access to KAON ontologies by providing classes such as `Concept`, `Property` and `Instance`. Because the API does not make any assumptions about the underlying ontology persistence mechanisms it totally decouples the user from all details of ontology access and storage. It is the concrete implementation of the API, which determines, for example, whether an ontology created with the KAON API will be stored in an RDF file or a local or remote database (cf. subsection 3.3.1). Currently three different implementations of the KAON API are available:

Engineering Server The engineering server (cf. section 4) is an ontology server using a scalable database representation for storing KAON ontologies. It is optimized for ontology engineering by offering scalable, transactional and concurrent access to ontology information.

RDF Server The RDF server uses the RDF API for storing and accessing RDF models. Although quite similar to the engineering server supporting transactions and multi-user operations, it does not provide any functionalities for conflict detection or bulk-loading.

APIonRDF Implementation The main memory implementation of the KAON API on the RDF API provides in-memory model manipulation for KAON. When you download the standard KAON distribution (cf. Appendix A) this is the default setting.

Since the main memory implementation can be considered as the standard implementation of the KAON API this chapter will focus on APIonRDF.

3.2 Important Features

Meta Modeling Meta Modeling means that a concept or a property may be considered as an instance of a meta-concept. Such an instance is then called the *spanning-instance* of the regarding concept or property. It can be retrieved by using the `getSpanningInstance()` method, which is defined in the `Entity` interface.

Evolution Strategies Since each change to an ontology might leave the model in an inconsistent state, the KAON API supports the use of evolution strategies (cf. subsection 3.3.11) for computing sequences of additional changes, which are necessary for safely performing the requested change.

Change Notifications Implementing the Observable design pattern the OI-Model interface allows listeners to receive notifications about model updates.

Lexical Layer Lexical information such as labels or documentation can be added to an OI-Model by assigning `LexicalEntry` objects to the instance interpretation of concepts, properties or instances.

Modularization KAON as well as the KAON API support building ontologies modularly by means of ontology inclusion. Each OI-Model may include other OI-Models provided that those are of the same type (e.g. RDF-based or server-based). Because the inclusion is implemented as a link, not as a copy, all changes to the included OI-Model will immediately affect the including OI-Model.

3.3 Examples

This section gives a brief introduction on using the KAON API on the basis of a small sample ontology which is also used in chapter 2.

3.3.1 Select Implementation

The KAONConnection interface is provided by the KAON API in order to separate clients from different API implementations. Each of these implementations must include at least one implementation of KAONConnection, which can be used by clients in order to access the regarding OI-Model implementation.

As long as a user is working with only one implementation of the KAON API instances of KAONConnection may be created directly by using the appropriate constructor (e.g. `new KAONConnectionImpl()`). If an application should work with any or with more than one implementation the KAONManager class has to be used to obtain a KAONConnection object. The following listing 3.1 demonstrates how to get a KAONConnection object for APIonRDF.

```
HashMap parameters = new HashMap();
parameters.put( KAONManager.KAON_CONNECTION,
    "edu.unika.aifb.kaon.apionrdf.KAONConnectionImpl" );
KAONConnection connection =
    KAONManager.getKAONConnection( parameters );
```

Listing 3.1: KAONConnection (APIonRDF)

Each set of parameters passed to `KAONManager.getKAONConnection` must contain a `KAON_CONNECTION` parameter which determines the type of connection, which is returned. A direct connection to an Engineering Server, for instance, would require the following `KAON_CONNECTION` parameter:

```
  parameters.put( KAONManager.KAON_CONNECTION,
"edu.unika.aifb.kaon.engineeringserver.client.
DirectKAONConnection" )
```

In addition to `KAON_CONNECTION` further parameters, like user name or password might be necessary depending on the implementation and the type of connection to be used.

3.3.2 Create New OI-Model

Once a KAONConnection object has been obtained it can be used to create a new OI-Model (cf. listing 3.2).

```
String m_sPhysicalURI = "file:/f:/temp/myTestOntology.kaon";
String m_sLogicalURI = "http://www.dummyurl.de/testOntology";
OIModel oimodel = connection.createOIModel( m_sPhysicalURI,
```

```
    m_sLogicalURI );
```

Listing 3.2: Create new OI-Model

Each OI-Model is uniquely identified by two URIs - a physical and a logical one, which are totally independent from each other.

Physical URI The structure of the physical URI, which is used to access the OI-Model, depends on the KAON API implementation used. If an OI-Model is locally stored in `f:\temp\myTestOntology.kaon`, for instance, its physical URI would be `file:/f:/temp/myTestOntology.kaon`.

Logical URI A logical URI can be chosen freely, but in contrast to the physical URI it has to be globally unique. For example, the above mentioned OI-Model could have the logical URI `http://www.dummyurl.de/testOntology`, although this URI does not really exist on the web.

3.3.3 Open OI-Model

An existing OI-Model can be opened by `connection.openOIModelPhysical( m_sPhysicalURI )`.

The method `connection.openOIModelLogical` only works for some well-known models (e.g. the lexical model) pre-registered with KAONConnection and for OI-Models, which are known to KAONConnection, because they have been previously opened by their physical URIs.

3.3.4 Add New Concepts

The following code fragment (cf. listing 3.3) creates two new concepts, *Person* and *Document*.

```
Concept person =
    oimodel.getConcept( m_sLogicalURI +"#Person" );
Concept document =
    oimodel.getConcept( m_sLogicalURI +"#Document" );
```

Listing 3.3: Create new concepts

Since the method `OIModel.getConcept` always returns a concept (even if there is no concept with the specified URI in the OI-Model), it can be used for both

creating new concepts and accessing existing ones. The only parameter required by `OIModel.getConcept` is a unique URI for the concept to be created or retrieved. Very often, the logical URI of the OI-Model is used as a prefix for newly created concepts, because in this case the URIs of all model entities will be serialized relative to the model's URI, when the OI-Model is serialized in RDF.

As soon as the two new concepts have been created they can be added to the OI-Model as shown below:

```
List changes = new LinkedList();
changes.add( new AddEntity( person ) );
changes.add( new AddEntity( document ) );
oimodel.applyChanges( changes );
changes.clear();
```

Listing 3.4: Add new concepts

With regards to ontology evolution (cf. subsection 3.3.10) the KAON API does allow for performing changes directly on the OI-Model. Therefore a list of change events has to be created and applied to the model. All available change events are located in the `edu.unika.aifb.kaon.api.change` package.

3.3.5 Add New SubConcepts

Adding a new subconcept to the OI-Model is very similar to adding a concept (cf. previous subsection), except that one additional change event is required. Listing 3.5 shows how to add a subconcept *Researcher* to the previously created concept *Person* and a subconcept *Paper* to *Document*.

```
Concept researcher =
    oimodel.getConcept( m_sLogicalURI +"#Researcher" );
Concept paper =
    oimodel.getConcept( m_sLogicalURI +"#Paper" );
changes.add( new AddEntity( researcher ) );
changes.add( new AddEntity( paper ) );
changes.add( new AddSubConcept( person, researcher ) );
changes.add( new AddSubConcept( document, paper ) );
oimodel.applyChanges( changes );
changes.clear();
```

Listing 3.5: Add new subconcepts

It is important to know that **each new subconcept has to be added to the OI-Model (`AddEntity`)** before it can be made a subconcept of any other concept

(`AddSubConcept`).

3.3.6 Add New Properties

Analogously to concepts (`OIModel.getConcept`) new properties can be created by `OIModel.getProperty`. The following code creates a new property *age* as an attribute to *Person*, a property *hasWritten(Researcher,Paper)* and a property *hasAuthor(Paper,Researcher)*.

```
Property age = oimodel.getProperty( m_sLogicalURI +"#age" );
Property hasWritten =
    oimodel.getProperty( m_sLogicalURI +"#hasWritten" );
Property hasAuthor =
    oimodel.getProperty( m_sLogicalURI +"#hasAuthor" );
changes.add( new AddEntity( age ) );
changes.add( new AddEntity( hasWritten ) );
changes.add( new AddEntity( hasAuthor ) );
changes.add( new AddPropertyDomain( age, person ) );
changes.add( new AddPropertyDomain( hasWritten, researcher ) );
changes.add( new AddPropertyDomain( hasAuthor, paper ) );
changes.add( new AddPropertyRange( hasWritten, paper ) );
changes.add( new AddPropertyRange( hasAuthor, researcher ) );
oimodel.applyChanges( changes );
changes.clear();
```

Listing 3.6: Add new properties

As shown by listing 3.6 up to four steps are required for adding a new property to the OI-Model:

- Creating a property (`OIModel.getProperty`)
- Inserting the property into the OI-Model (`AddEntity`)
- Defining the domain of the property (`AddPropertyDomain`)
- Defining the range of the property (`AddPropertyRange`)

Subproperties can be created by using the change event `AddSubProperty( superProperty, subProperty )` (cf. subsection 3.3.5).

3.3.7 Instantiate Concepts

Since an OI-Modelmay not only include concepts and properties, but also instances of both, the KAON API provides methods for the instantiation of concepts (cf. listing 3.7) and properties (cf. subsection 3.3.8).

The following example demonstrates how instances of the concepts *Researcher* and *Paper* can be added to the OI-Model.

```
Instance erwin =
    oimodel.getInstance( m_sLogicalURI +"#Erwin_Skela" );
Instance david =
    oimodel.getInstance( m_sLogicalURI +"#David_Montero" );
Instance paperXYZ =
    oimodel.getInstance( m_sLogicalURI +"#Paper_XYZ" );
changes.add( new AddInstanceOf( researcher, erwin ) );
changes.add( new AddInstanceOf( researcher, david ) );
changes.add( new AddInstanceOf( paper, paperXYZ ) );
oimodel.applyChanges( changes );
changes.clear();
```

Listing 3.7: Add new instances

3.3.8 Instantiate Properties

The instantiation of properties is very similar to the instantiation of concepts described in the previous subsection. The code fragment shown by listing 3.8 instantiates the properties *age* and *hasWritten(Researcher,Paper)* by assigning an age of 27 to *Erwin_Skela* and creating a *hasWritten* relation between *David_Montero* and *Paper_XYZ*.

```
PropertyInstance erwin_age_27 =
    oimodel.getPropertyInstance( age, erwin, "27" );
PropertyInstance david_hasWritten_paperXYZ =
    oimodel.getPropertyInstance( hasWritten, david, paperXYZ );
changes.add(
    new AddPropertyInstance( erwin_age_27 ) );
changes.add(
    new AddPropertyInstance( david_hasWritten_paperXYZ ) );
oimodel.applyChanges( changes );
changes.clear();
```

Listing 3.8: Instantiate properties

3.3.9 Pose Queries

KAON Query is an experimental conceptual query language, which is part of the KAON ToolSuite (cf. figure 1.1). It can be used programmatically by means of the KAON API for efficient locating of OI-Model elements. The following example demonstrates how to retrieve all instances of the concept *Paper*, which has been created in subsection 3.3.5.

```
String sQuery = "["+ m_sLogicalURI +"#Paper]";
Collection answer = oimodel.executeQuery( sQuery );
```

Listing 3.9: Pose queries

Since *Paper_XYZ* is the only instance of *Paper*, the collection returned by `OIModel.executeQuery` in the above listed code fragment contains only one element (cf. listing 3.10).

```
Query: [http://www.dummyurl.de/testOntology#Paper]
Answer: http://www.dummyurl.de/testOntology#Paper_XYZ
```

Listing 3.10: Query result

Since all instances of a certain concept can also be retrieved by using the `Concept.getInstances()` method, the same result would be returned by `paper.getInstances()`.

3.3.10 Remove Concepts

As shown by listing 3.11 a concept can be removed from an OI-Model by using the `RemoveEntity` change event.

```
changes.add( new RemoveEntity( paper ) );
List requestedChanges =
    oimodel.applyChanges( requestedChanges );
changes.clear();
```

Listing 3.11: Remove a concept

Similar change events are provided for removing subconcepts, properties or instances, for example.

The impact of removing concepts or other entities such as properties or instances from an OI-Model is demonstrated by the following RDF serialization (cf. subsection 3.3.12) of the sample ontology created in the previous subsections.

```
<rdf:RDF xml:base="http://www.dummyurl.de/testOntology"
    xmlns:rdf="&rdf;"
    xmlns:rdfs="&rdfs;"
    xmlns:a="&a;">
<a:Researcher rdf:ID="David_Montero"/>
<rdfs:Class rdf:ID="Document"/>
<a:Researcher rdf:ID="Erwin_Skela" a:age="27"/>
<rdfs:Class rdf:ID="Person"/>
<rdfs:Class rdf:ID="Researcher">
    <rdfs:subClassOf rdf:resource="#Person"/>
</rdfs:Class>
<rdf:Property rdf:ID="age">
    <rdfs:domain rdf:resource="#Person"/>
</rdf:Property>
</rdf:RDF>
```

Listing 3.12: Remove concept *Paper*

Obviously, the deletion of the concept *Paper* entails the deletion of

- its instance *Paper_XYZ*,
- the property *hasWritten(Researcher,Paper)*,
- the property *hasAuthor(Paper,Researcher)*,
- and the property instance *hasWritten(David_Montero,Paper_XYZ)*.

In order to manage such complex changes and to avoid potential problems like inconsistencies, for example, KAON provides different evolution strategies. The next subsection describes how these evolution strategies can be employed by means of the KAON API.

3.3.11 Use Evolution Strategies

Since each change to an OI-Model might potentially cause inconsistencies in this as well as in dependent ontologies, the KAON API supports the use of different evolution strategies (cf. [SMMS02] and [SSH02])). The following code fragment below shows how to use evolution strategies considering as example the deletion of *Paper* described in the previous subsection.

```
EvolutionStrategy strategy =
```

```
    new EvolutionStrategyImpl( oimodel );
changes.add( new RemoveEntity( paper ) );
List requestedChanges =
    strategy.computeRequestedChanges( changes );
oimodel.applyChanges( requestedChanges );
changes.clear();
```

Listing 3.13: Evolution strategies

Once an EvolutionStrategy object has been created it can be used in order to transform a list of change events (e.g. `RemoveEntity(Paper)`) into a new list containing all the change events, which are necessary for safely performing the desired changes. The content as well as the sequential order of this list depends on the evolution strategy implementation and the evolution parameters (`edu.unika.aifb.kaon.defaultevolution.EvolutionParameters`) specified by the user.

3.3.12 Serialization

The easiest way of serializing an OI-Model is to apply the `OIModel.save()` method, which stores the OI-Model either to a local file (determined by its physical URI) or, for instance, to a database - depending on which implementation of the KAON API is currently used.

Nevertheless, for debugging purposes it might be useful to create a textual output of the OI-Model . In this case the `RDFSerializer` class can be instantiated in order to write an RDF serialization to an output stream such as System.out (cf. listing 3.14).

```
RDFSerializer serializer = RDFManager.createSerializer();
OutputStreamWriter writer =
    new OutputStreamWriter( System.out );
serializer.serialize( ((OIModelImpl)oimodel).getModel(),
    writer, "UTF-8" );
writer.close();
```

Listing 3.14: RDFSerializer

The following extract shows an RDF serialization of the sample ontology, which has been created in the subsections 3.3.2 to 3.3.8.

```
<rdf:RDF xml:base="http://www.dummyurl.de/testOntology"
    xmlns:rdf="&rdf;"
    xmlns:rdfs="&rdfs;"
```

```
    xmlns:a="&a;">
<a:Researcher rdf:ID="David_Montero">
    <a:hasWritten rdf:resource="#Paper_XYZ"/>
</a:Researcher>
<rdfs:Class rdf:ID="Document"/>
<a:Researcher rdf:ID="Erwin_Skela" a:age="27"/>
<rdfs:Class rdf:ID="Paper">
    <rdfs:subClassOf rdf:resource="#Document"/>
</rdfs:Class>
<a:Paper rdf:ID="Paper_XYZ"/>
<rdfs:Class rdf:ID="Person"/>
<rdfs:Class rdf:ID="Researcher">
    <rdfs:subClassOf rdf:resource="#Person"/>
</rdfs:Class>
<rdf:Property rdf:ID="age">
    <rdfs:domain rdf:resource="#Person"/>
</rdf:Property>
<rdf:Property rdf:ID="hasAuthor">
    <rdfs:domain rdf:resource="#Paper"/>
    <rdfs:range rdf:resource="#Researcher"/>
</rdf:Property>
<rdf:Property rdf:ID="hasWritten">
    <rdfs:domain rdf:resource="#Researcher"/>
    <rdfs:range rdf:resource="#Paper"/>
</rdf:Property>
</rdf:RDF>
```

Listing 3.15: RDF serialization

Chapter 4

KAON Engineering Server

Currently, there are three different back-end implementations of the KAON API (cf. Section 3): the **Main Memory** implementation, the **RDF Server** as well as the **Engineering Server**. In this chapter we focus on the latter, most sophisticated KAON implementation, the Engineering Server.

4.1 Motivation

When building large ontologies (with probably thousands of concepts and relations and maybe millions of instances), it is with current standard technology rather infeasible to store that amount of data in main memory. In fact, when the ontology to be built exceeds a certain size, the usage of a database management system storing the mass of data is unavoidable. For that purpose, i.e. for managing the interaction with the database system, KAON includes an implementation of the so-called Engineering Server fulfilling that task.

In short, the Engineering Server is a **storage mechanism for KAON ontologies, based on relational databases and suitable for use during ontology engineering**. Its features include

- transactions,
- client-side caching with conflict detection,
- distributed change notification mechanism,
- bulk-loading of ontology elements,
- modularization (limited to models within the same database).

The Engineering Server has been tested with an ontology consisting of 100.000 concepts, 66.000 properties and 1.000.000 instances, where loading related information about 20 ontology entities took under 3 seconds, while deleting a concept took under 5 seconds.

4.2 Database Access

The Engineering Server is an ontology server that is optimized for ontology engineering. This optimization is in particular reflected in the database schema used by the server. Since ontology engineering often involves creating and deleting concepts, which should be multi-user capable and transactional, the engineering server has a database schema with a *fixed* number of tables.

An obvious alternative realization of an ontology servers might create a table per concept. However, this would make concept creation and deletion non-transactional, and in general, more heavy-weight. The schema employed by the Engineering Server is presented in Figure 1: One can see that it consists of a fixed number of tables. Indeed, this fact distinguishes the Engineering Server from other ontology servers implementations, which store all instances of a concept in a separate table (and thus require table creation and deletion every time a concept is created).

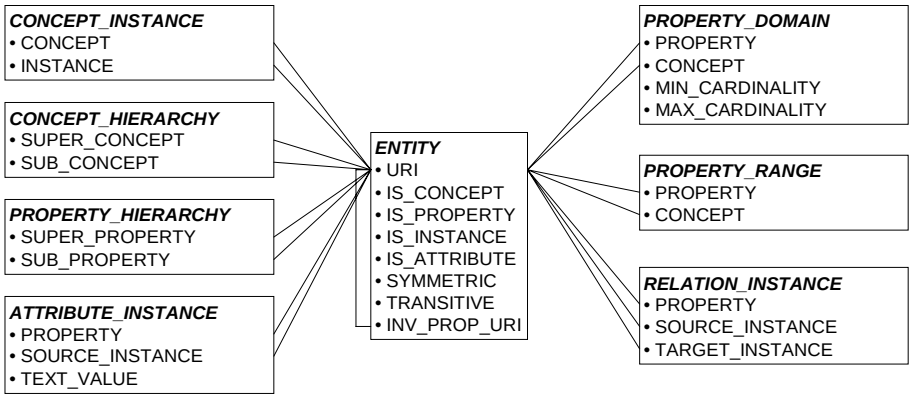

Figure 4.1: Engineering Server Database Schema

The Engineering Server offers scalable, transactional and concurrent access to ontology information. To achieve that, optimized bulk-loading is implemented, that allows fetching information about several ontology entities in one database request.

The Engineering Server has been tested with MSL SQL Server 2000. Also, it has successfully been run, but not thoroughly tested, on PostgreSQL, IBM DB2 7.2 and Oracle 9i. Although other databases may work, the Hypersonic database will *not* work, since it doesn't support many of standard features of relational databases that are needed for server operation. For users of IBM DB2 it is important to configure the DB2 database to use JDBC2 – please refer to the DB2 documentation for information on how to perform that.

4.3 Usage Scenario for the Engineering Server

Mainly, there are three ways of using the Engineering Server (Direct, Remote, and Local) which we describe in more detail in the subsequent sections.

In general, clients (e.g. ontology editors) access the Engineering Server through an appropriate KAON API implementation (e.g. the API Proxy, cf. Figure 1.1), which provides client-side ontology information caching, along with a mechanism for detecting incoherencies between the cache and the database. This feature thus significantly simplifies developing applications where ontology entities are loaded and kept at the client across transaction boundaries.

Before using the Engineering Server, it is necessary to create the necessary schema in the database used. For this, the Engineering Server comes with a number of scripts (SQL scripts, database-specific) whose execution results in the creation of the necessary database tables. If those scripts can be executed without errors, the Engineering Server is ready for usage. For more details concerning obtaining and installing the Engineering Server please refer to Appendix A.

Direct Engineering Server The *Direct Engineering Server* corresponds to a *two-tiered* setting (database and Engineering Server). This means that the Engineering Server accesses the database directly, which, of course, may be played on another host. In this setting distributed change notification is not available.

The Direct Engineering Server is useful for any type of application in which distributed event notification is not required. So, it represents the simplest variant to set up and use the Engineering Server (because it does not require a J2EE server such as JBoss).

Remote/Local Engineering Server In this *three-tiered* setting an additional intervening application server JBoss[1])is employed. The main advantage of these

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

forms of the Engineering Server is that clients can register themselves to be notified whenever other users make a change in the ontology. Thus the Engineering Server can serve as a basis for collaborative ontology development.

In the case of using the *Remote Engineering Server* that JBoss Application Server is accessed through another Java Virtual Machine (JVM) through remote interfaces. The Remote Engineering Server is useful for applications which need distributed event notification. This in particular relates to applications where ontologies are manipulated by several users concurrently, such as an ontology editor.

The *Local Engineering Server* represents a *three-tiered* solution as well. Here, however, the Engineering Server accesses the JBoss application Server from within the same JVM through local interfaces. The Local Engineering Server is useful in particular for web applications, since the web application and the server can be deployed in the same JVM, and thus increase performance (since the remote call overhead is eliminated).

Both, the Remote and Local Engineering Servers come in two versions: secure and non-secure. In the non-secure version no authentication of clients, that want to access the ontology/database, is needed. For the secure version, authentication is realized via JBoss. Note, that the version of the Engineering Server being used (i.e. secure or non-secure) is determined at deployment time.

4.4 Collaborative Ontology Engineering with the Engineering Server

As emphasized before, the main benefit of using the Engineering Server—apart from handling huge amounts of data via accessing a relational database system—is the possibility to collaboratively work on a single ontology model instance. That individual instance is maintained by the Engineering Server to which clients may connect.

Due to these multi-user capabilities of the Engineering Server it is feasible to develop an ontology in a distributed setting, e.g. with a group of ontology engineers or domain experts who are spread across several locations. Here, each participant connects to the instance of the Engineering Server (e.g. running on a server in Karlsruhe) with his/her local client (e.g. the OI-Modeler as part of the KAON Workbench). Then, it is possible to browse and explore the entire ontology without restrictions. Furthermore, if the respective user has the appropriate rights for writing, i.e. is also allowed to apply changes to the ontology, he/she may add, change, or delete ontology entities.

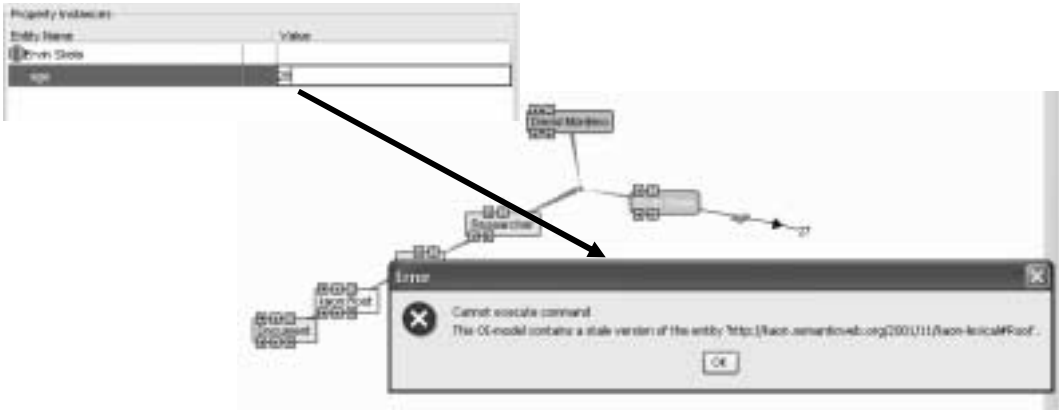

Figure 4.2: Ontology Version Conflict during Distributed Ontology Engineering

At its current stage of development the Engineering Server assigns a version number to each successive state of the ontology. In case a client's local "copy" of the ontology has a lower version number than the current ontology version maintained by the Engineering Server, the user is prompted that a conflict exists.

Imagine ontology engineer A has noticed that `Erwin Skela` was misspelled and corrects that typo so that that instance is called `Ervin Skela`. Moreover, ontology engineer B wants to change that instance's value for attribute AGE from 27 to 28. In case B has not updated his/her OI-Model, he/she will be notified about the conflict as depicted in Figure 4.2: Obviously, B is not allowed to change the value for attribute AGE as his/her current version of the ontology is obsolete. Now, ontology engineer B would have to refresh his/her ontology—which means that the Engineering Server's current ontology version is transferred to the client—and must reapply the changes he/she wanted to introduce, i.e. change the value for the AGE attribute as desired.

Chapter 5

TextToOnto

5.1 Overview

TextToOnto [MV01] is a tool suite built upon KAON in order to support the ontology engineering process by text mining techniques. Providing a collection of independent tools for both automatic and semi-automatic ontology extraction it assists the user in creating and extending OI-Models. Moreover, efficient support for ontology maintenance is given by modules for ontology pruning and comparison. In particular, the current distribution of TextToOnto comprises the following tools:

- **TaxoBuilder** for building concept hierarchies
- **TermExtraction** for adding concepts to an ontology
- **InstanceExtraction** for adding instances to an ontology
- **RelationExtraction** for semi-automatic learning of conceptual relations
- **RelationLearning** for automatic *and* semi-automatic relation learning
- **OntologyComparison** for comparing two ontologies
- **OntologyPruner** for adapting an ontology to a domain-specific corpus

Section 5.2 gives a detailled overview of the different tools by describing a sample workflow.

5.2 Tools

Before being able to start learning an OI-Model the user has to create a new document collection by selecting ”New Corpus” from the ”File” menu. The corpus management module provided by TextToOnto supports him in adding text, HTML or XML files to the corpus. In addition to the corpus an OI-Model can be selected, if the user wants to extend an already existing ontology.

5.2.1 TaxoBuilder

If a new ontology is to be created, a typical workflow usually starts by populating an OI-Model with concepts and instances. TaxoBuilder (see figure 5.1) automatically builds a concept hierarchy from the most frequent terms included in the corpus by inserting them into an empty OI-Model. Having started the tool via the menu items ”File”→”TaxoBuilder” the user is presented an interface, which allows him to specify a corpus and a newly created OI-Model as well as the number of words to be considered. Moreover the user can choose from two approaches to taxonomy construction: (i) The FCA-based approach described by [CST03] rests upon the assumption that a verb poses strong selectional restrictions on their arguments, so that a hierarchy of concepts can be derived from the inclusion relations between the extensions of the selectional restrictions of all the verbs, while the verbs themselves provide intensional descriptions for each concept. (ii) The second approach is based on a combination of Hearst-Patterns [Hea92], WordNet [Fel98] and various heuristics, which can be selected separately by the user.

5.2.2 TermExtraction

TermExtraction can be used to create new concepts from possibly relevant terms included in the corpus. Available parameters are the language (English, German or Italian) as well as the maximum number of words per term and the minimum frequency of occurrence necessary for considering a term as relevant. Moreover a linguistic filter can be defined by means of a regular expression over the language of Part-of-Speech tags. After the process of term extraction is completed a list of possibly relevant terms is displayed, which can be sorted according to measures like TFIDF, entropy or absolute frequency, in order to support the user in selecting terms to be added as concepts to the ontology.

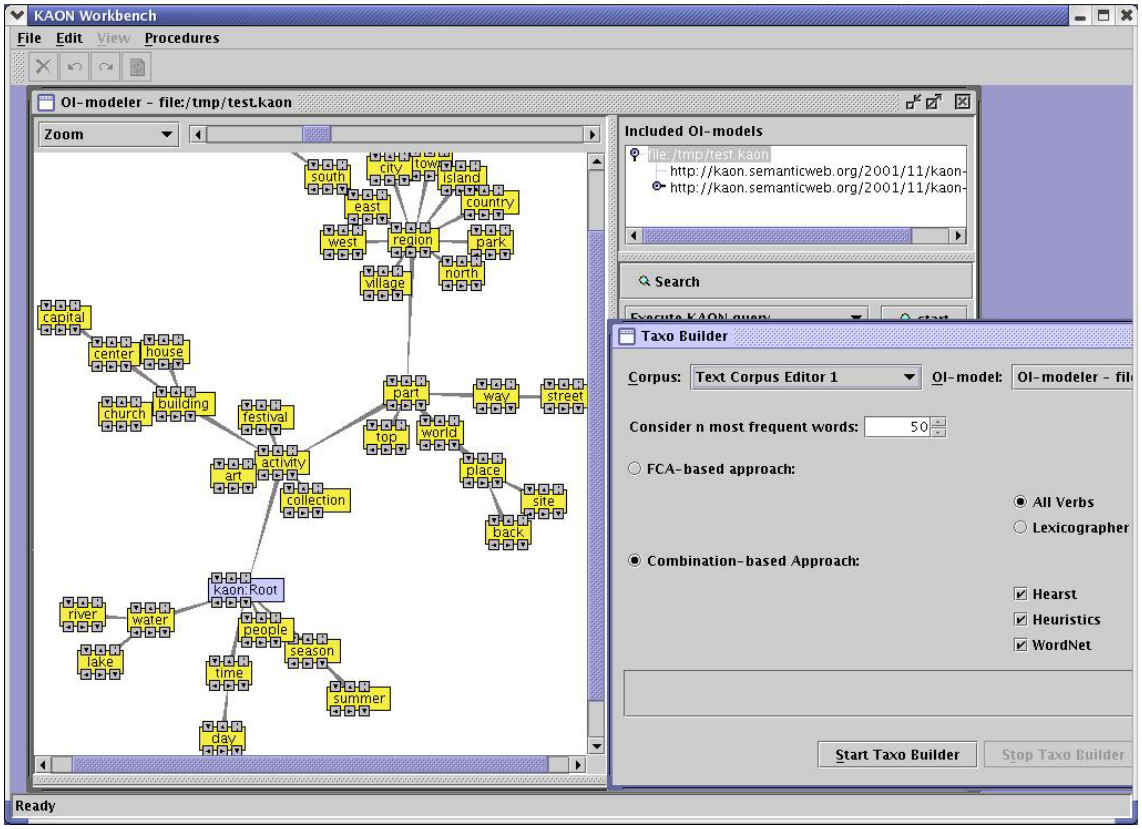

Figure 5.1: TaxoBuilder

5.2.3 InstanceExtraction

As soon as one or more concepts have been added to the OI-Model instances of these concepts are to be extracted from the corpus. InstanceExtraction (see figure 5.2) supports both semi-automatic and fully automatic learning of instances by applying a combination of various patterns from [Hea92] and [HS98], which can be selected separately by the user. If InstanceExtraction is performed semi-automatically each candidate term will be presented to the user in order to ask him for approval, whereas in the automatic mode newly discovered instances will be immediately added to the OI-Model.

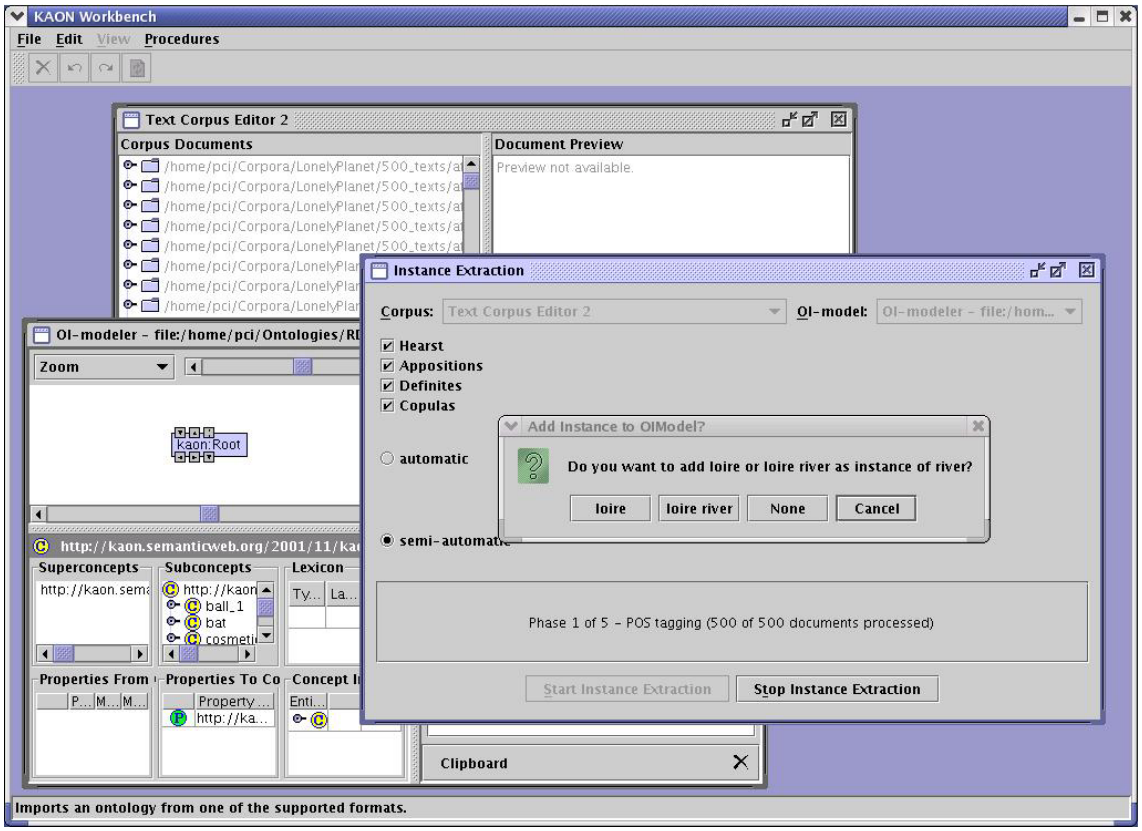

Figure 5.2: InstanceExtraction

5.2.4 RelationExtraction

RelationExtraction (see figure 5.3) is one of two tools provided by TextToOnto to extend the OI-Model by adding conceptual (and taxonomic) relations between concepts and instances already learned. Unlike the other tool, RelationLearning, which is briefly described by the following section, RelationExtraction only supports semi-automatic learning. Two approaches can be chosen in order to extract a list of candidate relations from the text. Whereas the first approach is based on association rules [MS00], the second one applies a set of text patterns very similar to those defined by Hearst [Hea92]. The user being presented this list can select single or multiple candidates and add them to the OI-Model - either as a property or a taxonomic relation. Available parameters for RelationExtraction - in addition to OI-Model, corpus and language - are the minimum confidence and the minimum support for relations to be displayed.

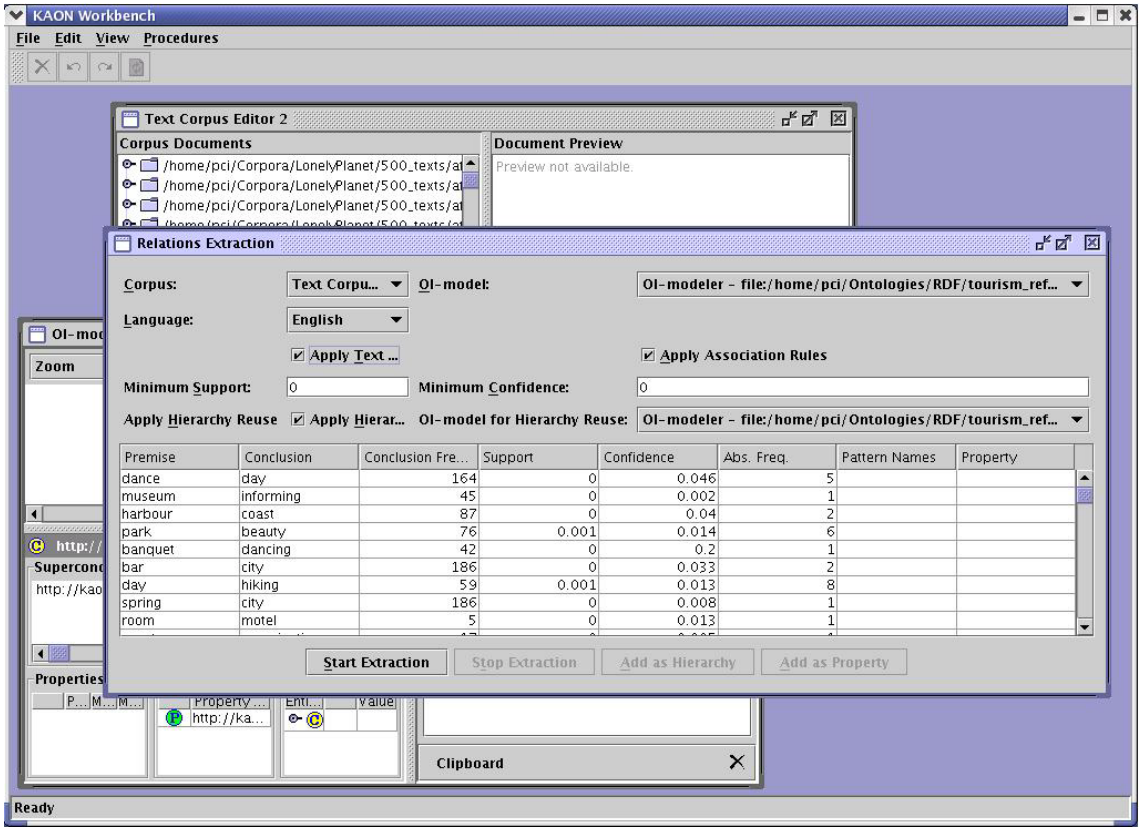

Figure 5.3: RelationExtraction

5.2.5 RelationLearning

RelationLearning, in contrast to RelationExtraction, supports both automatic and semi-automatic learning of conceptual relations. Moreover, if it is done semi-automatically, a name as well as a domain and a range for each relation are suggested to the user. Basically, the approach being applied by RelationLearning employs shallow text parsing in order to extract subcategorization frames, which can be restricted by using the information about selectional preferences [Res97], that is typical co-occurrences of predicates and conceptual classes, derived from the ontology.

5.2.6 OntologyComparison

In order to evaluate an OI-Model, which has been learned automatically or semi-automatically from a text corpus, it has to be compared with other - either learned or manually constructed - ontologies. OntologyComparison (see figure 5.4) is a tool provided by TextToOnto for comparing two OI-Models with respect to lexical and conceptual aspects [MS02] like taxonomic or relational overlap, for instance, which can be chosen separately by the user.

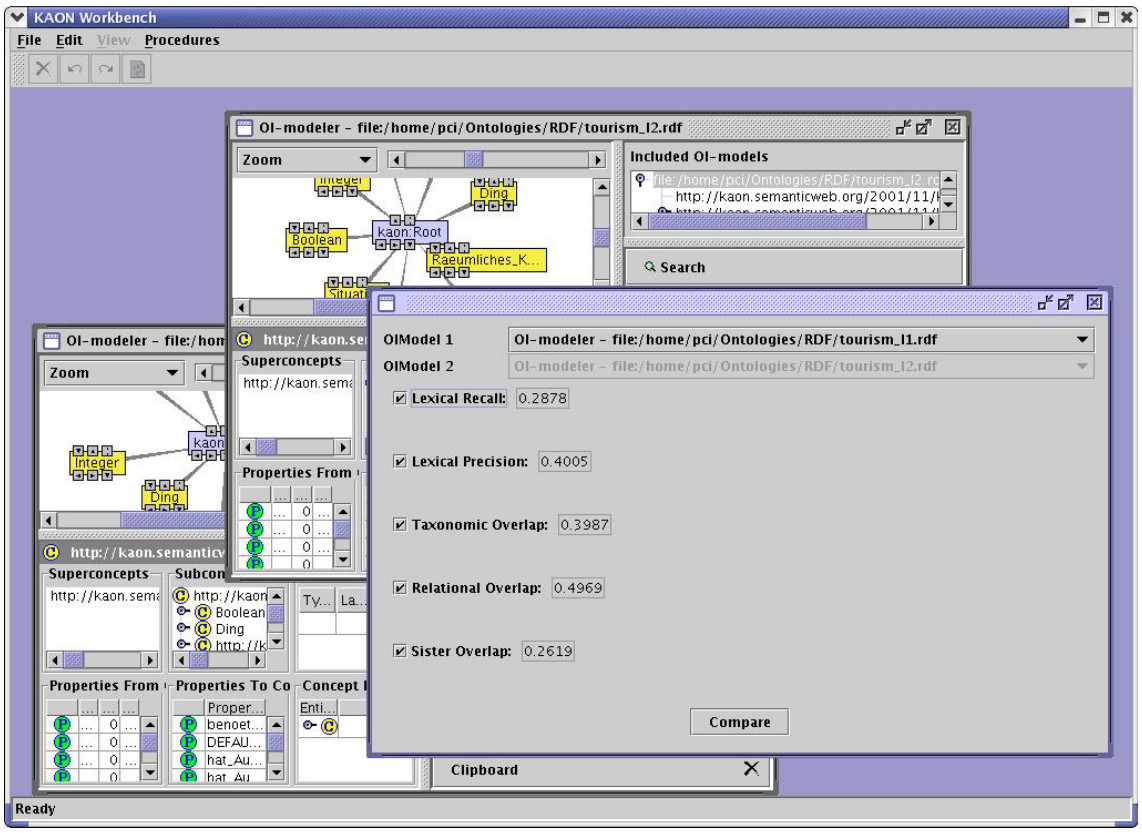

Figure 5.4: OntoComparison

5.2.7 OntologyPruner

Having extracted an OI-Model from a rather general corpus, the user might want to adapt it to the requirements of a more domain-specific corpus instead of learning a new ontology from the scratch. Therefore OntologyPruner supports ontology pruning by suggesting concepts to be removed on the basis of their frequency

within a given corpus. Parameters to be specified by the user are the language and the cumulative frequency threshold beyond which terms are considered as relevant.

Chapter 6

Outlook

Finally we will briefly indicate two relevant Open Source development projects: a back-end inference engine for OWL and a front-end ontology engineering environment.

- Firstly, in the Integrated Project DIP[1] our accompanying group at the FZI – Research Center for Information Technologies, Karlsruhe, will develop a hybrid reasoning framework – currently internally being called 'KAON-DL'. The aim is to provide efficient reasoning support for OWL (including A-Box reasoning). KAON-DL will act as a back-end server. It is planned that it will provide support for the WonderWeb OWL API[2] as well as for the KAON API.

- Secondly, in a joint effort (currently driven by Empolis, Ontoprise and UKARL) we will develop an Open Source ontology management framework based on the well-known Open Source universal tool platform **eclipse** (see `http://www.eclipse.org`. The aim is to provide a full-fledged front-end modelling environment which can be easily extended by third-parties such as SEKT partners. Essential part of such an environment will be the evolution features provided already in KAON (and OI-Modeler) today – and numerous further 'must have' features to be developed!

[1] see `http://dip.semanticweb.org`

[2] see `http://sourceforge.net/projects/owlapi` for source code and `http://owl.man.ac.uk/api.shtml` for further information

Appendix A

Download & Installation

This chapter gives hints concerning download and installation of tools and components related to KAON.

After a short overview on current versions and download sites we devote a single subsection to the installation of each tool presented in this document.

A.1 Download Overview

The following table A.1 summarizes download sources for and version numbers of the tools described within this document. Note, that future versions might not necessarily be compatible with the current versions described here. Be aware, that KAON needs at least Java 1.4.0.

Tool	**Version**	**Download Site**
KAON	V1.2.7	http://sourceforge.net/projects/kaon
Note: You may choose between the source code and a binary version.		
KAON Extensions *Includes:* KAONToEdit	V1.0	http://sourceforge.net/projects/kaon-ext/
TextToOnto	V0.95b	http://sourceforge.net/projects/texttoonto/
Java	V1.4.2	http://java.sun.com/j2se/1.4.2/download.html
OntoEdit	V2.6	http://www.ontoprise.de/customercenter/software_downloads/
JBoss	V3.2.1	http://www.jboss.org

Table A.1: Downloading KAON

A.2 Installation of KAON, its Workbench, and OI-Modeler

After having obtained KAON from the download site mentioned in Table A.1, you may proceed making that software applicable.

Note, that the term "KAON Workbench" (cf. Section 1.2) in particular comprises the OI-Modeler, KAON's ontology editor.

Installation: Installing KAON and its Workbench is straightforward. To install KAON just unpack the downloaded archive to some directory without spaces.

Using OI-Modeler: Set the KAON_ROOT environment variable to point to the root of your KAON distribution, for example via `c:\>set KAON_ROOT=c:\kaon`. Then, you may start the OI-Modeler by invoking `KAON_ROOT\bin\kaongui.bat`

As shown in Figure 1.1 the KAON archive you have downloaded comprises KAON's Engineering Server as well. However, since installing and using that software involves a number of steps to be followed, we describe that proceeding in very detail in the following section.

A.3 Installation of the Engineering Server

The Engineering Server represents KAON's implementation for large-scale and distributed ontology engineering. Note, that the Engineering Server is a part of KAON. So, the files and libraries related to it are included in the KAON archive you probably have downloaded and installed in Section A.2. The following steps describe how to install and use the Engineering Server.

Exemplarily, we describe the installation process for Microsoft SQL Server 2000 as the database system the Engineering Server collaborates with, for which it has been tested thoroughly. However, it should work in principle with all SQL2-compatible databases. It has been successfully run on IBM DB2, PostgreSQL and Oracle 8i/9i.

A.3.1 Direct Engineering Server

Applying the Direct Engineering Server corresponds to a two-tiered setting. Of course, the functionality of both tiers (i.e. client and relational database system) may be placed on the same machine. Anyway, the first phase of the Engineering Server's installation involves the creation of the server's database and filling it with the necessary database schema (see Section 4.1).

1. Install the relational database system and make sure it is up and running.
2. Create a new database (e.g. `myOntologyDatabase`) with your database management tool (in case of MS SQL Server that tool is the "SQL Server Enterprise Manager").
3. Use your database tool to execute the `schema.sql` script located in `KAON_ROOT\engineeringserver\schema\`. For example, for the MS SQL Server you may use the "SQL Query Analyzer" for that purpose. That script will create the schema in your database. The file is in the SQL2 format and uses the semicolon character as the command separator. Depending on your database, you may need to replace that character with the keyword used by your database (for example, *older* versions of MS SQL Server used the keyword GO as the command separator).
4. Depending on your database (i.e. if you are not using MS SQL Server), execute the supplementary schema script. For example, in case of Oracle database, execute `engineeringserver\schema\schema_oracle.sql` file. The same comments about the command separator apply.
5. Create a database user with your database management tool, e.g. MS SQL Server Enterprise Manager. Pay also attention that the security settings of MS SQL Server Enterprise Manager do not only allow for Windows authentication, but for "SQL Server and Windows".

Now, the Engineering Server is ready to be used in its *direct* version. From within the OI-Modeler you may create a new or open an existing ontology model via the Direct Engineering Server by supplying the following information:

- Start the OI-Modeler and choose open or create an OI-Model.
- Choose the tab "Direct Engineering Server".
- User Name and Password as specified in 5.

- Host Name: localhost or the server the SQL Server is running on
- Driver: Here, you can choose between MS SQL Server, IBM DB2, Oracle, PostgreSQL, and other.
- Port: 1433
- Database: name of your database, e.g. `myOntologyDatabase`

A.3.2 Remote/Local Engineering Server

For the *remote or local* version of the Engineering Server, you need the JBoss application server and you have to deploy the Engineering Server's EJBs to that application server. To do so, follow these steps:

1. Download JBoss from `http://www.jboss.org/downloads.jsp`. Unpack the JBoss into a directory without spaces.
2. Set the `JBOSS_HOME` environment variable to point to the root of the JBoss distribution (e.g. `C:\java\jboss-3.2.1_tomcat-4.1.24`).
3. KAON distribution contains JBoss 3.2.1 libraries (in the 3rdparty directory) that facilitate connection to the JBoss application server. The version of the client-side libraries must match to the JBoss version. If you are using some other version of JBoss, then you should exchange the JBoss libraries in the 3rdparty directory with the appropriate libraries from your JBoss distribution.

   Please note, that we conducted all our tests with version 3.2.1 of JBoss, thus we recommend using that JBoss version as incompatibilities might arise otherwise.
4. Customize JBoss to connect to your database. This involves the following steps:
   - Copy the template database configuration file for your database (for MS SQL Server that file is called `mssql-ds.xml`) from `JBOSS_HOME\docs\examples\jca` directory to `JBOSS_HOME\server\default\deploy`.
   - Open that file in a text editor.
   - Customize the name of the JNDI data source to KAON (by editing the value of the `<jndi-name>` element).

- Enter other information about your database (such as connection string, user name and password).
- Make the JDBC driver available to JBoss by copying it to `JBOSS_HOME\server\default\lib` directory. The JDBC driver for MS SQL, for example, consists of three files (`msbase.jar`, `mssqlserver.jar`, and `msutil.jar`).

5. Start JBoss (by invoking the `JBOSS_HOME\bin\run.bat` script).

6. Deploy the Engineering Server. Here, you will have to decide whether you intend to use the Engineering Server in its *secure* or *non-secure* version (see below). For the non-secure version invoke the `engineeringserver\deploy.bat` script (which will copy `engineeringserver-beans.jar` file to the `JBOSS_HOME\server\default\deploy` directory). For the secure version invoke the `deploy_secure.bat` script (which will copy `engineeringserversecure-beans.jar` file to `JBOSS_HOME\server\default\deploy` directory).

Now, the Local/Remote Engineering Server is ready for use. From within the ontology editor OI-Modeler you may now access it. Depending on the decision whether you intend to use the Engineering Server in its non-secure or secure version, proceed as follows:

Non-Secure Version In the non-secure version no authentication mechanism is present.

- Start the OI-Modeler and choose open or create an OI-Model.
- Choose the tab “Engineering Server”.
- Host Name: localhost or the server on which JBoss is running
- Port: 1099
- User Name and Password: You can leave these fields blank as no authentication methods are employed in the non-secure version.

Secure Version In the secure version the same information as in the non-secure version have to be supplied. Moreover, you have to fill in your user name and password allowing you to access JBoss. Of course, you have to customize JBoss in prior so that it supports authentication. This basically means, you have to define a set of users and grant them rights to read and/or modify the ontology.

For this you must set up a security domain called `kaon` and specify how authentication is performed. A simple way of authenticating users is by using `UsersRolesLoginModule` of JBoss. This module expects two files in `JBOSS_HOME\server\default\conf` directory: `users.properties` contains entries of the form `userName=password`, whereas `roles.properties` contains entries of the form `userName=role1,role2`.

The Engineering Server supports two roles: `KAONReader` role allows the user to read the OI-Model, whereas `KAONWriter` role allows user to change the OI-Model. The `UsersRolesLoginModule` can be started by editing `JBOSS_HOME\server\default\conf\login-config.xml` file and appending the following fragment:

```
<application-policy name="kaon">
 <authentication>
  <login-module
   code=
   "org.jboss.security.auth.spi.UsersRolesLoginModule"
   flag="required" />
 </authentication>
</application-policy>
```

For installation of other authentications modules, please refer to JBoss documentation.

A.4 Installation of TextToOnto

TextToOnto has been developed as an open-source project and therefore can be freely obtained from the address mentioned in table A.1. Because it does not need any additional libraries or software apart from the Java Runtime Environment[1], the installing and running TextToOnto is straightforward:

- decompress the binary distribution into a directory <`INST-DIR`> (for example `c:\TextToOnto`)
- go to <`INST-DIR`>`\bin`

[1] http://java.sun.com/j2se/desktopjava/jre/index.jsp

- set the classpath and start TextToOnto:

  DOS / Windows: invoke <`INST-DIR`>\`bin`\`texttoonto.bat`

  Unix / Linux:

  - execute the shell script `../sentenv.sh` in order to set up your environment
  - start TextToOnto via `java -cp "%TEXTTOONTO_CLASSPATH%" edu.unika.aifb.texttoonto.TextToOnto`

Bibliography

[CST03] P. Cimiano, S.Staab, and J. Tane. Automatic acquisition of taxonomies from text: Fca meets nlp. In *Proceedings of the PKDD/ECML'03 International Workshop on Adaptive Text Extraction and Mining*, 2003.

[Fel98] C. Fellbaum. *WordNet, an electronic lexical database*. MIT Press, 1998.

[Hea92] M.A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th International Conference on Computational Linguistics*, 1992.

[HS98] U. Hahn and K. Schnattinger. Towards text knowledge engineering. In *AAAI'98/IAAI'98 Proceedings of the 15th National Conference on Artificial Intelligence and the 10th Conference on Innovative Applications of Artificial Intelligence*, 1998.

[Kar02] FZI Karlsruhe. OI-Modeler user's guide, 2002.

[KK04] FZI Karlsruhe and AIFB Karlsruhe. KAON the Karlsruhe ontology and semantic web framework — developer's guide for KAON 1.2.7, 2004.

[MMS+03] A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz. An infrastructure for searching, reusing and evolving distributed ontologies. In *Proceedings of the twelfth international conference on World Wide Web*, pages 439–448, Budapest, Hungary, 2003. ACM Press.

[MMV02] A. Maedche, B. Motik, and R. Volz. A conceptual modeling approach for semantics-driven enterprise applications. In Springer-Verlag, editor, *On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBA*, pages 1082 – 1099, October 30 - November 01 2002.

[MS00] A. Maedche and S. Staab. Discovering conceptual relations from text. In W. Horn, editor, *Proceedings of the 14th European Conference on Artificial Intellignece (ECAI'2000)*, 2000.

[MS02] A. Maedche and S. Staab. Measuring similarity between ontologies. In *Proceedings of the European Conference on Knowledge Acquisition and Management (EKAW)*. Springer, 2002.

[MSSV02] A. Maedche, L. Stojanovic, R. Studer, and R. Volz. Managing multiple ontologies and ontology evolution in ontologging. In *Proceedings of the IFIP 17th World Computer Congress – TC12 Stream on Intelligent Information Processing*, pages 51 – 63, Montreal, Canada, 2002. Kluwer.

[MV01] A. Maedche and R. Volz. The ontology extraction and maintenance framework text-to-onto. In *Proceedings of the ICDM'01 Workshop on Integrating Data Mining and Knowledge Management*, 2001.

[oG03] ontoprise GmbH. How to work with OntoEdit — user's guide for OntoEdit version 2.6. http://www.ontoprise.de/documents/tutorial_ontoedit.pdf, September 2003.

[Res97] P. Resnik. Selectional preference and sense disambiguation. In *Proceedings of the ACL SIGLEX Workshop on Tagging Text with Lexical Semantics: Why, What, and How?*, 1997.

[SMMS02] L. Stojanovic, A. Maedche, B. Motik, and N. Stojanovic. User-driven ontology evolution management. In *Proceedings of the 13th European Conference on Knowledge Engineering and Knowledge Management EKAW*, volume 2473 of *Lecture Notes in Computer Science*, pages 285 – 300, Siguenza, Spain, October 1-4 2002. Springer.

[SSH02] L. Stojanovic, N. Stojanovic, and S. Handschuh. Evolution of the metadata in the ontology-based knowledge management systems. In *German Workshop on Experience Management*, pages 65 – 77, 2002.

[Vol04] R. Volz. *Web Ontology Reasoning with Logic Databases*. PhD thesis, University of Karlsruhe (TH), 2004.