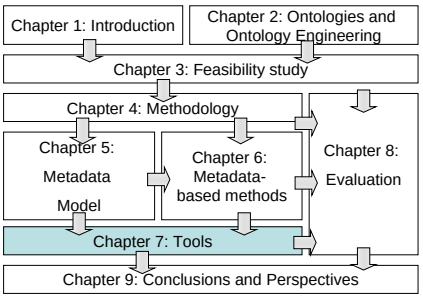
7 Tools Supporting Ontology Reuse



In this chapter we describe tools supporting the systematic and controlled operation of ontology reuse. At first we introduce PROMI, a platform for ontology reuse which prototypically realizes many of the theoretical considerations exposed so far in this thesis (Section 7.1). Further on, we focus on methods for metadata creation and management; OntoMeta, a service for automatic metadata acquisition, is introduced in Section 7.2. The tools are described in terms of their design, architecture, implementation and utilization. We close the chapter with an overview of its contents in Section 7.3.

7.1 PROMI

The project PROMI (**P**latform for the **R**euse of **O**ntologies through **M**erging and **I**ntegration) was initiated in February 2005 within the working group Network Information Systems at the Free University of Berlin with the aim to provide a comprehensive, easy-to-use and -adapt platform for reusing ontologies and ontology-like knowledge sources in Semantic Web applications.[1] The core project team consists of two researchers at the aforementioned institution (including the author of the thesis), as well as several undergraduate students. The work reported in this section was fundamentally conceived and supervised by the author, but it could not have been accomplished without the help of the remaining members of the project team.

The first release of the platform in August 2005 was restricted to the field of ontology merging and integration. Building upon our ontology reuse methodology [168, 170, 172] it implemented an incremental approach to merging and integration on the basis of elementary linguistics-oriented matching services. In a second phase, in December 2005, PROMI was revised towards a context-sensitive behavior w.r.t. the aforementioned tasks. We investigated how contextual dependencies, as those elaborated in Section 6.2 in the previous chapter, could be added to the system in order to aid the engineering team in taking the decision upon the appropriate merging/integration strategy. Finally, in April, 2006, PROMI was extended with an ontology evaluation component realizing the core ideas of the method introduced in Section 6.1. Further extensions are planned—beyond the scope of this thesis—in relation to the MOMA project [148, 150]. Since May 2006 the tool is available as an open source project at `http://sourceforge.net/projects/promi/`.

[1] `http://promi.ag-nbi.de` last visited in July, 2006

7.1.1 Design and Architecture

PROMI is intended to provide ontology engineers and domain experts with *methodological* and *technological* support in carrying out ontology reuse. Accounting for the results of our feasibility study, the platform was conceived as a *dedicated ontology engineering environment*, in which participants at the reuse process are given advice in how to perform the reuse process or particular phases of it (i.e. *methodology*), and which methods and tools can be applied to operationalize it effectively and efficiently (i.e. *technologies*).

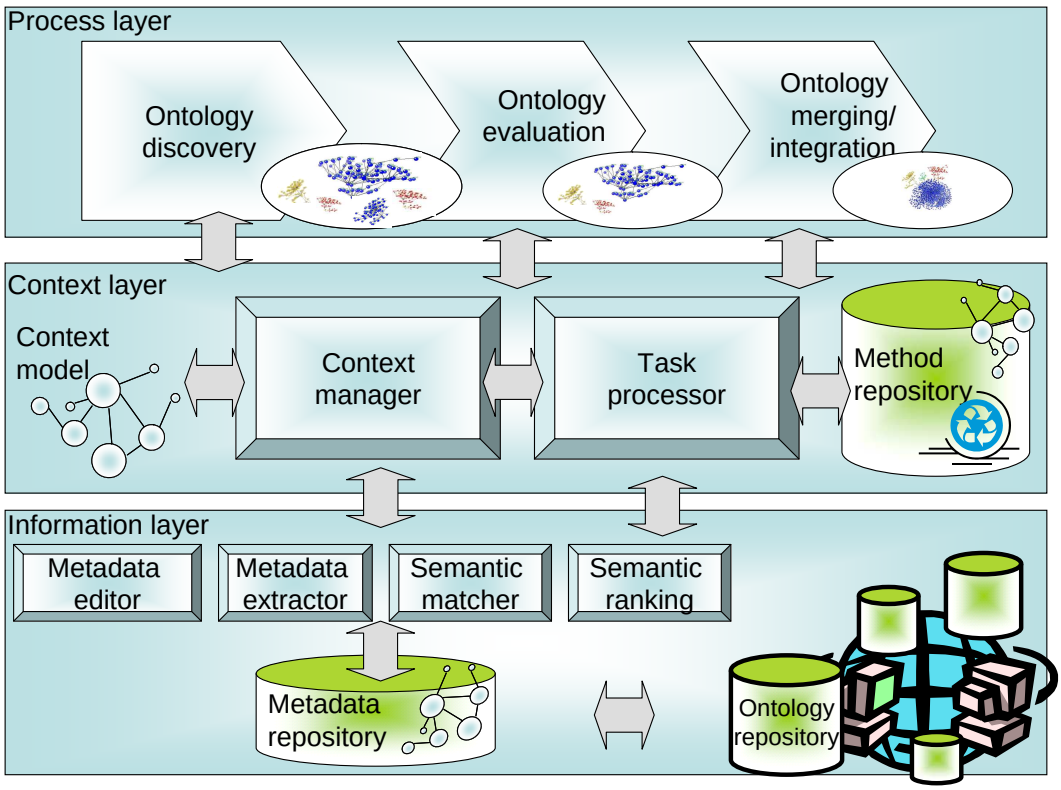


Figure 7.1: PROMI Layered Architecture

From a process-oriented perspective the platform builds upon the presented methodology, which describes in detail the reuse process and its main phases in terms of participants, activities they perform, and results achieved. Further on, the methodology recommends methods, techniques, heuristics and tools likely to be helpful to the engineering team during the operation of the process (cf. Chapter 4). Technologically this process-driven description is augmented by an inventory of self-developed methods and techniques, which can be used to systematically accomplish specific aspects of the reuse process in an automatized manner (cf. Chapters 5 and 6). Consequently, PROMI can be divided into three layers, which are depicted in Figure 7.1:

1. **Process layer**: on this layer the platform controls the execution of the overall reuse process, in that it guides the user with respect to the activities which are required for reusing existing ontologies, their order of execution and the results expected from each activity.

2. **Context layer**: this layer is responsible for the context-sensitive behavior of the platform. On the basis of a simple, yet powerful context model and rules for selecting situation-relevant information, it is concerned with the context-dependent execution of the ontology reuse steps. The layer is further decomposed in three modules:

   a) **Context manager** associates a particular reuse activity to the appropriate reuse metadata. The selection is realized in form of rules combining pre-defined context patterns (e.g., specific roles, application types) to particular metadata elements.

   b) **Task processor** guarantees the execution of the reuse process with or without taking into consideration contextual aspects. In the former case, the additional information acquired by the context manager induces a selective behavior of the system with respect to a particular reuse activity. In the latter the reuse process is performed in a context-independent way, without taking into account the specificity of the setting at hand.

   c) **Method repository** is a generic collection of methods applied to aid ontology engineers and domain experts in accomplishing certain tasks. The PROMI system currently embeds a prototypical implementation of the methods presented in Chapters 4 to 6. They were realized in collaboration with undergraduate students of the Free University of Berlin and members of the working group Networked Information Systems.

   The context layer has access to the

   a) **Context model** which formally defines the structure and the organization of contextual aspects within ontology reuse processes by means of a Semantic Web ontology, and to the

   b) **Contextual information** instantiating this model with concrete examples derived from the specific design of our solution space. This includes the design of the metadata model, of the ontology reuse process model and of the associated support methods. If new methods, or even a different process description, are conceived to be used in relation to the platform, these are required to be described in terms of the context model in order for PROMI to provide the desired behavior. Note that the context layer does not directly manage descriptive information about ontologies or ontology management services. This information is processed and stored in the layer below. In accordance to the way we understand the notion of context, the present layer solely defines which aspects of the information globally available to the platform are *context-relevant* and how they should be taken into account for specific purposes.

3. **Information layer**: the third layer is concerned with the acquisition and management of information describing the ontology reuse process, its main activities, as well as associated Web resources and services. It includes the ontology reuse metadata model and metadata instances acquired during ontology engineering empirical studies or using (semi-)automatic heuristics. Further on, the information layer incorporates information about the methods to be automatically executed during the operationalization of the reuse process (i.e. metadata about various matchers and merging techniques, translators etc.). From a functional point of view, it distinguishes among four modules:
   a) **Metadata extractor** comprises methods to enrich the metadata repository with automatically acquired metadata information.
   b) **Metadata editor** can be utilized to manually create and revise metadata entries.
   c) **Semantic matcher** provides a set of linguistic, taxonomical and heuristic matching algorithms employed to semantically compare metadata entries and, implicitly, the items the metadata describes.
   d) **Semantic ranking** computes relevance scores for particular items such as ontologies or ontology reuse methods on the basis of the results delivered by the semantic matcher.

   The information layer makes use of the information stored in a metadata repository and has access to ontologies described by the metadata:

   a) **Metadata repository** consists of metadata schemas (such as for Semantic Web resources, merging methods or matching algorithms) and instance data referring to existing ontologies, documents or services.
   b) **Ontologies** online available and described in terms of the pre-defined metadata schemas.

   As aforementioned the methods and techniques applied to achieve a specific sub-goal of a reuse endeavor (such as the merging of two ontologies) could be homogeneously accessed in terms of Web Services and described using WSDL or even semantics-aware service descriptions [120, 152, 200].

In the following we will give an overview of the three layers.

Process Layer

A detailed description of the reuse process is given in Chapter 4. PROMI commits to this methodology, guiding the user in carrying out the necessary tasks along a pre-defined workflow.

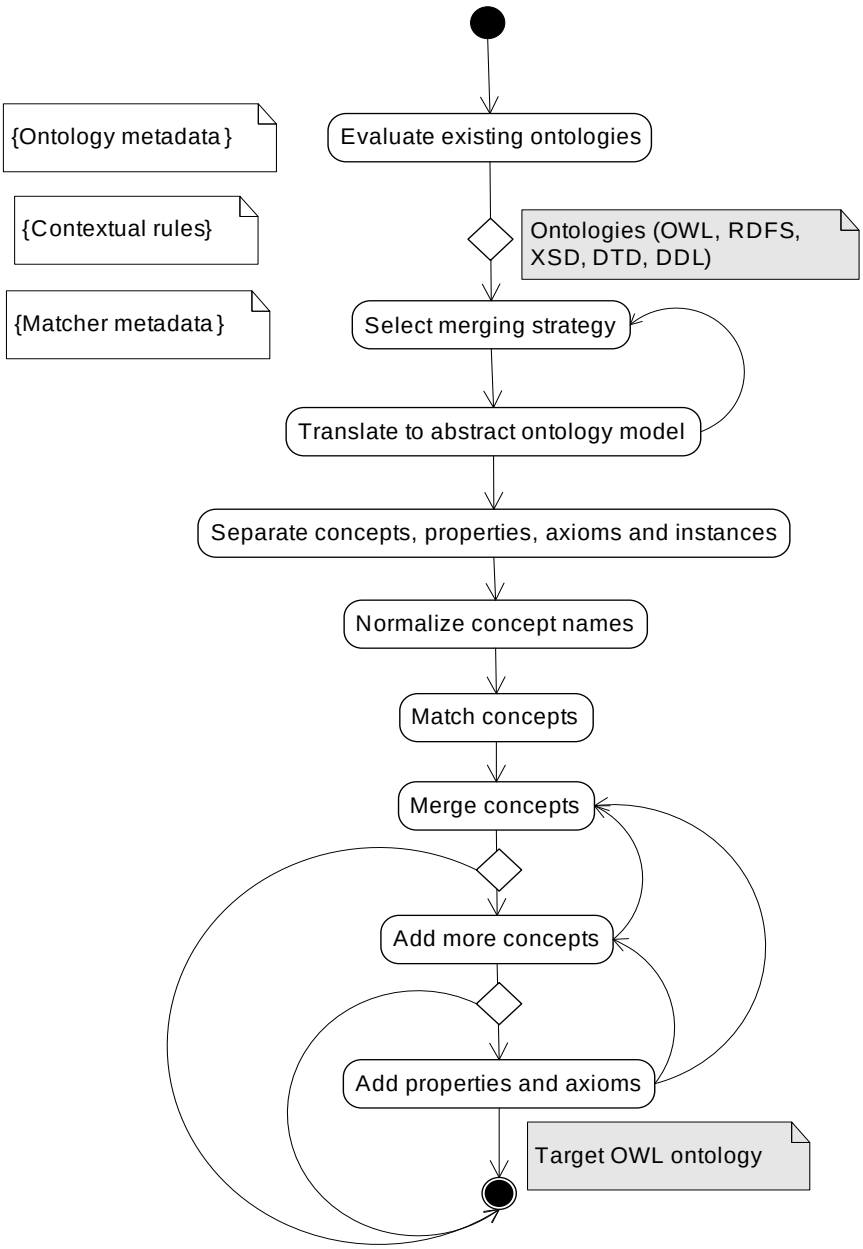


Figure 7.2: PROMI Ontology Reuse Process

Prior to starting any reuse-relevant activity the user is provided an overview of the supported process steps—as depicted in Figure 7.2. During the reuse process the tool keeps track of the tasks currently being performed and guides the engineering team in accomplishing their goals in an optimal order of execution. This behavior is realized at menu level, while its effects are consolidated through a working progress window, which permanently highlights the current state of the reuse process and the alternatives to proceed (cf. Figure 7.3, left). The tool is accompanied by a comprehensive help component, which provides an in-depth description of its methodological background and detailed guidance for a correct usage of the tool features. A screenshot of the PROMI help facility is depicted in Figure 7.3, right.

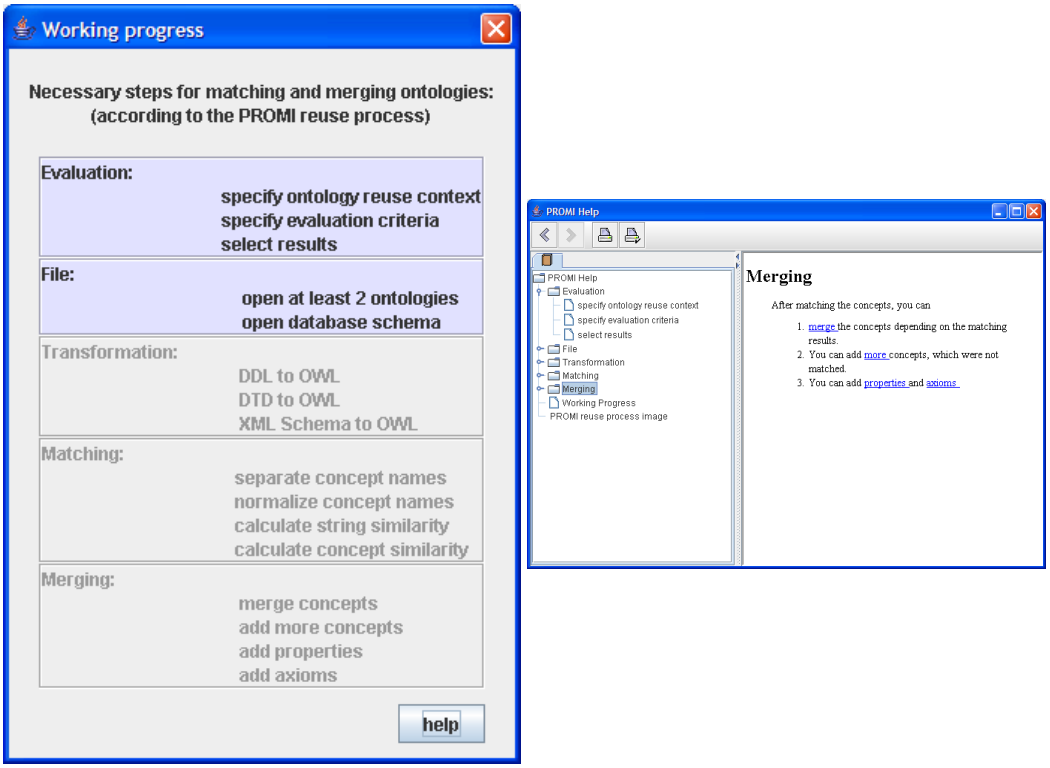


Figure 7.3: PROMI Process Support: Working Progress Window (left) and Help Window (right)

Context Layer

The context layer relies on a suite of methods operationalizing particular steps of the reuse process. The methods make use of structured metadata capturing reuse-relevant information about ontologies, as well as about the capabilities of the techniques and tools involved in specific reuse activities. As derived from the feasibility study, the choice upon a particular strategy and the results of applying it strongly depend on the context in which the reuse process is actually being performed: the method applicants, the reuse candidates, as well as the scope and purpose of the overall ontology engineering project. These dependencies are made explicit in the architecture of the platform in form of a *context model* and *contextual dependency rules* associated to it.

As already explained in Chapter 6 the context model differentiates among four information categories:

1. **User-related information**: in this category the model captures the participants at the reuse process. The user information is required in order to select and customize a particular method involved in a reuse task to the level of expertise of its applicant. Representatives for this category from an implementation point of view are ontology engineers and domain experts. Conceptually we consider in addition users and programmers.

2. **Task-related information**: this information relates to particular steps of the reuse process in order to select the appropriate methods. As introduced before, the context layer contains methods to aid the discovery, the evaluation and the integration of ontologies and the underlying activities.

3. **Environment-related information**: this type of information relates to the application scenario in which the final ontology, which will incorporate the reusable components at hand, is going to be used. An application scenario is characterized by a minimal set of information as follows:
   - **Ontology application system**: information about the system the final ontology will be used in (type of system, industrial sector etc.)
   - **Ontology task**: the task intended to be accomplished by the planned ontology
   - **Ontology role**: the role accomplished by this ontology in the context of this task.

4. **Target-related information**: information about the ontologies being analyzed during the reuse process. This information is stored and managed in the information layer.

This context model (Figure 7.4), which has been designed in [166] for the development of a case-based reasoning system in the medicine domain, is consistent to the broad majority of research investigations concerning the formal representation and the definition of context [167]. As aforementioned, the main objective of our work is to show how context issues affect the success of an ontology reuse endeavor, while the way contextual information is electronically represented is of secondary importance. Therefore the context ontology commits to an existing proposal in the field, which through its generality and flexibility fits well to the characteristics of our domain of research.

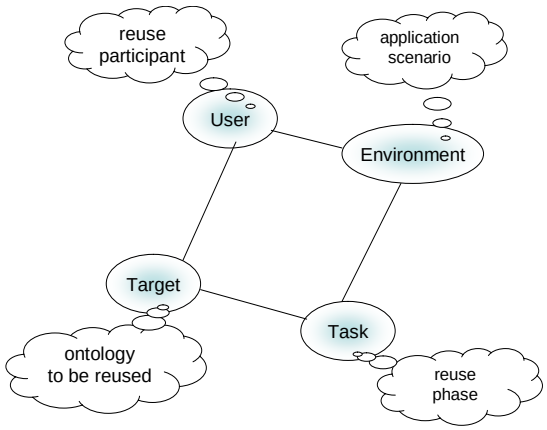


Figure 7.4: PROMI Context Model

Provided knowledge about the user involved in a particular task, the context manager analyzes the environment and the target information (available in form of metadata) in order to select method(s), technique(s) and algorithm(s) adequate to accomplish particular ontology reuse tasks. The task processor is responsible for their execution, the generation of the results and their presentation to the user. Consequently, the PROMI platform is able to provide methodological and technological support for a wide range of reuse participants, with or without any experience in ontological engineering. Further on, for the accomplishment of a particular goal, it takes into consideration solely methods, which are likely to handle the characteristics of the ontologies to be reused and the target application context successfully. A description of this selection mechanism for the tasks of ontology evaluation and merging is provided in Chapters 4 and 6.

Information Layer

The core of the information layer is the metadata model, which captures information about a particular Web resource, be that an ontology, an ontology fragment or an arbitrary Web document. This information is likely to enhance reusability, as indicated by the requirements analysis and confirmed by our expert-based evaluation (cf. Section 8). The metadata is represented in a formal and structured way in order to enable the context layer to automatically process it for specific purposes. Metadata and ontologies may be stored in dedicated or general-purpose persistent storage systems, which are *accessed* by the information layer, without being a direct part of it. In addition, the information layer includes descriptions of basic services supporting the reuse process, which are located in the context layer. This information can be expressed in terms of an ontology, similarly to the ontology metadata model—such as the one introduced in Section 6.2. However, it can also resort to other task-specific representations which strongly depend on the way the methods are realized at implementation level. In the context of the Semantic Web, we might expect these methods to be available in form of Web Services, whose functionality and characteristics are formalized

using (semantic) service-specific description formats [4, 120, 152, 200].[2]

The information layer comprises modules (or has access to services) for the generation and management of metadata entries and their matching. The metadata generator OntoMeta (cf. Section 7.2) applies model-based heuristics in order to automatically derive the values of specific metadata elements. In doing so, it fundamentally contributes to the usability of the entire reuse platform, since creating metadata manually is widely recognized as one of the key obstacles in the real-world dissemination of the Semantic Web. The matching component includes a basic set of algorithms which allow higher-level methods to compare metadata entries semantically, i.e. using ontology-driven similarity measures. In particular, the semantic matcher implements the functionality required for the realization of the ontology evaluation method (cf. Section 6.1), which primarily relies on heuristic and semantic similarities to assess the application usability of a certain ontology. The semantic ranking is another core component of the information layer. Its aim is to compute relevance scores for particular information items such as ontologies, but also matching, merging, integration or evaluation services—provided the semantic description of their capabilities—on the basis of the similarity values obtained from the semantic matcher.

7.1.2 User Guide

In order to illustrate the way the PROMI platform can be used to enhance ontology reuse processes, we provide a complete example of its capabilities in terms of the eHealth case study introduced in Section 3.3. In this scenario, a team of ontology engineers and domain experts were confronted with the task of building an medical ontology using existing standard classification systems in the field.

PROMI Ontology Discovery

In order to find potential reuse candidates the engineering team accesses an ontology repository (such as the Taxonomy Warehouse)[3]. This repository provides some search facility in terms of an implicit or explicit metadata model. The criteria which are likely to be useful during this step are introduced in the ontology reuse methodology. In this case the methodology recommends the usage of simple search queries containing the most important concepts which are expected to be covered by the ontology *"medicine"*, *"lung"*, *"pathology"*, *"anatomy"* and synonyms of the term "ontology". Note that our work addresses the question of how to find existing ontologies *only at process level*. PROMI does not provide fully-fledged ontology discovery support. However, in Chapter 5.3 we introduced applications such as Oyster and Onthology which already use our metadata model in order to operationalize this task. A collaboration in a similar direction with the operators of the SchemaWeb portal is currently being finalized.[4] Furthermore, the user can refer to the ontology repositories listed in Section 2.2.2.

[2] This issue is however out of the scope of this thesis.

[3] `http://www.taxonomywarehouse.com` last visited in September, 2005

[4] `http://www.schemaweb.info` last visited in May, 2006

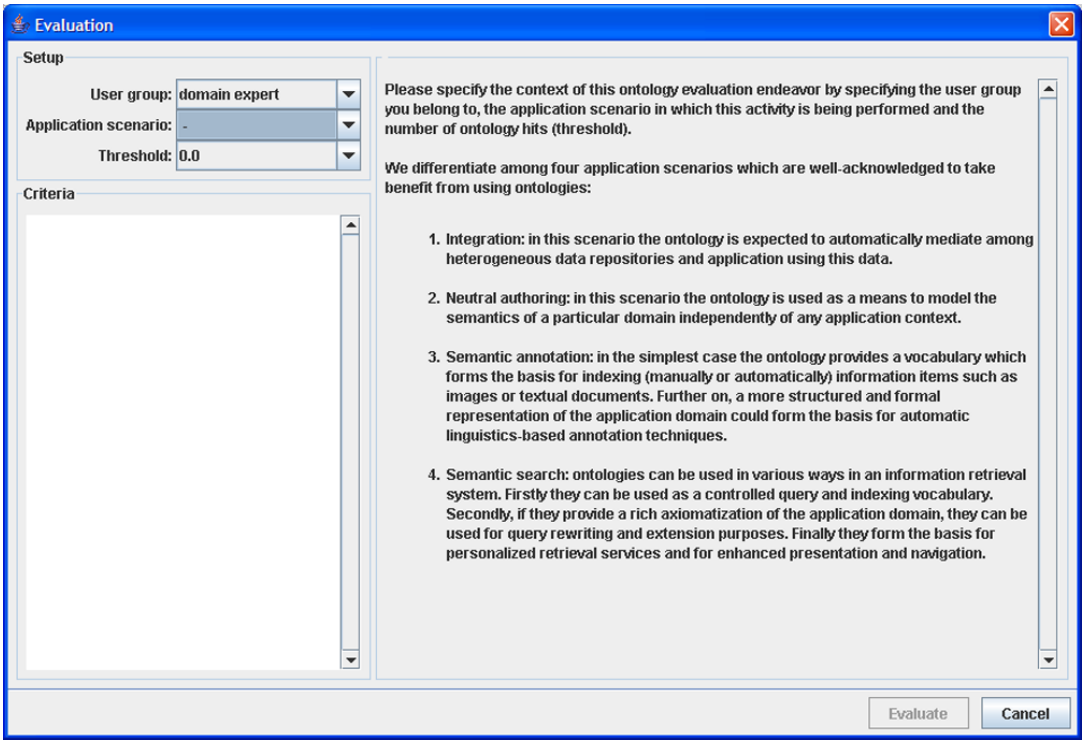


Figure 7.5: PROMI Evaluation Support: Context Specification

PROMI Ontology Evaluation

Once a list of reuse candidates has been established, the engineering team is guided towards the evaluation of these resources as described in the methodology. The system provides each of the process participants the task which is foreseen to be accomplished by him in the adequate terminology. The selection of these tasks and the associated support methods is performed automatically by triggering matching rules between task, method and ontology descriptions of the information layer. If no metadata is available for the selected sources, the metadata extractor can be used to acquire (parts of) this information automatically. This functionality is provided by the OntoMeta service, which is currently not integrated into the platform.

Figure 7.5 illustrates the first step of the ontology evaluation method, which requests a specification of the core contextual aspects of the current scenario. These are situated on the left side of the window on the top. The right half of the window documents the application scenarios considered.

Figure 7.6 shows the interfaces generated for domain experts (top) and ontology engineers (bottom) depending on the context specified by the user in the first step.

For each of the evaluation-relevant ontology criteria the user is requested to choose among particular values (referring to metadata entries) or to provide free text input. The features are further weighted using a five point scale. Some of these are depicted in Figure 7.7. Once the evaluation criteria are selected, the platform detects ontologies which are likely to satisfy the

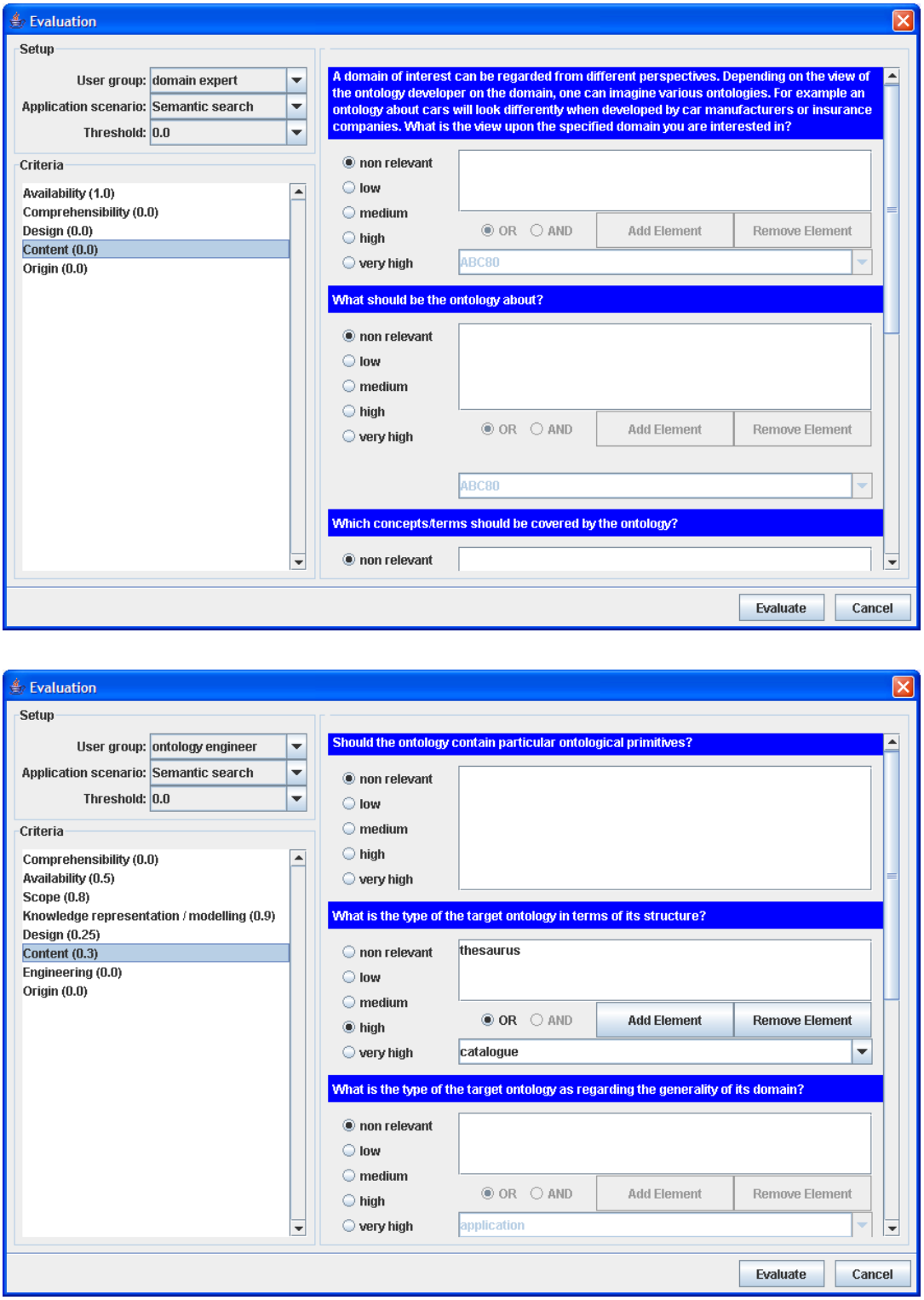


Figure 7.6: PROMI Evaluation Support: Context-sensitive Method Presentation for Domain Experts (top) and Ontology Engineers (bottom)

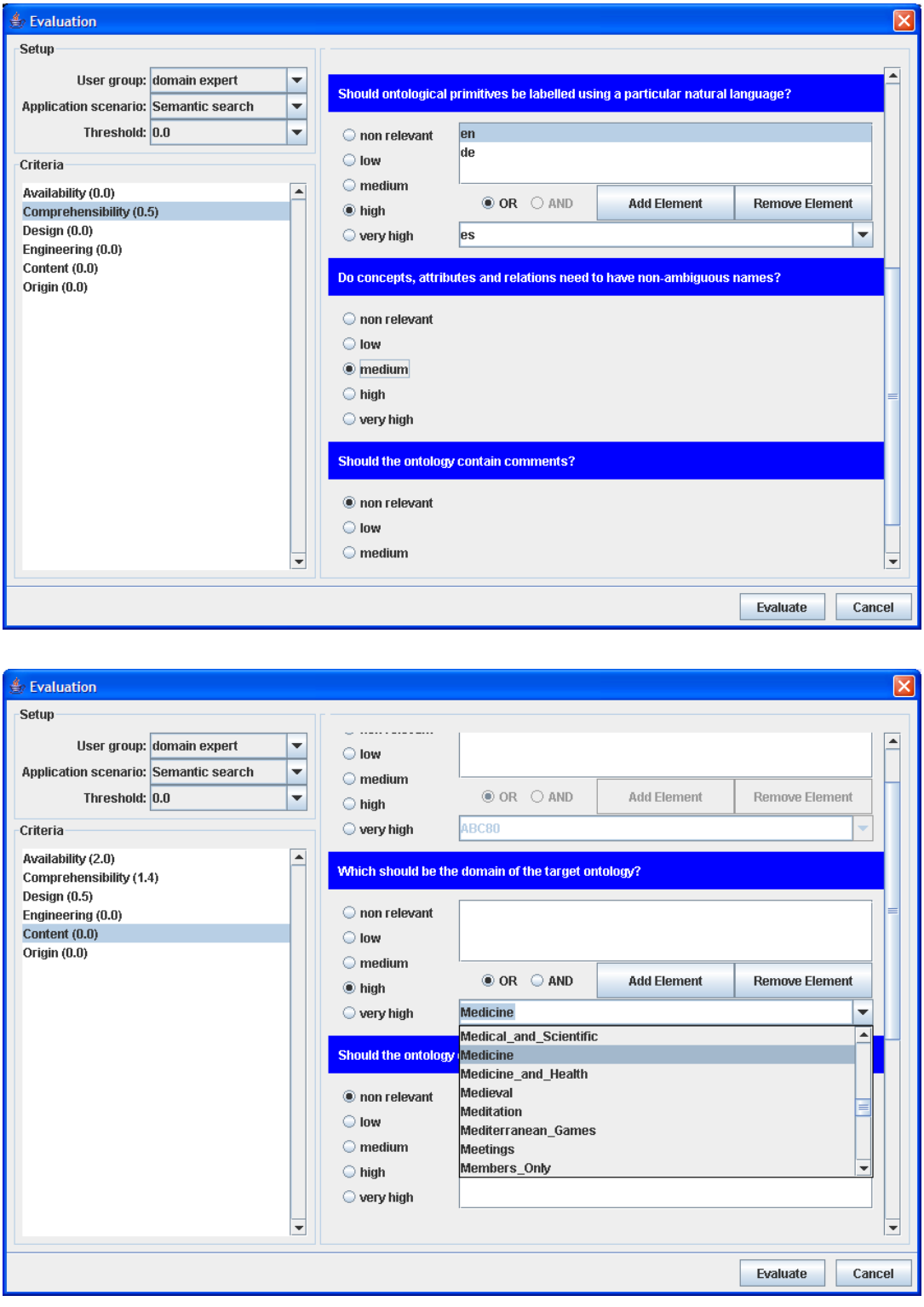


Figure 7.7: PROMI Evaluation Support: Feature Selection

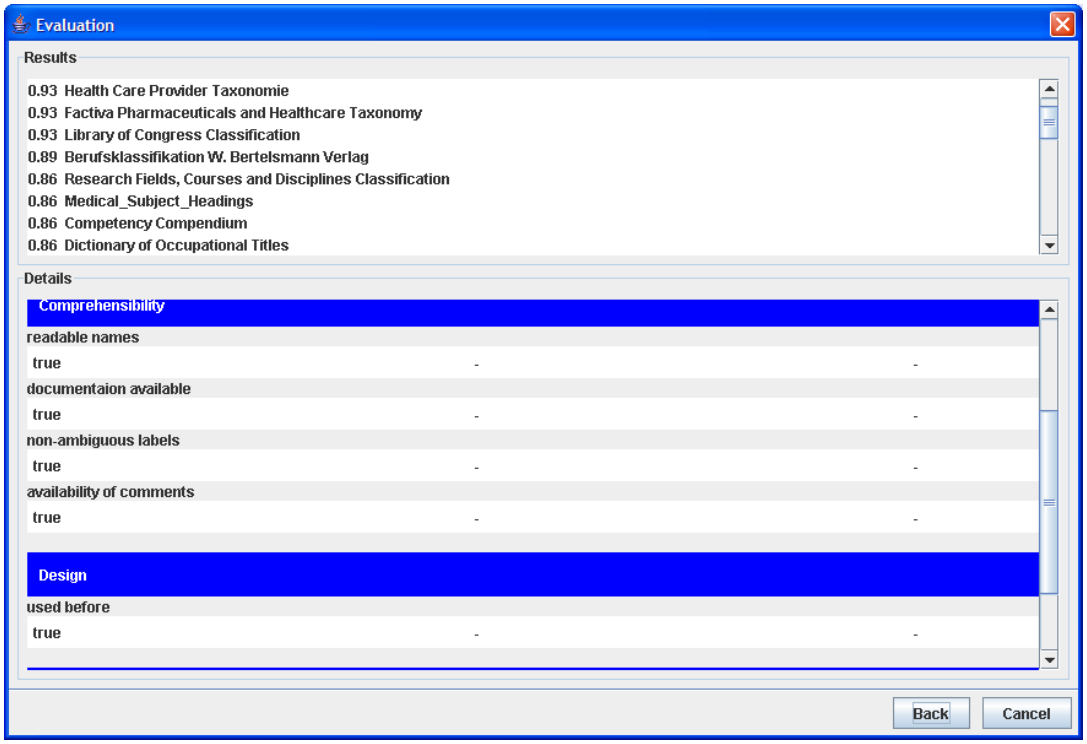


Figure 7.8: PROMI Evaluation Support: Usability Computation and Result Selection

user specification. Figure 7.8 shows the results computed by the PROMI tool in this case and the corresponding metadata. The heuristics behind this computation was elaborated in the previous chapter.

PROMI Ontology Merging and Integration

In the third step the selected ontologies need to be customized, which means in this case translated to a new representation language, and, as they cover overlapping domains, merged to a single ontology. In order to perform the first task, the context manager acquires information about the translation tools and match it against the ontology representation language in order to detect the most appropriate method/tool to execute the translation.

Once the ontologies have been translated to the common format (cf. Figure 7.9), we need methods to merge them to a final ontology. Depending on the properties of the two ontologies, in this case two taxonomies containing concepts in the same natural language, properties, axioms, and no instance data, the tool decides to suggest a merging method based in hierarchical structures, string distances or constraints to generate the final ontology. The applicable rules are displayed in addition to the window in which the user is requested to specify the matching algorithm he will use to compare concept labels and their compounding terms (cf. Figure 7.10).

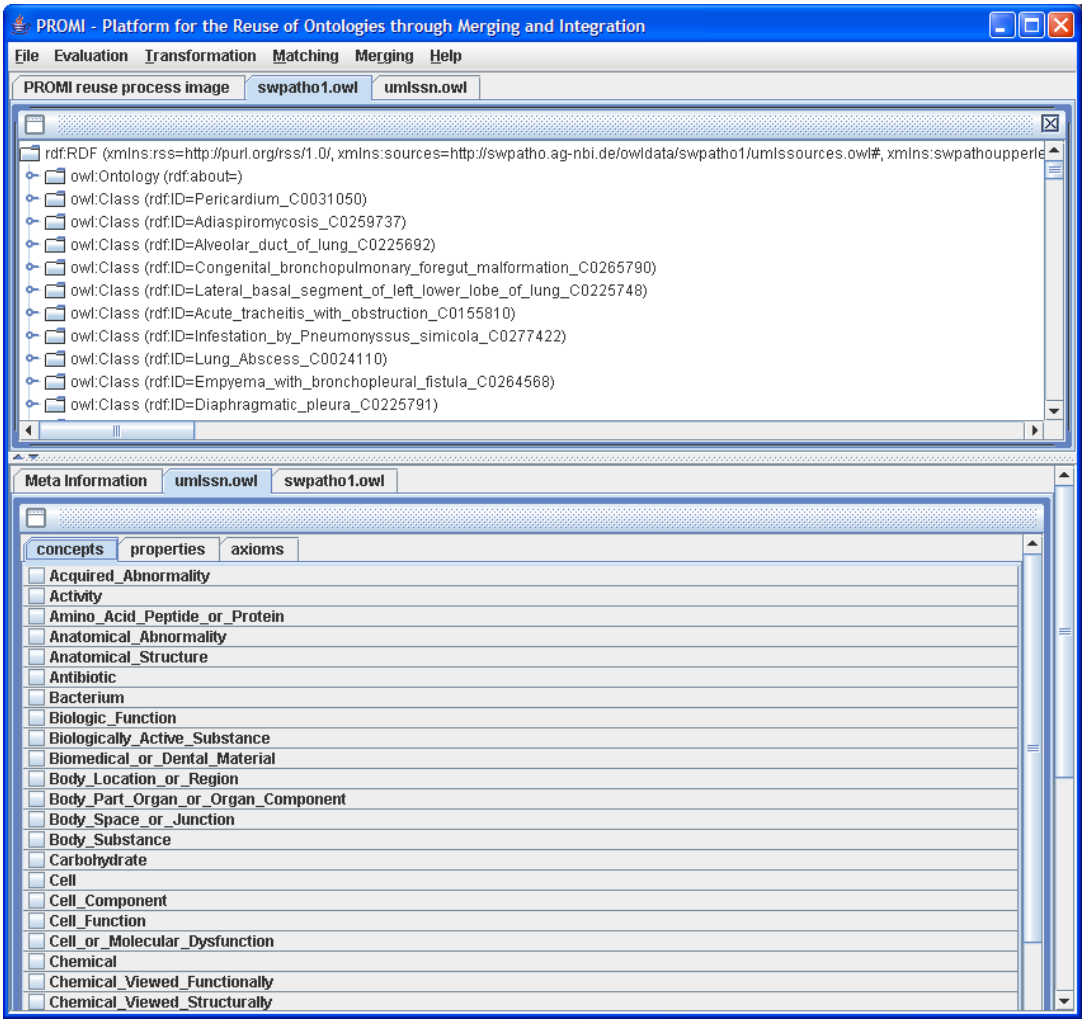


Figure 7.9: PROMI Translation Support

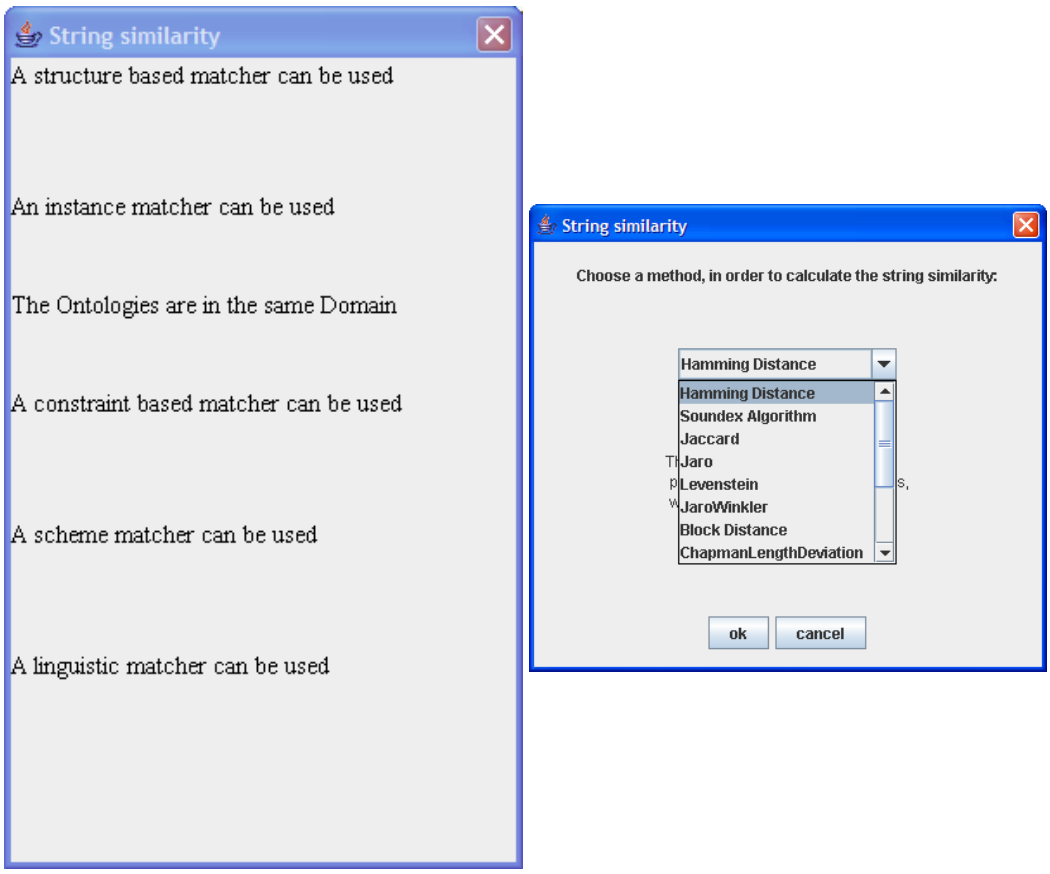
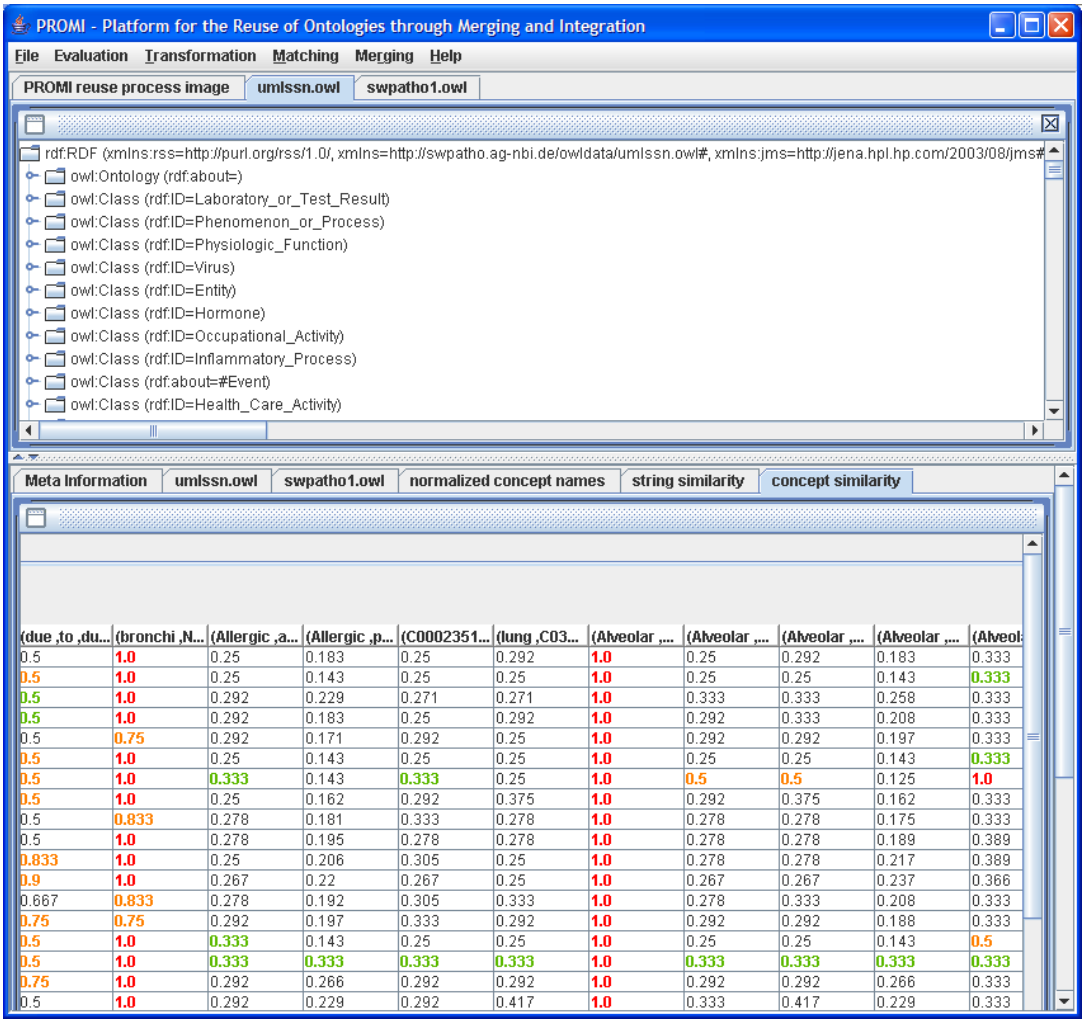


Figure 7.10: PROMI Merging Support: Context-based Selection of the Matching Algorithms

The current PROMI implementation includes and actively utilizes a series of linguistic similarity measures, which are applied incrementally depending on the complexity of the input ontologies. Graph-based matchers are implemented as well, but their integration in the application logic and the associated user interaction are subject of future development. In the merging/integration first concepts/classes are compared terminologically. Figure 7.11 shows the results of this operation for two ontologies modelling core medical terms. Due to the fact that properties and axioms can not be subject of similar computations in a meaningful way—the labeling rules do not apply with the same rigor for them in comparison to classes and instances—the merging of these additional primitives is decided manually by humans (cf. Figure 7.12). The choice upon the appropriate matching algorithm can be additionally supported with the help of a diagram displaying the overall distribution of the similarities among concepts (cf. Figure 7.13). The utilization of graph-based measures will imply a slight modification of the merging and integration workflow. The process could still be carried out in an incremental fashion: in a first phase one can compare the involved ontologies using a taxonomical matcher, whose results are complemented manually by the user inputs concerning axioms. A fully-fledged solution for the matcher selection and execution is not subject of this thesis. In the context of the PROMI framework this issue is approached in more depth in

[148].



(due ,to ,du...	(bronchi ,N...	(Allergic ,a...	(Allergic ,p...	(C0002351...	(lung ,C03...	(Alveolar ,...	(Alveolar ,...	(Alveolar ,...	(Alveolar ,...	(Alveol
0.5	1.0	0.25	0.183	0.25	0.292	1.0	0.25	0.292	0.183	0.333
0.5	1.0	0.25	0.143	0.25	0.25	1.0	0.25	0.25	0.143	0.333
0.5	1.0	0.292	0.229	0.271	0.271	1.0	0.333	0.333	0.258	0.333
0.5	1.0	0.292	0.183	0.25	0.292	1.0	0.292	0.333	0.208	0.333
0.5	0.75	0.292	0.171	0.292	0.25	1.0	0.292	0.292	0.197	0.333
0.5	1.0	0.25	0.143	0.25	0.25	1.0	0.25	0.25	0.143	0.333
0.5	1.0	0.333	0.143	0.333	0.25	1.0	0.5	0.5	0.125	1.0
0.5	1.0	0.25	0.162	0.292	0.375	1.0	0.292	0.375	0.162	0.333
0.5	0.833	0.278	0.181	0.333	0.278	1.0	0.278	0.278	0.175	0.333
0.5	1.0	0.278	0.195	0.278	0.278	1.0	0.278	0.278	0.189	0.389
0.833	1.0	0.25	0.206	0.305	0.25	1.0	0.278	0.278	0.217	0.389
0.9	1.0	0.267	0.22	0.267	0.25	1.0	0.267	0.267	0.237	0.366
0.667	0.833	0.278	0.192	0.305	0.333	1.0	0.278	0.333	0.208	0.333
0.75	0.75	0.292	0.197	0.333	0.292	1.0	0.292	0.292	0.188	0.333
0.5	1.0	0.333	0.143	0.25	0.25	1.0	0.25	0.25	0.143	0.5
0.5	1.0	0.333	0.333	0.333	0.333	1.0	0.333	0.333	0.333	0.333
0.75	1.0	0.292	0.266	0.292	0.292	1.0	0.292	0.292	0.266	0.333
0.5	1.0	0.292	0.229	0.292	0.417	1.0	0.333	0.417	0.229	0.333

Figure 7.11: PROMI Merging Support: Concept Similarity Computation

The result of the process is an application ontology which can be stored alternatively as RDFS or OWL with a user-specified namespace.

7.1.3 Technical Details

The aim of this section is to give a general overview of the core issues of the implementation of the PROMI framework. This topic is not addressed in detail in the context of this thesis, as it is described in-depth by the main developers in [38, 126, 189].

External Components

The current release of the PROMI system makes use of several external tools and APIs:

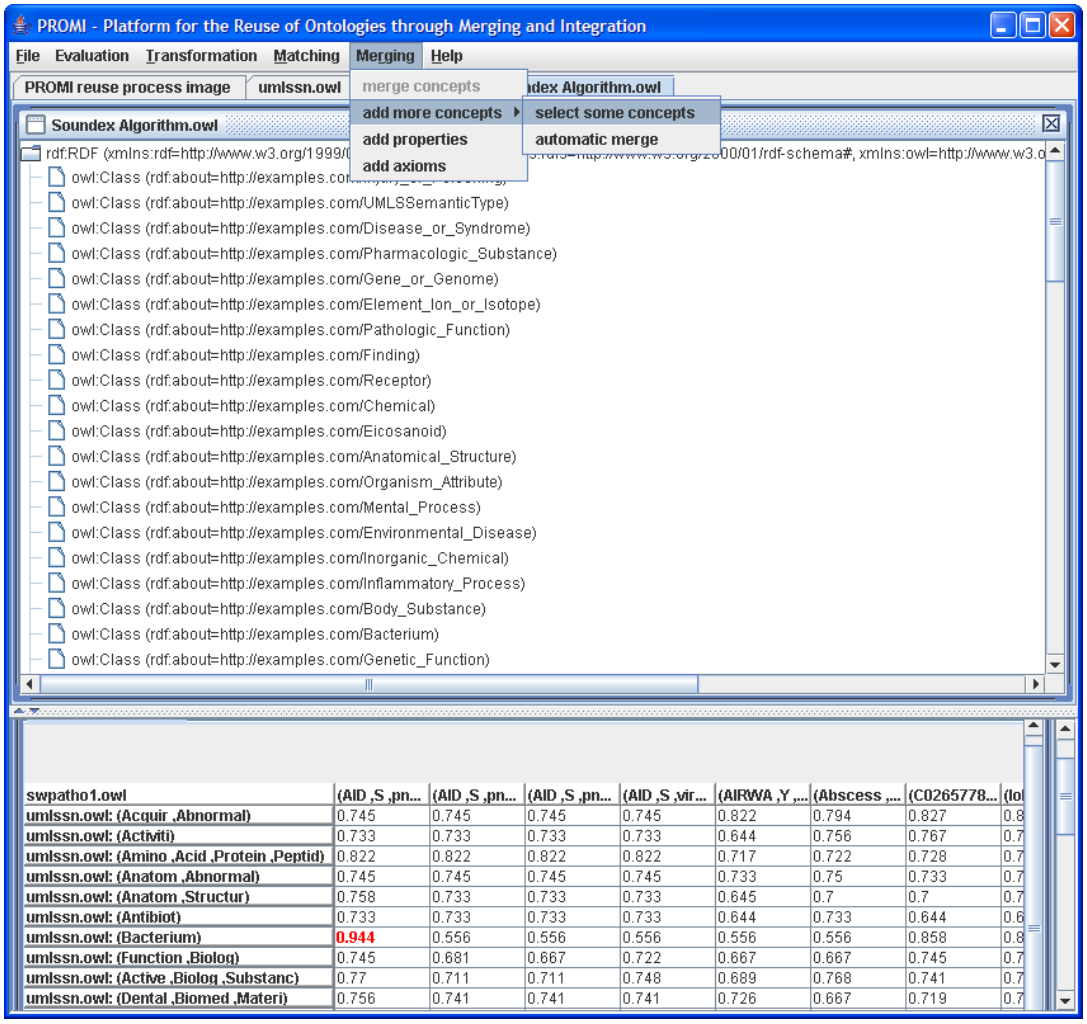


Figure 7.12: PROMI Merging Support: Incremental Merging Process

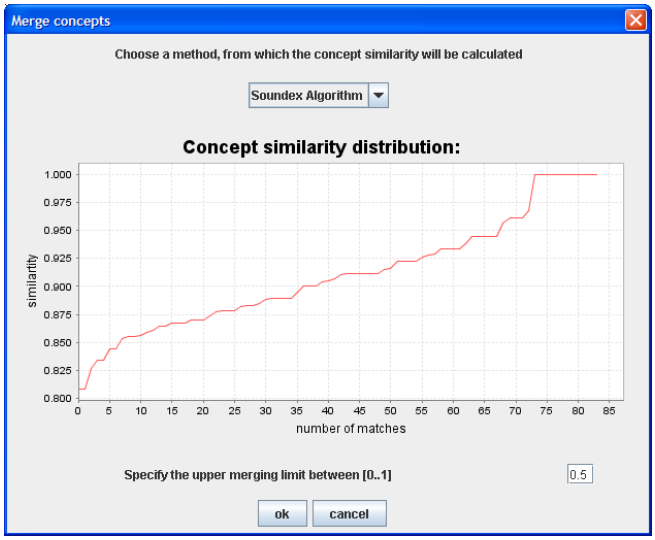


Figure 7.13: PROMI Merging Support: Distribution of the Similarity Results

Jena API: is a Java API for Semantic Web applications. It provides a rich suite of methods to create and manipulate RDF data as well as RDFS and OWL ontologies.[5]

RDF Validator: is an online tool which carries out syntactic correctness checking as regards the RDF-XML syntax specification.[6]

Inference Engines: Pellet and RacerPro are reasoners for RDFS and OWL (Lite and DL). They can be used to carry out consistency checking and to semantically answer various queries on ontology or method metadata at both schema and instance level.[7] Reasoning services are integrated into the PROMI implementation with the help of the DIG interface which is supported by the Jena API.

JavaNLP: is a Java Library comprising core natural language processing services developed at the Stanford University.[8] The PROMI platform uses the stemming and normalization features of this library.

SimMetrics: is a library implementing string distance functions. The similarity values are normalized in order to enhance their alignment.[9]

Similarity: is a Java implementation of a series of very popular taxonomical similarity measures for RDFS and OWL ontologies from the Free University of Berlin.[10]

[5] `http://jena.sourceforge.net` last visited in December, 2005
[6] `http://www.w3.org/RDF/Validator/` last visited in May, 2006
[7] `http://www.racer-systems.com/de/index.phtml`, `http://www.mindswap.org/2003/pellet/index.shtml` last visited in June, 2006
[8] `http://nlp.stanford.edu/javanlp/` last visited in June, 2006
[9] `http://sourceforge.net/projects/simmetrics/` last visited in March, 2006
[10] `http://page.mi.fu-berlin.de/~grote/swpatho/` last visited in June, 2006

JFreeChart: is a free Java chart library which provides easy-to-use means to design and integrate standard charts into arbitrary applications.[11] PROMI needs such functionality to support the decision of the ontology engineers upon the appropriate means to measure similarities between ontologies in the context of a merging or integration activity (cf. Figure 7.13).

JavaHelp: is the standard Java help system that enables developers to incorporate online help in applets, components, applications, operating systems, and devices. [12] It is used to generate the documentation of the tool (cf. Figure 7.3).

Packages Overview

Figure 7.14 provides an overview of the most important packages within the Java implementation of the PROMI platform. In the following we describe their main functionality before getting into more details of the ontology evaluation and merging components.

1. **Package `controll`**: covers the main frame of the platform and the main control class.
2. **Package `model`**: deals with the application logic and is further divided into 5 sub-packages:
   a) **Package `transformation`**: is concerned with the classes required to handle specific input formats for ontologies (RDFS, OWL, XMLS, DTD and DDL).
   b) **Package `matching`**: deals with matching services at linguistic and taxonomical level. It is used by both evaluation and merging components and consists of 2 sub-packages:
      i. **Sub-package `stringsimilarity`**: realizes various methods for computing resemblance between terms. The computation is realized using threads.
      ii. **Sub-package `conceptsimilarity`**: is used to calculate the similarities between concept labels of ontologies given a set of basic similarities for their compounding terms. Again threads are used in order to parallelize these operations for all ontological sources involved.
   c) **Package `merging`**: is responsible for aggregating concepts, relations and axioms to a final ontology based on the results of the concept similarity computations and of the user input.
   d) **Package `evaluation`**: implements the metadata-based ontology evaluation method.
   e) **Package `meta`**: is concerned with the context-sensitive selection of appropriate merging methods and the management of the metadata describing services and information sources these are expected to handle. Its sub-package `rules` implements the processing of the contextual dependencies.
3. **Package `view`**: contains all GUI classes for PROMI.

[11] `http://www.jfree.org/jfreechart/` last visited in June, 2006
[12] `http://java.sun.com/products/javahelp/` last visited in June, 2006

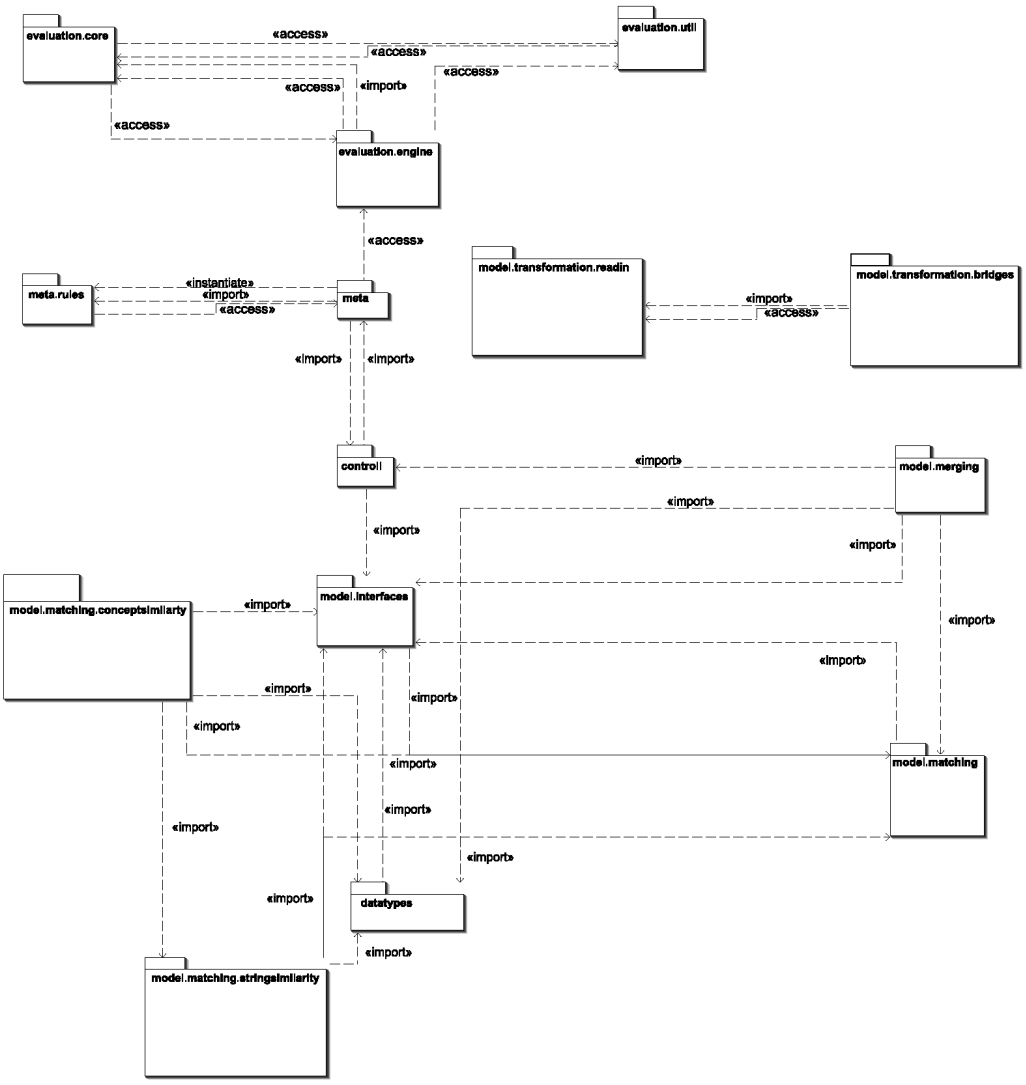


Figure 7.14: Overview of the PROMI packages

We now turn to the description of the two main reuse steps, which are supported by PROMI: ontology evaluation and ontology merging and integration, respectively.

Ontology Evaluation Component

The evaluation component can be functionally divided into (cf. Figure 7.15):

1. **Metadata Repository**: includes descriptive information about the ontologies, the metadata schema and the context model.
2. **Semantic Matcher**: is responsible for calculating semantic similarities between specific ontology features.
3. **Similarity Repository**: persistently stores method-specific similarities between concepts in the metadata ontology (cf. Chapter 6).
4. **Semantic Ranking**: processes the user-defined queries as described in the previous chapter and ranks the results according to the set of weights.

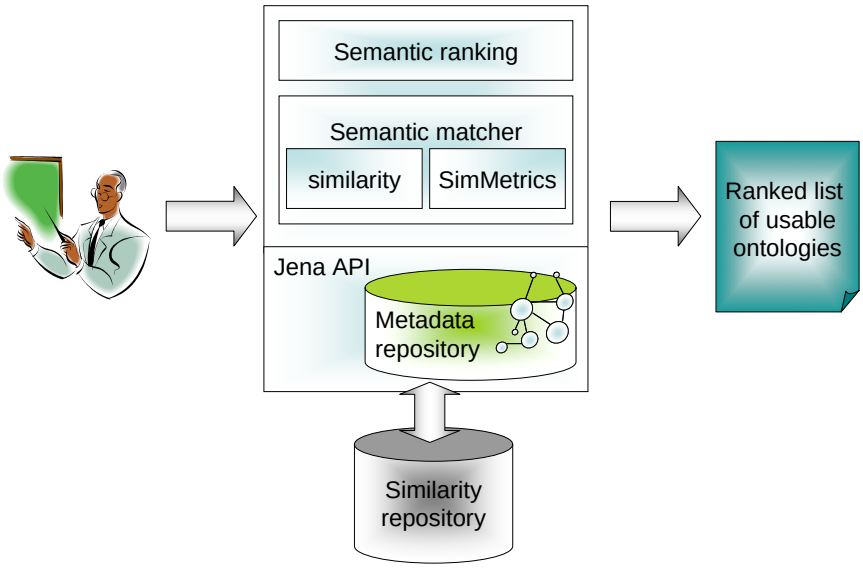


Figure 7.15: High-Level Architecture of the Evaluation Component

The metadata repository currently consists of the metadata schema and the metadata instances, stored separately in two files, which are online available. An alternative persistent storage system can be easily embedded using the corresponding Jena functionality. The semantic matcher builds upon the two aforementioned external libraries and the heuristic measures elaborated in the previous chapter, in order to compute the similarities between those ontology features (represented as ontological primitives in the metadata ontology) which have been identified as relevant for the ontology evaluation task. The data is stored persistently in a relational database (here MySQL), because of the resource-intensive nature of the similarity computations, which can not be carried out at run-time without significant performance

losses. Once the query vector has been specified and revised on the basis of the method-internal contextual dependency knowledge, the semantic ranking component uses the results of the matcher in order to calculate the scores for each of the available ontologies. Alternatively, the user could specify a threshold for the maximal number of hits to be returned. This is, however, not implemented in PROMI yet, as the number of ontologies covering the same or related domains, which were available for test purposes and beyond, is still manageable. Our experiences in the enumerated case studies confirmed this observation.

Packages The ontology evaluation is fundamentally implemented in the package `evaluation`, with its two sub-packages:

- **Sub-package `core`**: deals with the management of the evaluation criteria/ ontology features and with the processing of the manually defined similarities.
- **Sub-package `engine`**: is concerned with the query execution and results ranking.

New similarity measures on the basis of the proposed metadata model can be easily defined as a new `Metric` sub-class in the `core` package. The context-dependent behavior is realized generically with the help of a context ontology. Changes at this level can be realized by revising the latter.

Ontology Merging and Integration Component

The merging/integration component takes as input a list of ontologies (formalized in XML-Schema, DTDs, DDLs or Semantic Web languages) and assists the user in customizing, merging and integration them. Accordingly, it can be decomposed into the following modules:

1. **Translator**: deals with the transformation of the enumerated input languages to an internal abstract format consisting of concepts, properties and axioms.
2. **Matcher selector**: aids the ontology reuse participants in selecting the appropriate matcher services.
3. **Matching/merging**: detect the commonalities between the source ontologies and integrate them into a target resource.
4. **Repositories of ontologies, their metadata and metadata-driven similarity data**: contain the reuse candidates, as well as additional information required to perform certain reuse activities.

After the application-relevant ontologies have been identified, the reuse process continues with their transformation to a uniform representation format. In PROMI we implemented a one-to-many approach to the issue of language heterogeneity: every of the supported input formats is mapped into a simple ontological language consisting of concepts, properties, axioms and instances. This intermediary format can be easily transformed to a Semantic Web language, while avoiding the complexity of powerful formalisms such as OWL, which further differentiate among a multitude of types of relationships and axioms.

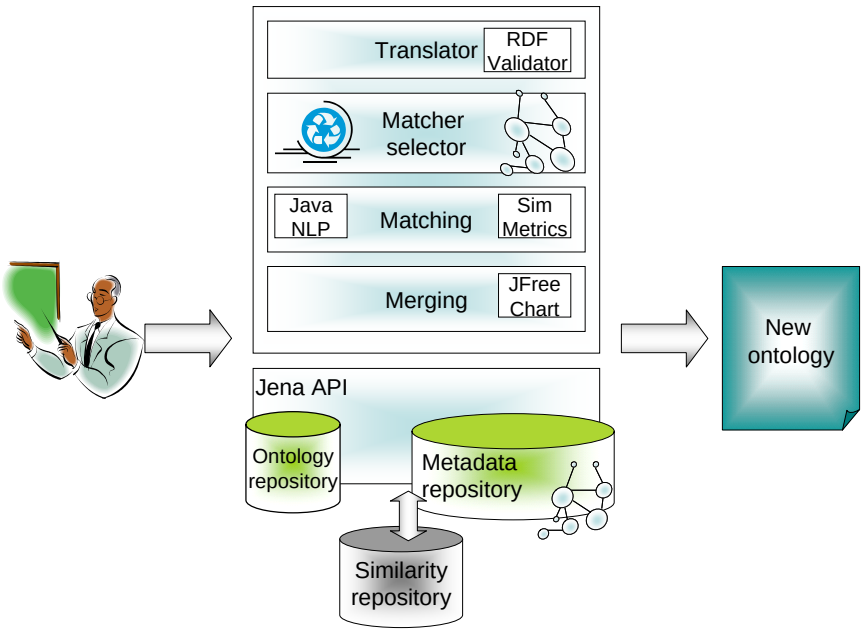


Figure 7.16: High-Level Architecture of the Merging and Integration Component

The current release of PROMI supports solely standard input formats for ontological sources: XML-Schema, DTD, relational databases (expressed using DDL), as well as Semantic Web languages. We are aware of the fact that this represents a limitation for the usability of the tool in arbitrary situations, notably in the actual context of the Semantic Web, which is characterized by the availability of high amounts of ontological knowledge in semi-structured and un-unstructured form. This drawback can be overcome however only partially in an automatized environment. While document-based ontology learning services could definitely be embedded to the system if available at a feasible quality, the question of how to automatically create Semantic Web ontologies from arbitrary classification systems can not be answered rigorously given the heterogeneity of these standards. As demonstrated by our case studies, each of the existing classifications or taxonomies is available in a proprietary format. Converting from this format, whose semantics is though semi-formal in most of the cases, to more standardized ontology representation languages typically requires dedicated parsers. This is the main reason why PROMI does not provide support for this class of problems, while it assumes users to translate their ontologies to one of the widely-used formats before processing it in the framework of a reuse process.

Further on, the sources need to be integrated into a final application ontology. In order to optimize this task, the PROMI framework applies contextual dependency rules as those which have been introduced in the previous chapter in order to select:

- which ontologies should be merged: ontologies covering similar domains (as computed by the DMOZ-driven similarity measure, cf. Section 6.1) are likely to be merged to a single model in a meaningful way. The remaining ontologies will be integrated into the final setting without any attempt to identify commonalities.

- how should ontologies be merged: with respect to this issue PROMI currently differentiates between
  - ontologies containing labels in the same natural language. These can be meaningfully aligned using natural language processing matchers.
  - ontologies which do not contain human-readable labels or which are described using different natural languages. Under these circumstances only structure-based matchers can be applied meaningfully.

  We do not consider ontologies with instances, as PROMI is targeted at ontology reuse and not data integration. Such a focus would primarily mean techniques and tools to merge heterogeneous data repositories instead of their underlying schemas. PROMI also avoids considering rules regarding ontologies containing axioms, as no matching or merging algorithms taking into account this additional level of complexity are available at a level of detail which would allow a reliable re-implementation.

The system accesses the metadata entries of the relevant ontologies, which trigger the generic rules eliminating unappropriate algorithms. Subsequently, PROMI performs the following steps:

- separation of ontological primitives by means of dedicated readers. In this step an ontology file is parsed, while concepts, instances, properties and axioms are identified.
- the labels denominating concepts (e.g., classes in OWL and RDFS) are prepared for the linguistic task. For this purpose the tool tries to transform every label (which can not be by default assumed to be a term listed in a lexicon) in a bag of terms which can be meaningfully compared. Common naming conventions are used as input for the completion of this configuration step, which is followed by stemming.
- similarity computation at term level performed using specific algorithms. The current release supports 12 algorithms, among which we mention Soundex, Hamming, Levenstein and QGrams [126].
- similarity computation at concept level followed by the merging of those concepts which have been accepted by the user. Furthermore, properties and axioms from the original ontologies can be manually added to the newly created resource.

Packages The merging/integration functionality is covered by the packages `transformation`, `meta`, `matching` and `merging`, respectively.

The component can be easily extended in order to support new input types and new element-based matchers [126]. The integration of schema/structure-based algorithms requires, however, additional implementation effort, particularly at view level.

To summarize PROMI provides many of the features which have been identified as useful for the operationalization of the reuse process during our requirements analysis and in the context of the reuse methodology(cf. Sections 3.5 and 4.5, respectively). The results of first user tests are very promising and enhance our motivation for further developing the

framework to improve the efficiency of reuse. The conceptual background is already available in terms of our methodology and methods and related work in our institute. These issues are further discussed in relation to the evaluation of our solution and in the discussion of the planned future work.

7.2 OntoMeta

In this section we provide details on the implementation of a tool for the automatic acquisition of ontology metadata. Building upon our ideas preliminarily elaborated in Chapter 5, the tool was conceived in collaboration with students at the Free University of Berlin, which were mainly responsible for its implementation and testing [38].

7.2.1 Design and Architecture

The primary goal of OntoMeta was to investigate to which extent meta-information about Web ontologies can be automatically generated. The tool should be easily accessible by ontology developers and integrable in metadata-based ontology engineering environments such as PROMI or ontology repositories such as Onthology. OntoMeta calculates pre-defined syntactic, semantic and pragmatic metadata information given an input in form of an ontology, and delivers a metadata entry structured according to our metadata schema.

The architecture of OntoMeta is depicted in Figure 7.17. It contains three components for the acquisition of each of the metadata categories, and uses online services such as those offered by Google and WordNet in order to gain access to Web knowledge, which can be of use to derive particular metadata entries as subsequently elaborated.

Figure 7.18 illustrates the start Web site of the tool, in which the user may input the URL of the ontology he requests the metadata for. Figure 7.19 shows the results of the algorithm, which can be locally saved as OWL using the button on the bottom of the page.

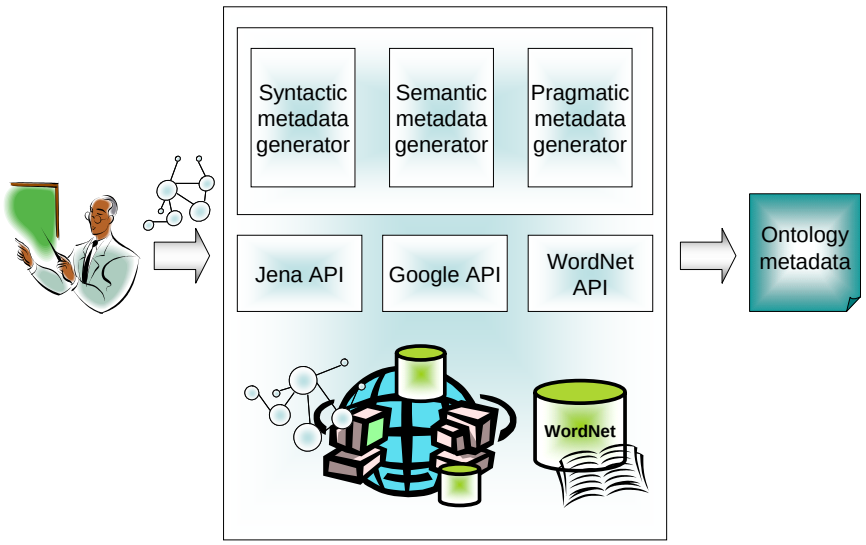


Figure 7.17: High-level Architecture of OntoMeta

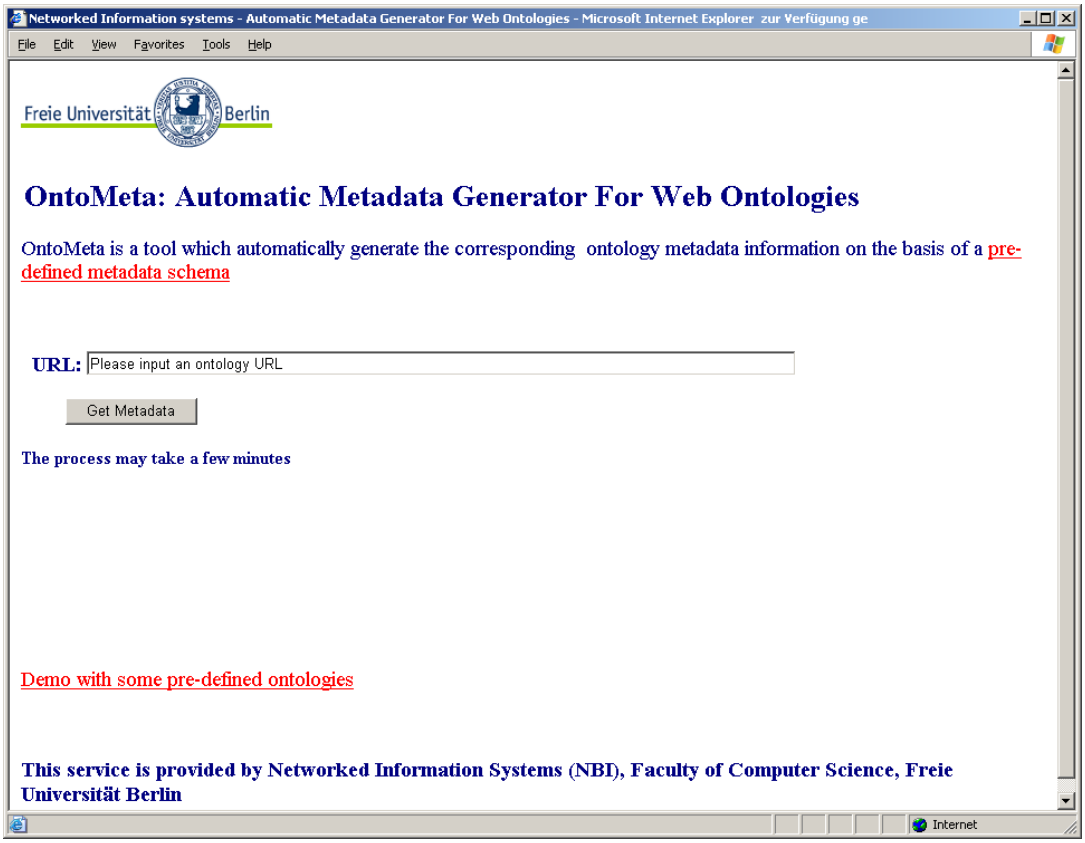


Figure 7.18: OntoMeta Start Page

OntoMeta: Automatic Metadata Generator for Web Ontologies

http://reliant.teknowledge.com/DAML/Government.owl

Metadata Item	Result	Comments
number of classes	52	number of classes defined in this ontology
number of properties	62	number of properties defined in this ontology
number of instances	944	number of instances defined in this ontology
number of axioms	0	number of axioms defined in this ontology
number of inheritance levels	4	depth of the inheritance tree
ratio of incorrect statements	0.0	validation result from OWL Validator
ratio of used syntax	0.386	result with the help of SAX parser
is this ontology consistent	true	measured by Pellet reasoner
number of comments	0	number of comments in this ontology
is this ontology readable	false	whether all class names are readable
ratio of readable labels	0.923	ratio of readable classes
is this ontology unambiguous	true	whether all class names are unambiguous
ontology type (domain generality)	domain	upper-level, domain or core ontology
ontology type (internal structure)	Frame	glossary, thesauri or frame
ontology domain	Government	the domain modelled by this ontology
view upon the domain	United_States	the view upon the described domain
natural language	American English	the natural language used in denominating ontological primitives
creation date	NoÂ information available	the creation date of this ontology
version information	No information available	the version information of this ontology
author information	No information available	the author information of this ontology
creation tool	No information available	tools used during the development process
documentation	http://reliant.teknowledge.com/DAML/Government.owl;	domain-relevant online documents
	http://reliant.teknowledge.com/DAML/ATO98MessageSet_Ontology.owl;	
	http://www.daml.org/ontologies/321;	
	http://www.daml.org/ontologies/434;	
	http://engforum.pravda.ru/showthread.php3?threadid=96442;	
	http://www.cgs.uiuc.edu/academics/courses/ps282.pdf;	
	http://www.theinternetparty.org/freedom/f_s2.php?section_type=fre&td=20020322131736	
	http://www.pej.org/html/modules.php?op=modload&name=News&file=article&sid=2686&mode=thread&order=0&thold=0;	

Download Metadata as owl File

This service is provided by Networked Information Systems (NBI), Faculty of Computer Science, Freie UniversitÃ¤t Berlin

Figure 7.19: OntoMeta Result Page

7.2.2 Technical Details

OntoMeta is implemented in Java in two variants: a client which can be installed and locally executed, and a servlet whose functionality can be exposed online (as illustrated in the screenshots above). Both accept as input a list of Semantic Web ontologies (i.e. implemented in RDFS or OWL) and produce an OWL file containing the metadata instances in conformity with our ontological schema.

External Components

Both implementations use a series of external libraries as follows

Jena API: is a standard Java API for Semantic Web applications. It provides a rich functionality to handle RDFS and OWL ontologies.[13]

RDF Validator: is an online tool which carries out syntactic correctness checking with respect to the RDF-XML syntax specification.[14]

OWL BBN Validator: is an online tool which checks the syntactic validity of Semantic Web ontologies with respect to the RDF-XML syntax of the OWL language.[15]

Inference Engines: Pellet and RacerPro are reasoners for RDFS and OWL (Lite and DL). They check the consistency of Semantic Web ontologies.[16]

Google Web API: Google provides a series of Java-based Web Services that allow application developers to interface with the Google-relevant Web content. In this thesis we primarily used two functions of the Google API: the search procedure to systematically query the Web and to process the obtained results, and the Google directory service.[17]

JWNL API: JWNL is a Java API for accessing WordNet-style relational dictionaries for the English language. Besides term look-up, it provides methods for inter-terms relationship discovery and morphological processing.[18]

Class Overview

Figure 7.20 gives an overview of the most important classes within the implementation of the OntoMeta, whose functionality will be briefly explained in the following:

1. Classes `SyntMeta`, `SemantMeta` and `PragMeta`: define methods for the acquisition of the syntactic, semantic and pragmatic metadata. Each metadata element is handled by a Java method. For the syntactic validity OntoMeta uses the BBN OWL

[13] `http://jena.sourceforge.net` last visited in December, 2005
[14] `http://www.w3.org/RDF/Validator/` last visited in May, 2006
[15] `http://owl.bbn.com/validator/` last visited in December, 2005
[16] `http://www.racer-systems.com/de/index.phtml`, `http://www.mindswap.org/2003/pellet/index.shtml` last visited in June, 2006
[17] `http://code.google.com/apis.html` last visited in June, 2006
[18] `http://sourceforge.net/projects/jwordnet` last visited in June, 2006

Validator API, while the consistency information is delivered by the reasoners RacerPro and Pellet accessed through the DIG interface within the Jena API. The semantic metadata generation makes use of the Google directory service and of WordNet knowledge accessed using the JWNL library.

2. Class `GoogleStuff`: contains methods for performing queries using Google (as common search engine and as directory service) and for processing the delivered results.

3. Class `JenaLib`: extends some specific functions of Jena API in order to maintain a clear structure of the implementation.

4. Class `Main`: for a set of ontologies, specified by URIs or local paths, this class triggers the generation of the metadata information and stores the results as individuals of our ontology metadata model.

In the remaining of this section we provide some details on the implementation of the most important metadata elements covered by our schema.

Syntactic Metadata Generator Component

The category of syntactic metadata contains those ontology features which are related to the implementation of the ontology, (as opposed to its conceptualization). The most representative syntactic metadata elements are thus quantitative nature, while the information captured by them is strongly implementation dependent. In its current release OntoMeta covers the following features:[19]

- number of classes, instances, properties, axioms, imported ontologies and inheritance levels
- language, syntax, as well as ratio of used syntax
- ratio of invalid statements.

If the model of an ontology is defined as pure OWL, the number of particular ontological primitives is delivered automatically by methods of the Jena API. We consider solely asserted properties and axioms. Inferred items could also be added to the system, however, the corresponding figures are highly reasoner-dependent. RDF and RDFS data can be handled similarly, if in addition we apply a reasoning step which maps all the `rdfs:Classes` or `rdf:classes` to `owl:Classes`. The depth of the inheritance tree is defined as the maximal generalization/specialization path length in an ontology. It is calculated recursively starting from the second level classes—since the class `owl:Thing` is the root class of each ontology—to the instances [38]. Individuals are considered in this case as there is no differentiation between them and classes at RDF level. As aforementioned the syntactic validation is performed using an external component. The ratio of the correct statements is the division of the number of errors returns from a validator (e.g., BBNs OWL validator) and the number

[19]The exact meaning of these features has been discussed in Section 5.3.

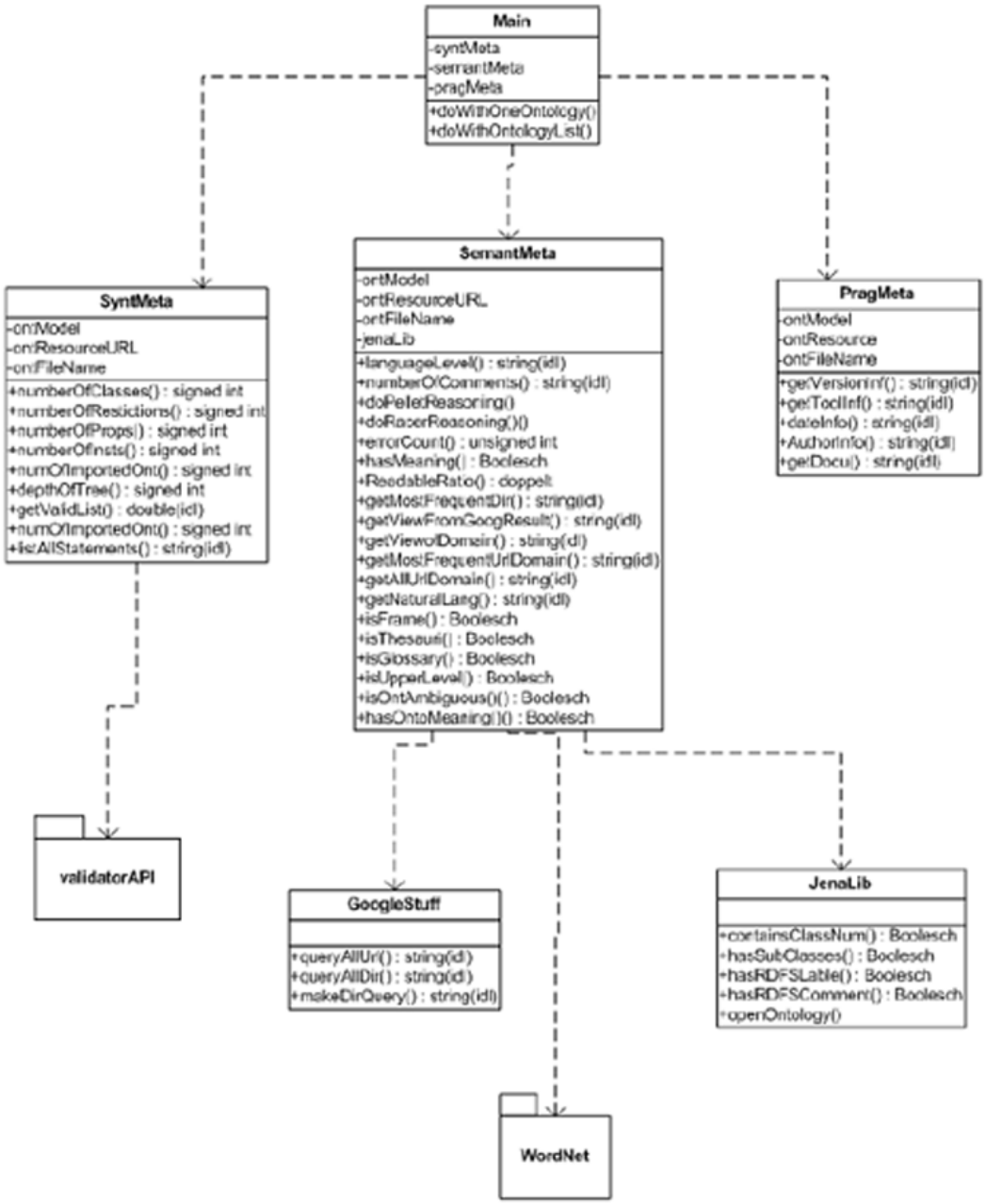


Figure 7.20: Overview of the OntoMeta Classes

of statements of the model as computed by Jena. The result for the ratio of correct statements is a value of type double. For the acquisition of language and syntax information we can also rely on dedicated Jena methods. Finally, the ratio of used syntax is implemented by checking whether a particular ontology model contains particular syntactic constructs and counting the positive hits.

Semantic Metadata Generator Component

Metadata in this category is related to the meaning of the content of the ontology as comprehended and interpreted by humans or by a reasoner. This includes on the one hand the domain which is modelled by means of the ontology, but also the formal semantics of the language which is used for the conceptualization. OntoMeta contains heuristics for the acquisition of the following semantics-driven features:

- readability and unambiguity of the labels used to denominate ontological primitives
- number of comments and definitions
- domain and view upon the modelled domain
- type of ontology
- natural language of the labels.

Label readability and unambiguity An ontology is considered to be readable, if and only if each concept name defined in this ontology has a meaning as stated by a thesaurus or a lexicon. An ontology is unambiguous, if and only if this ontology is readable and each concept name defined in this ontology has exact only one meaning. In its current implementation OntoMeta uses the WordNet thesaurus (or its relational variant accessed using the JWNL API) to derive the value of these metadata elements. WordNet is a comprehensive thesaurus of the English language, which has become a de facto standard in many natural processing tasks. Nevertheless, as the issue of multilinguality for WordNet is still subject of research and development in the computer linguistics community, the readability and unambiguity processing is currently restricted to English-labeled resources.

The readability checking algorithm firstly transforms every label in an ontology in a set of normalized terms taking into account conventional naming conventions. Further on, it looks-up in WordNet whether these terms are contained in the thesaurus, i.e. whether they have a meaning and can thus be interpreted by humans. If every of the compounding terms satisfies this condition, the ontology label is considered "readable". For example, the meaning of a concept name such as *"UnitedStatesRegion"* can be understood by humans, while a concept such as *"XYZTest"* will fail this test.

A pre-condition to establish whether an ontology label has an unambiguous meaning is its readability. Once this holds true, OntoMeta applies ideas elaborated in [108], which proposed the usage of semantic similarities for sense disambiguation. In this case these are provided by the length of the paths between synsets in WordNet [147].

Ontology type Ontologies can be differentiated by the generality of the modelled domain into upper-level, core, domain, and application ontologies (cf. Chapter 5.2). While the boundaries between core and domain ontologies, on the one hand, and domain and application ontologies, on the other hand, are known to be blurred, an automatic classification of an ontology in upper-level, core and domain can be reliably carried out if we consider the following distinctive features:

- **Upper-level ontologies**: Upper-level ontologies have the characteristic that they use a notably abstract vocabulary. For example terms such as "object", "unit", "relation", "abstract", "quantity", "class", "attribute", "process", "thing" are typical for this category. The domain modelled by such ontologies is clearly abstract, thus not being classifiable (using the heuristics explained below) to a specific DMOZ category.
- **Core ontologies**: The number of concept names defined in core ontology is lower than that of domain and application ontologies. Further on, the domain of interest of a core ontology is usually situated at the root or on the first level of the DMOZ directory (e.g., "Science", "Medicine", "Computers").
- **Domain ontologies**: The size of these ontologies is highly variable, while their domain of interest is more specific than that of a core ontology. OntoMeta considers for this purpose the levels of the DMOZ classification lower than 2.

A second classification of ontologies concentrates on the complexity of their internal structure, which partially correlates with their degree of formality. According to the way this classification is interpreted in our ontology metadata schema, OntoMeta is able to distinguish between classifications, vocabularies, and axiomatized ontologies. Vocabularies contain classes or concepts. Classifications or taxonomies additionally include a type of property (e.g., specialization/generalization, but also part-of or other types of relationships) by which concepts are ordered into a hierarchical structure. Axiomatized ontologies usually cover concepts, various properties and logical axioms or restrictions. The algorithm checks whether these types of constructs are supported by the ontology and returns the most complex type of ontology satisfying the tests.

Ontology domain The domain which is modelled by a specific ontology is defined in our ontology metadata schema by a combination of two inter-related topics in the DMOZ directory. While the first topic describes what is commonly understood as the "domain of interest" of an ontology, the contextual approach to ontology reuse described in this thesis employs an additional content descriptor for this purpose, the view upon the modelled domain. As explained before, this is intended to decrease the ambiguity with respect to the exact contents of an ontology, as each domain (such as, for instance, "medicine") can be modelled from a variety of perspectives ("science", "business", "insurances" etc.). Technically the view of a domain is defined as an element of the set of DMOZ super-topics of a given topic.

As a human, if we are expected to decide upon the domain described by an ontology, we typically associate the covered concepts (interpreted through their human-readable labels) to a broader category subsuming them. In addition, this procedure naturally eliminates potential naming ambiguities. OntoMeta simulates this line of reasoning in order to determine the

DMOZ topic which optimally characterizes the domain of interest of an ontology and the associated view of the ontology developers on this domain. In a first step the algorithm calculates a weighed vocabulary of the ontology, by splitting concept names according to predefined naming conventions, and weighing the resulting normalized terms by their frequency. Every term is used as a query for the Google directory service. The algorithm monitors the categories returned by Google for each query and aggregates the results taking into account the weights of each search term.[20]

Natural language of an ontology In absence of a Java-based technology for accessing available natural language detector tools, OntoMeta uses a Google-supported heuristics for determining the language used to denominate labels and to formulate comments and definitions within an ontology. It exploits the inherent language dependency of many of the Internet top-level domains (such as those related to a particular country or to types of organizations). After a preliminary step in which it computes a weighed vocabulary of the ontology—just as for the ontology domain feature as previously explained—the algorithm queries the Web using the terms within the vocabulary and counts the top-level domains taking into account the frequency of occurrence of each term. Then it maps each top-level domain to a natural language, thus determining the natural language used in the analyzed ontology. National domains are mapped to the official language of the country, while general ones (.edu, .org, .com etc.) are ordered for historical reasons to English.

Pragmatic Metadata Generator Component

Pragmatic metadata is used to describe development and deployment aspects of an ontology. The metadata schema foresees the following elements in this category:

- creation data
- authors and contributors
- versioning information
- engineering method
- engineering tool
- documentation
- application scenarios for which the ontology was developer for or already used in.

By contrast to the previous categories, this information is seldom recorded in any way in the ontology itself. Hence, and due to its unstructured nature, it can not be generated automatically (or even manually) in a reliable way. The automatic acquisition of pragmatic metadata fundamentally depends on the availability of monitoring information produced during the creation or usage of the ontology by the tools applied as technological support in these

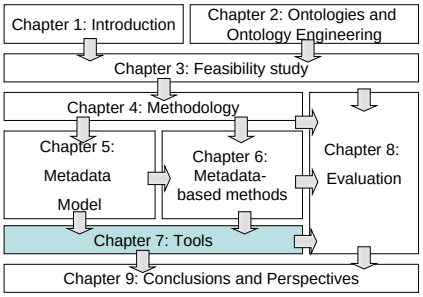
[20] Refer to [38] for a more detailed description of the heuristics by which one can select the most representative DMOZ topics to characterize the two metadata entries.

activities. This requirement is not fulfilled by the majority of ontology engineering tools to a satisfactory extent. Available metadata information is restricted to the creation date, the author and the tool processing the ontology at most.[21]

The creation date is usually recorded in an ontology using the Dublin Core metadata element `dc:date`, XML comments such as `<!-- Created: Thu Oct 10 15:45:43 2002 -->` or `rdfs:comments`. Versioning information, though scarcely available, is formulated in terms of constructs such as `owl:priorVersion` and `owl:versionInfo`, as well as XML comments. Provenance information can be found as the value of the `dc:creator` property, while tools are mainly present as comments e.g., `<!-- Created with Protege (OWL Plugin 1.1 beta, Build 129) -->`. These cases are handled by OntoMeta.

In summary, OntoMeta provides a suite of heuristics to acquire many of the most important ontology features captured by our ontology reuse metadata schema. While many important pragmatic metadata can not be generated without human intervention, considerable parts of the remaining metadata elements proved to be reliably extracted in an automatic manner. The results of the evaluation of the enumerated algorithms are presented in the next chapter.

7.3 Summary



This chapter introduced several tools which aim to support the systematic operation of ontology reuse processes. We first gave an overview of the PROMI platform. The tool, realized in collaboration with members of the working group Network Information Systems at our university, prototypically implements many of the core ideas elaborated in this thesis. It provides methodological and technological support for two of the three fundamental reuse process steps: ontology evaluation and merging and integration, respectively. We then addressed the issue of metadata creation and management, presenting OntoMeta, a tool which automatically acquires reuse-relevant information about Web ontologies. The way these these tools have been tested and the results of this endeavor are discussed in the next chapter, which is dedicated to the evaluation of our solution.

[21] In a study of over 100 ontologies, approximatively 20% contained information on the enumerated topics.