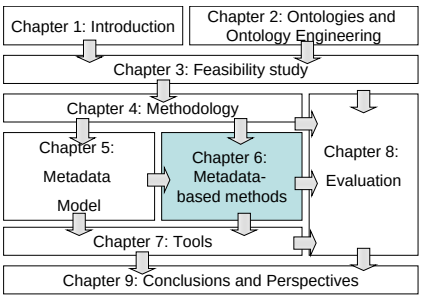
6 Methods Supporting Ontology Reuse



In this chapter we approach methods to aid ontology engineering participants in their attempt to reuse existing ontological resources. We introduce a method for evaluating these resources in regard to their context-oriented usability and briefly address an approach to optimize current techniques on merging and integration (Sections 6.1 and 6.2, respectively) These methods rely on the requirements derived during the design of our ontology reuse methodology and utilize the ontology metadata model described in Chapter 5. They are prototypically implemented as integral part of the PROMI platform.
***References**: This chapter is based on the publications [141, 142, 149, 150, 168, 170, 171].*

6.1 Ontology Evaluation Method

This section describes a method which aids participants in an ontology reuse process in examining ontologies with respect to their suitability in a given application setting. From a methodological perspective it builds upon the ideas introduced in the ontology reuse process description. There we proposed a component-oriented approach to ontology evaluation which foresees the completion of five evaluation tasks:

1. **Content evaluation**: primarily intends to answer the question whether the analyzed ontology models a domain which is relevant in a particular context.

2. **Knowledge representation evaluation**: addresses the quality of the conceptual model underlying an implemented ontology from a domain-independent perspective.

3. **Technical evaluation**: pragmatically estimates the suitability of an implemented ontology given the current state of the art with respect to technology and tools.

4. **Application evaluation**: refers to the usability of the ontology for a particular application system, task and purpose.

5. **Availability evaluation**: is strictly related to the costs and conditions for using an existing ontology in new applications.

Further on, the methodology pointed out the need for particular methods and tools which are expected to contribute to the operationalization of the evaluation activities. A central requirement was related to the *availability of metadata information*. This issue was addressed in the previous chapter. Furthermore we singled out there the need for a context-sensitive approach to ontology reuse and implicitly, to ontology evaluation. This resulted in the formulation of *context-specific guidelines* which influence particular decision making activities within ontology engineering—from the selection of the suitable ontologies to the choice upon tools and methods successfully handling them.

On the basis of these considerations we designed the method for ontology evaluation which is presented here. The method preserves the context-sensitive behavior with regard to ontology reuse, refines the situation-specific guidelines introduced in the methodology, provides process support, and utilizes the ontology metadata model as basis for the operationalization of the evaluation task. The method has been prototypically implemented within the PROMI framework, which is the subject of the next chapter. There we will go into more details about the concrete realization of the method and give examples of its utilization in practice.

6.1.1 Method Overview

From a process-driven perspective the ontology evaluation method consists of four steps:

1. **Context specification**: firstly the user is asked to provide information about the context of the ontology evaluation task. Which information is considered contextual and the way it is further involved in the process is explained in Section 6.1.2.

2. **Features selection**: in this step the engineering team specifies context-independent evaluation criteria and their importance for the application setting. The criteria correspond to ontology features captured by the ontology metadata model. This model was designed to describe the whole range of activities in an ontology reuse endeavor, including those aspects which are relevant for the operationalization of ontology evaluation approaches.

3. **Usability computation**: the feature selection is adjusted to the actual context and translated to a semantic query on the basis of a vector model [6]. The features are associated to *user-defined* and *context-specific weights*, which contribute to the computation of the ranking of the results. The *context-specific weights* are calculated internally by the ontology evaluation tool. The tool disposes of process knowledge formalized in terms of contextual dependencies which forms the basis for such calculations. The revised query is executed on the metadata repository. The assessment of the criteria against the available metadata is *semantic* in that it uses reasoning services and ontology-based similarity measures to detect and rank the significant results.

4. **Result assessment**: the user decides whether to conclude the ontology evaluation and to proceed with the third step of the ontology reuse process (i.e. customizing and integrating the reusable ontologies). A second option is to re-run the current step in case of unsatisfactory results. This re-iteration implies a revision of the feature selection, followed by the usability assessment, which inherently produces a new list of potential reuse candidates.

Conceptually the method is built upon the following parameters:

1. **Ontology features** F_i which are part of the ontology metadata model and are associated with **weights** w_i normalized to a specific numerical range (see below).
2. **Similarity measures** S_i associated to the ontology features F_i.
3. **Contextual information** describing ontology evaluation activities. Every ontology evaluation endeavor is described in terms of its *context* C, referring to information about the *application environment*, the *analyzed ontologies* and the *participants* carrying out a particular evaluation activity.

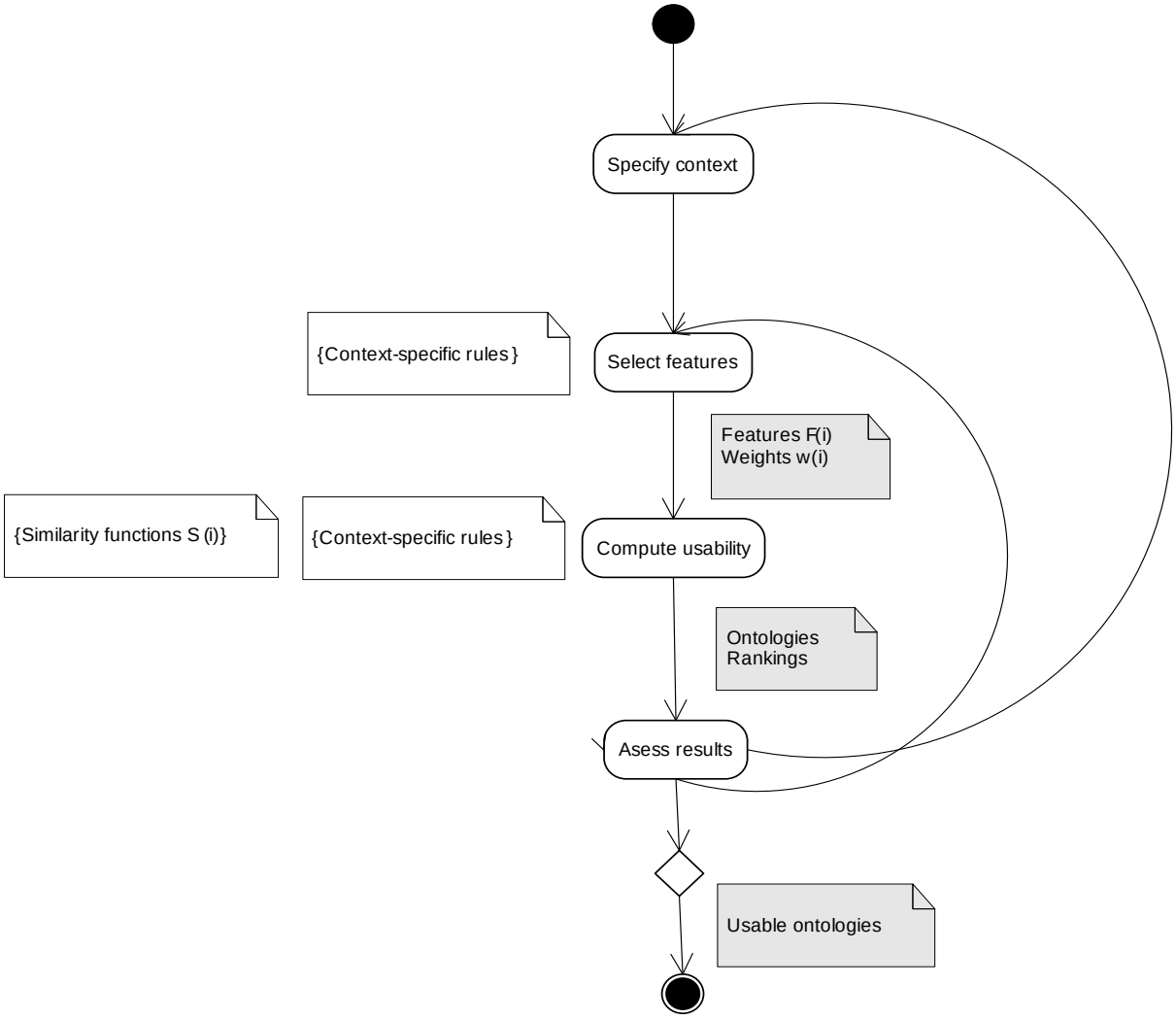


Figure 6.1: Ontology Evaluation Method

The ontology evaluation workflow is illustrated in Figure 6.1. We now turn to a detailed description of its main steps.

6.1.2 Context Specification

In the first step the ontology evaluator is expected to state the ontology reuse context precisely. He indicates his role in the engineering process, details about the application scenario and optionally the reuse level. This information is used to select the appropriate evaluation criteria and to customize the evaluation questionnaire which is presented to the user in the next step. Such contextual dependencies are stored within a *context model*, formalized independently of the ontology evaluation method by means of an *ontology*. Application-driven optimizations of the evaluation results are encoded in terms of a query rewriting heuristic at method logic level (see below).

Context Ontology

The context model is in accordance to the one underlying our ontology reuse methodology, though customized to the characteristics of the ontology evaluation task. The contextual information is clustered in four categories [166]:

1. **User-related information**: the participants at the evaluation process: ontology engineers, programmers, domain experts, users. This information is required to customize the way the evaluation criteria are presented to the evaluator.

2. **Task-related information**: information about the ontology evaluation task in terms of the prospected level of reuse and evaluation dimensions.

3. **Environment-related information**: this type of information primarily relates to the application scenario in which the final ontology is going to be used. In particular we concentrate on the *purpose* the final ontology will be utilized for, described in terms of *ontology tasks* and *roles*, adopting the same items as in our ontology reuse methodology. Knowledge about the application scenario induces particular evaluation requirements to be prioritized higher than others, as we will explain bellow.

4. **Target-related information**: the *ontologies* being involved in this analysis, described by the available metadata.

According to this model, the context in which the ontology evaluation is performed is characterized by a particular level of reuse, a particular user role, a particular application scenario and the metadata repository describing the available ontologies. The goal of our method is thus to determine the relevant target ontologies provided contextual information about the remaining context dimensions: user, task and environment and ontology metadata.

The question of how to define, formally represent and use context information is acknowledged as one of the most challenging research topics across various disciplines in computer science. As highlighted in the introductory chapter of this thesis (cf. Section 1.4.3) research on context can be divided into two mainstreams: on one hand, approaches focusing on the nature of context and the way this type of information can be capture for machine processing; on the other hand, those concerned with the usage of contextual aspects in improving the quality of particular information services. Our research can be categorized in the second category. Instead of striving for giving full particulars on the nature of context in relation

with ontology reuse and on its computer-aided representation, our aim is to demonstrate how a pre-defined set of contextual aspects can contribute to the systematic operation of ontology reuse and to the development of support methods and tools. Issues of context representation and reasoning are secondary for our work, therefore the context ontology commits to existing proposals in the field.

Figure 6.2 depicts an overview of the context ontology, in terms of the main classes, the sub-class relationships between them and their instances.[1] In the following we elaborate on the content of the context ontology in more detail:

- `ApplicationScenario`: comprises the application scenarios considered by our ontology reuse methodology.
- `Feature`: covers usability-relevant ontology features referencing classes or instances of the ontology metadata model. Every feature refers to application scenarios, user roles, metadata entries etc. as illustrated in the subsequent example.

  The feature `representationLanguage` is stated to be relevant for the application scenario `semanticSearch`. In the evaluation questionnaire this feature will be verbalized using the string value of the `hasQuestionText` property. The user is provided with a `multipleChoice` answer to this question, while the answers refer to the `individuals` of the meta.owl class `RepresentationLanguage`.

  The feature `view` is targeted by contrast to both ontology engineers and domain experts. For each of the target groups the context specifies a different verbalization of the evaluation criterion.

```
<Modelling rdf:ID="representationLanguage">
  <rdfs:comment xml:lang="en">representation language</rdfs:comment>
  <hasApplicationScenario rdf:resource="#semanticSearch"/>
  <hasValueType rdf:resource="#individual"/>
  <hasSelectionType rdf:resource="#multipleChoice"/>
  <hasOptions rdf:resource="...meta.owl#RepresentationLanguage"/>
  <hasApplicationScenario rdf:resource="#semanticAnnotation"/>
  <hasApplicationScenario rdf:resource="#integration"/>
  <hasText>
    <Text rdf:ID="representationLanguage_OE">
      <hasTargetGroup rdf:resource="#ontologyengineer"/>
      <hasQuestionText rdf:datatype="xsd#string">
        In which representation language should the ontology
        be implemented?
      </hasQuestionText>
    </Text>
  </hasText>
</Modelling>

<Content rdf:ID="view">
  <rdfs:comment xml:lang="en">
```

[1] The documentation has been generated using the Protégé ontology engineering environment, just as in the case of the metadata ontology.

```
    view of the developer team upon the domain modelled by the ontology
    </rdfs:comment>
    <hasText>
      <Text rdf:ID="view_OE">
        <hasTargetGroup rdf:resource="#ontologyengineer"/>
        <hasQuestionText rdf:datatype="xsd#string">
          Which view upon the specified domain should the ontology
          model?
        </hasQuestionText>
      </Text>
    </hasText>
    <hasOptions rdf:resource="dmoz#Topic"/>
    <hasApplicationScenario rdf:resource="#integration"/>
    <hasSelectionType rdf:resource="#multipleChoice"/>
    <hasApplicationScenario rdf:resource="#neutralAuthoring"/>
    <hasText>
      <Text rdf:ID="view_DE">
        <hasTargetGroup rdf:resource="#domainexpert"/>
        <hasQuestionText rdf:datatype="xsd#string">
          A domain of interest can be regarded from different
          perspectives. Depending on the view of the ontology
          developer on the domain, one can imagine various
          ontologies. For example an ontology about cars
          will look differently when developed by car manufacturers
          or insurance companies. What is the view upon the specified
          domain you are interested in?
        </hasQuestionText>
      </Text>
    </hasText>
    <hasValueType rdf:resource="#individual"/>
    <hasApplicationScenario rdf:resource="#semanticSearch"/>
    <hasApplicationScenario rdf:resource="#semanticAnnotation"/>
  </Content>
```

- `Interval`: is an additional class introduced in order to cluster ontologies by their size. We differentiate between four categories of ontologies as regards this parameter:
  - ontologies with less than 100 primitives (small ontologies)
  - ontologies with less than 1000 primitives (medium-sized ontologies)
  - ontologies with less than 5000 primitives (large ontologies), and
  - those with more than 5000 primitives (very large ontologies)

  This classification, which was derived by studying the statistical overviews provided by current ontology repositories and search engines (cf. Section 2.2.2), will play a role in the definition of numerical similarity measures in the next section.

- `OntologyTask`/`OntologyRole`: are imported from the metadata ontology.

- `TargetGroup`: comprises the main participants in the ontology evaluation tasks. These are modeled as instances of the class: `ontologyengineer`, `domainexpert`, `programmer` and `user`.

- `Type`: is an artificial class modeling specific answer types for the questions verbalizing the evaluation criteria. Examples of instances of this class are `multiplechoice`, `singlechoice`, `singleinput`, `multipleinput`.

- `Value`: describes various types of values allowed as answers for the evaluation questionnaire. Instances of this class are for example `individual` or `ontClass`. In case of the latter the answer range of a particular question comprises all sub-classes of a specific class. In the former, this is defined as the set of individuals of the class.

- `Weight`: comprises a set of ratings for the importance of ontology features. These can be modeled as individuals. The prototypical implementation of the method is based on five ratings, from `unImportant` to `veryImportant`.

The usage of an ontology as a means to model contextual information has the benefit of separating the contextual dependencies related to the structure and the form of the evaluation tool from the usability assessment heuristics. In this way one can easily adapt the method and its implementation (including the user interface) to new user roles, application settings, user preferences etc.

Impact of Context on the Usability Assessment

The analysis of the case studies revealed a series of contextual dependencies at application scenario level. Their fundamentals are an integral part of the ontology reuse methodology, which points out context-specific issues for each of the proposed process stages and their activities (cf. 4). At method level we further elaborated these dependencies in order to enable their automatic processing:

Integration: in this scenario the ontology is expected to automatically mediate among heterogeneous data repositories and applications using this data. According to this goal we derived the following evaluation criteria:

- The conceptualization of the ontology should be consistent in order to enable the execution of correct inferences which might be required to define mappings between heterogeneous sources or to support interprocess communication.
- The definitions of the ontological primitives should be non-ambiguous. Ambiguity might have consequences on the generation of linguistics-based matchings between the input schemas.
- The implementation of the ontology should be syntactically correct. Otherwise it can not be used programmatically.
- The ontology should be formal, as its semantics is expected to be processed by machines.
- The ontology should cover as many features of the source and target systems as possible in order to minimize information losses. This can be primarily reduced to a compatibility of the formalisms used to structure the data in the corresponding environments.

- ApplicationScenario
  - *Instances : semanticSearch, integration, neutralAuthoring, semanticAnnotation*
- Author
  - *Instances : teamAuthored, singleAuthored*
- Feature
  - Availability
    - *Instances : isNotFreelyAvailable, isFreelyAvailable*
  - Comprehensibility
    - *Instances : documentation, naturalLanguage, readableLabels, comments, ambiguousLabels*
  - Content
    - *Instances : view, conceptsIncluded, typeByStructure, typeByGenerality, domain*
  - Design
    - *Instances : usedInApplicationSystem, possibilityOfIntegration, usedBefore, usedForTask*
  - Engineering
    - *Instances : createdUsingTool, automatizationLevel, involvedEngineers, createdUsingMethod, createdUsingMethodology*
  - Modelling
    - *Instances : representationLanguageSyntax, consistency, formalityLevel, representationLanguage, syntacticalValidity*
  - Origin
    - *Instances : originInfo*
  - Scope
    - *Instances : numOfProperties, numOfClasses, numOfImportedOntologies, numOfInstances, numOfAxioms*
- Interval
  - *Instances : lte100, lte1000, gt1000, lte500*
- meta:OntologyRole
- meta:TasksPerformedByOntologies
- NaturalLanguage
  - *Instances : de, es, it, en, fr*
- TargetGroup
  - *Instances : ontologyengineer, domainexpert*
- Type
  - SelectionType
    - *Instances : multipleChoice, check, multipleInput, singleSelection, singleInput*
  - ValueType

Figure 6.2: Excerpt of the Context Ontology (Classes, Sub-Classes and Instances)

- [225] recommends the usage of ontologies of manageable size and simple structure. These two factors have a direct impact on the complexity of the reasoning services and affect thus the performance of the integration task.

Semantic search/retrieval: ontologies can be used in various ways in an information retrieval system. Firstly, they can function as a controlled query and indexing vocabulary. Secondly, provided a rich axiomatization of the application domain, these machine-understandable facts can be used to improve the precision and recall of traditional search heuristics. This includes the situation in which the ontology is utilized as a matching platform for defining similarities between the items in the repository. In this context ontologies are also a means to declaratively define search preferences and user profiles, which induce a customized behavior of the search component. Lastly, once a list of presumably relevant documents has been computed, the ontologically modeled knowledge can be used to provide domain- or user-specific presentation and navigation services. The presentation of the results can be adjusted to the user preference ontology. The results can be clustered by their annotations. Navigation paths can be defined

along relationships between ontological concepts which are referenced in particular documents. Taking into account these usage patterns we identified the subsequent requirements for the ontologies to be reused:

- The level of formality of the ontology should be at least semi-formal. An informal ontology which is implemented in a language without machine-processable semantics can not be used effectively to enhance the quality of the search heuristics.
- Since this quality improvement is guaranteed only if the ontology can be processed automatically by various tools (e.g., reasoning services) it is vital that it is syntactically and semantically valid.
- The utilization of the ontology for vocabulary purposes by humans is inconceivable without the availability of natural language labels verbalized in an appropriate natural language. This means that the natural language of the ontology should be the same as the one used by the retrieval system to formulate the queries.
- Enhanced navigation and visualization functionality can be provided only on the basis of ontologies which contain domain-specific relationships.
- If the ontology is expected to form the basis for the definition of similarity measures, which are a valuable means to improve conventional retrieval mechanisms in closed domains, then it is essential that the ontology contains a a homogeneous taxonomical structure (with respect to the number of classes pro inheritance level), a sufficiently large number of inheritance levels and a well-balanced distribution of the ontology instances.

Semantic indexing/annotation: in the simplest case the ontology provides a vocabulary which forms the basis for indexing (manually or automatically) information items such as images or textual documents in a repository. Further on, a more structured and formal representation of the application domain could form the basis for automatic linguistics-based annotation techniques. The following criteria have been identified as relevant for this scenario:

- Ontological primitives should be labeled using natural language terms in order to allow humans to use the ontology for classification tasks and to support NLP.
- The labels should adhere to naming conventions in order to ease the automatic annotation process.
- The natural language used to label the ontology should correspond to the one used in the documents to be annotated if this task is to be performed automatically. Otherwise, it should be the same as the one used to query the annotated documents in order to avoid supplementary translations. If the information items annotated with the help of the ontology are formulated in a different natural language, or are not of textual nature (e.g. multimedia) the annotations can not be generated automatically or semi-automatically. In this case the ontology users manually insert references to ontological primitives to the information items in the repository.
- It is of benefit to use comprehensive ontologies, which maximize the domain coverage in order to be able to classify a wide range of information items adequately.

- In terms of the structure of the ontology, this should be "thesaurus-like". This means a relatively simple structure covering specialization/generalization relationships complemented by a small set of domain-specific ones.
- The ontology engineers should prefer ontologies which themselves have been extracted from document corpora. These are likely to fulfill many of the aforementioned requirements provided a representative collection of input sources in an appropriate natural language.

Software engineering: building upon research in the area of knowledge-based configuration and design (cf. for example [15, 17, 91, 211]), model-driven software engineering foresees the usage of ontologies for design purposes. In this context ontologies act as a means to model domain knowledge independently of the underlying system implementation (refer to [157] for a recent overview). Beyond this use case ontologies are increasingly utilized in configuration tasks: they model valid configuration parameters and their interaction within the software system [157]. Expressing such constraints in a language with a machine-understandable semantics has the essential advantage that incorrect configurations can be reliably identified automatically using reasoning services. Given this twofold utilization an ontology to be reused in software engineering scenarios should fulfill the following requirements:

- The ontology should be correct from a syntactic and semantic perspective. This requirement holds true for both use cases, as ontologies are used by machines be that for the generation of software architectures, or the identification of invalid system configurations.
- For configuration purposes the ontology should be highly axiomatized. This implies that the meaning of the concepts organized in the sub-class hierarchy should be formalized explicitly in the ontology using constraints. Every more specific concept inherits properties and attributes from the concepts positioned higher in the specialization / generalization hierarchy. The difference between this concept and its generalizations can be expressed by restricting the inherited properties or by defining new ones. Otherwise the system will not be able to identify missing parameters or invalid parameter combinations. For design purposes the ontology should provide a high-quality description of the application domain. This does not impose any constraints with respect to the structure of the ontology, but assumes a satisfactory domain coverage and a correct modeling of domain facts and the compliance with common modeling practices.
- The size of the ontology can cause performance problems in the usage of Semantic Web frameworks or the execution of particular inference tasks. This also applies for particular language constructs, which are not commonly supported by reasoners or other ontology management environments.[2] If such facts are captured by the ontology metadata (i.e. that concrete domains can not be reasoned

[2] See for example the performance tests and the comparison between well-known reasoners such as Pellet, RacerPro and Fact++ described in [199] and at `http://www.mindswap.org/2003/pellet/performance.shtml`, last visited in August, 2006.

over in any of the existing tools) this dependency can be taken into account in the usability computation.

Knowledge representation: ontologies are acting as means to represent the most relevant facts about a particular domain of interest in terms of concepts, relationships, instances, axioms or rules. Every ontology potentially reusable in a knowledge representation endeavor should satisfy the following requirements:

- The ontology should be easily understandable by its potential users. This implies first of all the availability of documents, comments, definitions and human-readable labels. Secondly, the size and the complexity of the graph structure underlying the analyzed ontologies is a factor which surely affects its ease-of-use. However, there is a trade-off between size and domain coverage, as well as between structure and ambiguity.
- There is no constrain regarding the level of formality of the ontology, though semi-formal ones are acknowledged to perform better in many everyday situations [225].
- The ontology should minimize its ontological commitments [84]. This is one of the key design principles for building reusable ontologies, which, however, is acknowledged to be unpracticable in pre-defined application scenarios.

These dependencies are in relationship to particular metadata entries, as illustrated in Table 6.1.

Additionally to the ontology features one can consider a context-sensitive revision of the relevant *evaluation dimensions* depending on the application scenario and the reuse level. Such dependencies have been addressed from a process-oriented perspective, autonomously of any means to operationalize ontology evaluation, in the reuse methodology (cf. Section 4.3) At method level however, this additional refinement does not affect the set of ontology features to be analyzed: most of them occur in at least one of the evaluation dimensions whose significance does not depend on any contextual aspects. As per the proposed methodology, three evaluation dimensions are recommended to be taken into consideration independently of the ontology reuse context—though their operation might be optimized depending on particular contextual aspects: the content, the application and the availability evaluation. The technical and knowledge representation evaluation are of value for particular applications and levels of reuse. The way these activities are performed is, however, context-independent (cf. Chapter 4).

Formalizing Context Dependencies

The ontology reuse guidelines introduced in the previous section can be exploited to complement the process support offered by the ontology reuse methodology with more automatized techniques. A pre-requisite for this new level of operation is the encoding of this body of expertise in a machine-processable form.

A first possibility is to implement contextual dependencies formally as if-then rules which refer to concepts of the metadata ontology. The subsequent usability assessment could be exercised using a reasoning service on the knowledge base consisting of rules, the metadata

APPLICATION SCENARIO	ONTOLOGY FEATURE
Integration	Validity and consistency for machine-processability Non-ambiguous labels for matching Manageable size for comprehension and performance Formal representation for reasoning Simple structure for mappings
Semantic search/ retrieval	Semi-formal or formal representation for reasoning Validity and consistency for reasoning Taxonomical structure for reasoning Domain relationships for navigation and presentation Natural language labels for query formulation Well-balanced structure for ontology-based matching Manual engineering for domain representation
Semantic indexing/ annotation	Natural language labels for manual indexing Natural language of the documents Naming conventions for automatic indexing Thesaurus-like structure Ontology learning generation Large-sized ontology for domain coverage
Software engineering	Formal representation for reasoning High axiomatization for configuration checking Validity and consistency for machine processability
Knowledge representation	Documentation, definitions, comments, labels for comprehensibility

Table 6.1: Alignment of Ontology Features to Application Scenarios

ontology and its instances describing particular ontologies. An alternative to the rule-based approach is to make use of this background knowledge to adjust the user-defined query. In this case the input criteria specification is revised so that that lacking important criteria are added to the query and existing ones are re-weighed. Our method follows the second approach for various reasons, the most important ones being directly related to the lack of reasoning tools currently able to provide the necessary functionality and to the advantages of vector models in fuzzy information retrieval [6].

Algorithms and feasible implementations being able to infer over heterogeneous knowledge bases combining DL-oriented ontologies and their rule extensions are still in their infancy in the Semantic Web field. In the same time, the usage of weights and of the vector model allows a flexible behavior of the system in comparison to the rule-based approach and guarantees a well-proved quality of the retrieval algorithms. This flexibility is essential for processes of evaluation, which—beyond every type of operational support—are still targeted at humans and thus retain a certain degree of subjectivity. A rule-based approach assumes that these pre-defined rules are equally important and hold true per default; they are triggered in a reasoner-specific order to answer the queries. User-defined preferences can not interfere with this task, being reduced to an input for the ranking of the final results. By contrast, the weights mechanism enables us to achieve a balance between the importance of the system-internal expertise and the wishes of the users. This flexible behavior is intensified by the usage of similarity measures. They are a reasonable alternative to reasoning as they preserve

the ultimate human-targeted nature of evaluation [27, 75] while additionally taking into account expert knowledge on ontologies which is hardly representable in structured rules.[3] A third alternative, which is not considered in this thesis in more detail, is to apply probabilistic query models to retrieve the appropriate information. The core advantage of this approach against the classical vector model resides in theory in the provision for mutual independency between vector features. In practice however the effect of such correlations on the overall performance (e.g., between a level of formality and a particular type of ontology) is in the opinion of many information retrieval specialists questionable [6].

6.1.3 Features Selection

First the evaluation method pre-selects which ontology features optimally fit to the actual context, which was specified by the ontology evaluator in the previous step. This decision is supported by the context ontology, which captures information about the way the metadata on ontologies is relevant to pre-defined contextual dimensions.

The ontology metadata defines a comprehensive set of reuse-relevant ontology features in terms of ontology classes, attributes, properties and instances. At implementation level they are typically modeled as ranges of the pre-defined properties of the class `Ontology`, thus providing a viable basis for the realization of a generic customization strategy for the evaluation task. The context model associates to each ontology feature application scenarios, reuse levels, user roles, weights as well as natural language verbalizations in form of evaluation questions (cf. Section 6.1.2). The evaluation tool analyzes this data and selects for the specified context the relevant features and their layout.[4]

The ontology evaluator then indicates the importance of the presented features according to his experience and to the recommendations of the methodology.[5]

Note that some of the ontology features presented to the user for selection do *not* depend upon any contextual information. In this category we particularly mention the *content*, *availability* and *application* evaluation dimensions and the ontology characteristics such activities are supposed to involve. A small set of features are, however, relevant solely to application-scenario-specific activities such as the knowledge representation and the technical evaluation. The sum of all ontology features is presented in Table 6.2.

6.1.4 Usability Computation

Once the context of the ontology evaluation is defined the method applicant makes his choice upon the context-independent relevant ontology features, specifies their value range and their importance. These inputs are translated to a semantic query which is rewritten on the basis of the internal knowledge on the contextual dependencies between particular application settings

[3] We will follow the rule-based approach for the more technically centered ontology merging and integration method.

[4] The context-sensitive generation of the layout is addressed primarily in the next chapter, which is dedicated to implementation issues. The way this issue is handled in the context ontology is explained in the example in Section 6.1.2.

[5] Note that many of these recommendations are already taken into account automatically using formal contextual dependencies.

TASK	ONTOLOGY FEATURE
Content evaluation	Ontology domain View upon the domain Type of ontology Keywords Documentation
Knowledge representation evaluation	KR-language KR-paradigm Validity and consistency Specific metrics
Technical evaluation	KR-language and syntax Tools used to build the ontology Level of formality
Application evaluation	Level of formality KR-language and syntax Natural language Specific metrics Systems using the ontology Ontology task and role Generation methods and methodologies
Availability evaluation	Provenance Versioning Development status Costs

Table 6.2: Ontology Features Relevant for the Ontology Evaluation Step

and effectively usable ontologies. During the rewriting phase the query is subject to two types of transformations:

- **Expansions**: the query is expanded with ontology features—acknowledged to be essential to viably assess the usability of existing sources— that are missing from the user specification (i.e. they are rated as unimportant). For example, if their is no information on the structure of the desired ontology employed in a classification task, the query should be expanded with the requirement for a taxonomical model. The changes induce an improvement of the precision parameter, without any loss with respect to recall. Firstly, the newly added evaluation criteria are clearly relevant for the corresponding setting (cf. Chapter 3). Secondly, the results of the original query are still included to the final hit list, but with a different ranking.

- **Modifications**: the prioritization of the ontology features might be subject to adjustments in that some weights are revised on the basis of the tool-internal knowledge. In a semantic annotation scenario it is essential that the language of the documents is the same as the language of the ontology labels. If this constraint has not been rated high by the user the value of the corresponding weight is adjusted internally. Note that these changes do not imply a decrease of the recall value, but merely a different ranking of the final results.

The semantic query can be modeled as a vector of feature-weight pairs [188]:

$$\overrightarrow{q} = (F_1, w_1) \dots (F_i, w_i) \tag{6.1}$$

where F_i are evaluation criteria associated to the elements of the metadata model and w_i indicate priority ratings in the interval $[0, 1]$. As repeatedly mentioned in this chapter, the context ontology captures information about the importance of the properties of the potentially relevant ontologies in a given ontology reuse context. This information is stored by means of the property `hasPresetValue`, which associates default weights to evaluation criteria and application scenarios. This default value is inserted in the query vector if the corresponding criterion has been ignored by the engineering team. Otherwise, the user-specified value is compared to the default and modified if lower than it to a concrete degree. The exact percentage should be set up empirically. In our experiments (cf. Chapter 8) we used an increment of 30% with promising results. For instance, if the range of the weights is defined as a discrete set of 3 numerical values, this correction would imply a scale-up of the initial weight with one position. In case of a continuous interval ranges, the scale-up is performed straight forward.

The revised query is sequentially evaluated against the metadata describing available ontologies using ontology-driven similarity functions. The individual similarity results are assembled to a final ontology ranking value taking into account the weights associated with each feature.

Similarity Measures

The similarity measures have the aim of providing a means to compare among values of predefined metadata elements and quantify the results of these comparisons. Every similarity measure can be defined as a function

$$sim(x, y) : D \rightarrow [0, 1] \tag{6.2}$$

where D is the set of values of a certain ontology feature, $sim(x, y) = 1$ for $x = y$, and $sim(x, y) = sim(y, x)$ for each x, y in D.

Recalling the metadata model and its ontology-evaluation-relevant contents described in Section 6.1.2 we can differentiate between six types of similarity functions:

- similarity functions for *numerical data types* such as integer or float. The simplest similarity indicator for numerical values is the Euclidean distance. For our scope and purpose it is useful however to utilize an interval-based similarity function instead of the conventional measure, as this kind of precision is not required for the ontology engineering field. This holds true for metadata entries such as those capturing the *number of specific ontological primitives* or the *size of the overall ontology*.
- similarity functions for *URIs*: we use a two-valued similarity function which takes the value 1 in case of equality.
- similarity functions for *time-related entries*: date information is approximated at annual level, so that time entries located in the same year are considered equal. If this is not

true, one can define again interval-based similarity functions which diminish the strict effects of a numerical distance measure in accordance to application-specific expertise.

- similarity functions for *strings*: for this purpose one can apply one of the plethora of string-based distance measures implemented in various libraries (cf. Chapter 7).
- similarity functions for *ontological primitives*: these can be manually pre-defined by consulting the opinion of ontology engineering experts or automatically computed using ontology-specific similarity measures relying on linguistic and/or taxonomical criteria. The decision upon how to generate the requested similarities primarily depended on the organization of the ontological instances within the metadata model. In case we are dealing with a set of ontological instances referencing the same concept (i.e. all instances of a specific class), the similarity measure can be computed only on the basis of their labels in conjunction with human judgement. Structure-based measures can be meaningfully applied only when the set of ontological primitives is organized in a tree-like manner and the underlying structure is well-balanced (as regards the distribution of the number of child nodes for each node). Which ontology-specific similarity measures for which primitives are applied, how these measures are configured, or how experts define them manually varies from scenario to scenario. The suitability of existing similarities depends on the structure of the metadata ontology. The human-driven estimations clearly depend on her level of expertise. Consequently, the values applied in the usability computation step proposed in this work can not be considered universally valid. This work can solely claim to provide general guidelines about how such measures could be defined and justify the decision taken for the current implementation of the method.

In the following we introduce the similarity functions for each of the relevant metadata elements.

Similarities for numerical values This category includes the following metadata elements: `numberOfClasses`, `numberOfProperties`, `numberOfAxioms`, `numberOfInstances`, `numberOfInheritanceLevels`, `ratioOfUsedSyntax`, `ratioOfPropertiesPerClass`, `ratioOfCommentsPerClass` and `ratioOfDefinitionsPerClass`.

For two ontologies the number of inheritance levels as well as the relative measures should be compared using the Euclidean distance. For the remaining absolute metrics we can define intervals as those contained in the context ontology for the size dimension. There we differentiate among four categories for small, medium, large and very large ontologies, respectively. The associated similarity function is in this case:

$$\begin{aligned} sim_{size}(O_1, O_2) &= \frac{4 - |size(O_1) - size(O_2)|}{4}, where \\ size(O) &= 1 \; for \; every \; small \; ontology \; O \\ &= 2 \; for \; every \; medium \; sized \; ontology \; O \\ &= 3 \; for \; every \; large \; ontology \; O \end{aligned}$$

$$= \quad 4 \; for \; every \; very \; large \; ontology O \tag{6.3}$$

In the same manner one can define similarity measures for the remaining quantified syntactic metadata. These are, however, not explicitly considered in our ontology evaluation method so far. Indirectly, we do account for the presence of taxonomical relationships or axioms in relation with ontology types (see below).

Similarities for formality levels As introduced in Chapter 5 the degree of formality of an ontology is modeled as an object property with the range of type `Instance` of `FormalityLevel`. This class has three fixed instances: `formal`, `semi-formal` and `informal`, which have been associated to the numerical values 1, 0.7 and 0. This unbalanced scoring is justified by the way ontology engineering research seems to interpret the meaning of the `semi-formal` term in relation to ontologies: a semi-formal ontology is implemented in a language with a formal semantics or in a language whose semantics can be automatically translated to more formalized ones.[6] By contrast informal conceptual models can be "ontologized" primarily by humans. The similarity function is defined as follows:

$$\begin{aligned} sim_{formality}(O_1, O_2) &= 1 - |formality(O_1) - formality(O_2)|, where \\ formality(O) &= 0 \; for \; every \; informal \; ontology \; O \\ &= 0.7 \; for \; every \; semi - formal \; ontology \; O \\ &= 1 \; for \; every \; formal \; ontology \; O \end{aligned} \tag{6.4}$$

Similarities for ontology types The ontology metadata model distinguishes between types by structure and types by the level of generality of the domain. In our case we rely on the ontological continuum by [138] for the former, while the latter is primarily due to Guarino [86]. Based on these classifications and the way the authors define the corresponding ontology types,[7] an ontology engineering expert has manually assessed the similarity measures as illustrated in Table 6.3 and 6.4, respectively. The assigned values can be interpreted as a mapping of the enumerated classifications—de facto totally ordered sets—to the interval $[0, 1]$.

Building upon the classification in [138] the ontology engineering expert clustered the structure-driven ontology types in conformity with the complexity of the ontological primitives supported; The resemblance among plain structures such as catalogues, glossaries and vocabularies is accordingly higher than between them and taxonomical or even more complex sources. Taxonomies and classifications are more similar to each other than compared to thesauri, which contain different types of relationships. Formal ontologies are similar to thesauri, though to a lower extent because of the significantly richer representation of the domain of interest they provide.

[6] Note, that this numerical distribution does not claim to hold true in any application scenario. It is a hypothesis which has to be demonstrated to hold true in the empirical experiments using the evaluation tool.

[7] The opinion of the authors on these issues is widely accepted by the ontology engineering community.

	Catalogue	Glossary	Vocabulary	Classification	Taxonomy	Thesaurus	Axiomatized Ontologies
Catalogue	1	0.8	0.8	0.2	0.2	0.4	0.2
Glossary	0.8	1	0.8	0.2	0.2	0.6	0.2
Vocabulary	0.8	0.8	1	0.2	0.2	0.4	0.2
Classification	0.2	0.2	0.2	1	0.8	0.6	0.4
Taxonomy	0.2	0.2	0.2	0.8	1	0.6	0.4
Thesaurus	0.4	0.6	0.4	0.6	0.6	1	0.6
Axiomatized ontologies	0.2	0.2	0.2	0.4	0.4	0.6	1

Table 6.3: Similarities between Types by Structure

	Upper-level	Core	Domain	Task	Application
Upper-level	1	0.6	0.4	0.4	0.2
Core	0.6	1	0.8	0.6	0.2
Domain	0.4	0.8	1	0.4	0.6
Task	0.4	0.6	0.4	1	0.4
Application	0.2	0.2	0.6	0.4	1

Table 6.4: Similarities between Types by Generality

The similarity distribution for the generality types of Web ontologies was achieved in a similar manner with the help of the ontology engineering expert possessing knowledge about the classification by [86] and related ones. Core ontologies cover the most important concepts of a domain of interest. Therefore they are closely related to upper-level, but also to domain ontologies. Domain ontologies are, on the other hand, very similar to application ontologies, as a clear separation of the two is not provided in practice. Task ontologies share the same generality level as core ontologies, though covering a different focus in the domain of interest.

Similarities between representation languages We further defined similarities for knowledge representation languages commonly used in the ontology engineering field. They are represented in our ontology metadata in terms of instances of the class `OntologyRepresentationLanguage`:

- The Semantic Web ontology languages OWL Lite, DL and Full are defined to be similar to each other to a degree of 90% (i.e. a similarity of 0.9).
- The three OWL flavors are defined to be similar to RDF, RDFS and DAML+OIL to a

degree of 70% (i.e. the similarity value is set to 0.7)

- DAML+OIL is equally similar to RDF and RDFS to a degree of 70% (the value of the function equals 0.7)
- the aforementioned languages are similar to a degree of 60% to F-logic and to 40% to XML/XML-Schema, which corresponds to a value of the similarity measure of 0.6.

Alternative similarity functions could make use of the knowledge representation paradigm underlying each representation language (i.e. consider DL languages similar as opposed to logic programming language dialects). The similarity distribution chosen for the ontology evaluation method corresponds to a more technological view upon the current directions of research and development in the knowledge representation field: we cluster languages which have originated in the same context closer than the ones which are logically equivalent based on the assumption that for the former ontology engineers are likely to dispose of a wider range of tools and methods to deal with the language heterogeneity than for languages arisen in timely and locally dispersed communities.

As syntax information is directly related to representation languages, we consider syntax variants of the same language to be similar to 90%. Otherwise the similarity value is per default set to 0. The high score of 0.9 is justified by the fact that most of the tools provide support for multiple syntactic variants of the same language. A clear differentiation between different syntaxes is seldom relevant for the ontology engineering area.

The syntax information is utilized as a modifier for the similarity between representation languages:

$$sim_{lang,syntax}(O_1, O_2) = sim_{lang}(O_1, O_2) * (1 - mod_{sytax}(O_1, O_2)) \tag{6.5}$$

Similarities between keywords Keywords describing the content of an ontology are captured in the metadata model in form of strings. Similarities between keywords are calculated hence using string-based measures, which are not further specified at this point (refer for example to [42] for an study on string matching).[8] For example, the PROMI framework, which incorporates a prototypical implementation of the method presented in this section, uses both syntactic and semantic similarity measures for strings (cf. Chapter 7).

Similarities between ontology tasks and roles The tasks ontologies are developed for in a specific setting—or the ones they are subsequently involved in beyond their originating context—play a central role in our ontology reuse research. The proposed ontology metadata explicitly deals with the purposes ontologies are characteristically utilized in terms of two classes, `OntologyTask` and `OntologyRole`, respectively—in each task to be accomplished an ontology is assumed to act the roles of a `vocabulary`, a `formal model`, a `filter` or an `index`. A set of representative knowledge processing tasks, which are

[8] We assume that keywords are captured by the ontology metadata model in a controlled way, so that each keyword corresponds to a particular ontological primitive. This additional constraint leads to a more reliable similarity measurement. However, its fulfillment can not be guaranteed with ontology modeling means and should be ensured by quality checking mechanisms within ontology management tools.

widely expected to take benefit from employing ontologies, was compiled as a result of a comprehensive literature study. These are realized in the metadata model as instances of the corresponding task. The link to the adequate roles is implemented in terms of OWL constraints (cf. Chapter 5).

In order to assess similarity values for each task pair we can choose among two options: we can resort to ontology-based similarity measures—which are based on taxonomies, as linguistics do not apply to discrete term sets of this size—or define a new distance function reflecting the expert-perceived relatedness of the items. As the ontology metadata does not organize the ontology tasks taxonomically (a fact which was not negatively pointed out within the scope of the evaluation of the model) we choose the second possibility. The similarity values assigned by an ontology engineering expert do correspond to a four level classification of the tasks in correlation with a simple path-based distance. These values are displayed in Table 6.5.

	Annotation	Indexing	Query formulation	Configuration	Filtering	Query rewriting	Personalization	Matching	Search	Integration	Mediation
Annotation	1	0.8	0.6	0.2	0.2	0.2	0.2	0.2	0.4	0.2	0.2
Indexing	0.8	1	0.4	0.2	0.2	0.2	0.2	0.2	0.4	0.2	0.2
Query formulation	0.6	0.4	1	0.2	0.6	0.8	0.2	0.2	0.8	0.4	0.4
Configuration	0.2	0.2	0.2	1	0.4	0.2	0.2	0.2	0.2	0.2	0.2
Filtering	0.2	0.2	0.6	0.4	1	0.6	0.6	0.2	0.2	0.2	0.2
Query rewriting	0.2	0.2	0.8	0.2	0.6	1	0.4	0.2	0.8	0.4	0.4
Personalization	0.2	0.2	0.2	0.2	0.6	0.4	1	0.2	0.4	0.2	0.2
Matching	0.2	0.2	0.2	0.2	0.2	0.2	0.2	1	0.6	0.6	0.6
Search	0.4	0.4	0.8	0.2	0.2	0.8	0.4	0.6	1	0.2	0.2
Integration	0.2	0.2	0.4	0.2	0.2	0.4	0.2	0.6	0.2	1	0.8
Mediation	0.2	0.2	0.4	0.2	0.2	0.4	0.2	0.6	0.2	0.8	1

Table 6.5: Similarities between Ontology Tasks

Annotation was estimated to be closer related to indexing, query formulation and search as it can be considered a pre-requisite of the latter two, while showing major overlapping with the first one. In the same time indexing enforces retrieval, therefore we defined the task search and indexing to be similar to 40%. Query formulation is naturally related to query rewriting and search. Further on, a resemblance cluster was defined between query rewriting, filtering and personalization (mutual similarity of 60%), while query rewriting is a common method to enable mediation and subsequently integration. A second task which is essential to the latter is matching. Therefore we define it to be similar to the other two to a degree of 60%.

In order to increase the precision of the task similarity function, we extend it with a role-centered modifier. As aforementioned ontology tasks are closely associated to specific ontology roles. As these two features are strongly correlated, it is not recommended to model them separately within the evaluation query [6]. In our case we set up a two-valued increment for

the impact of the ontology roles on the resemblance between tasks: if the roles related to two tasks coincide the similarity function remains unchanged (i.e. the modifier takes the value 1). Otherwise we diminish the total value with 50% (i.e. the modifier is 0.5). In this way if we compare an application scenario in which an ontology is used as annotation vocabulary with one in which it should act as a formal basis for the realization of reasoning-based information retrieval, the final similarity value will be 0.1. This result is valid, as the characteristics of the ontologies optimally usable in the two scenarios are very different. Finally, if an ontology plays multiple roles within an application scenario (and the underlying application tasks) the modifier is determined by comparing the two role sets, an overlap indicating the nominal value.

The similarity function for tasks and roles is thus defined as follows:

$$sim_{task,role}(O_1, O_2) = sim_{task}(O_1, O_2) * (1 - mod_{role}(O_1, O_2)) \tag{6.6}$$

where the modifier $mod_{role}(O_1, O_2) = 1$ if $roles(O_1) \bigcap roles(O_2) \neq \emptyset$ and 0 otherwise.

Similarities between application systems Software systems potentially applying semantic technologies are represented as sub-classes of the class `OntologyApplication` in the metadata model. We adopted the classification provided by the Wikipedia encyclopedia, as one of the most recent and representative account in the IT field. For the scope of this method we confined us to manually assess similarity values to those software categories, which are commonly accepted to take advantage from using ontologies, and for which this application has already been concretized in real-world projects.

We interviewed an IT professional affiliated to industry, which was asked to define the degree of relatedness between a sub-set of the Wikipedia software taxonomy using values between 0 and 1 on the basis of his knowledge on the corresponding functional characteristics of the software categories. The result is presented in Table 6.6.

The main similarity clusters are defined in a straight forward manner. Firstly, systems concerned with management of information are grouped together. Secondly, there is a natural relationship between applications supporting sharing of data and those concerned with the collaborative accomplishment of tasks. A third category of systems, which were considered to be relevant for ontology engineering, is aimed at natural language processing. The most important cross-cluster similarities are between information and multimedia retrieval (because of the common goal), between content management systems and electronic and desktop publishing and between computer graphics applications.

Similarities between contents The similarity measure of the contents of two ontologies makes use of the specialization-generalization relationships defined in the DMOZ directory: as introduced in the ontology metadata model the domain of an ontology and the view upon this domain uniquely identify the contents of an ontology. They are represented as topics in the DMOZ classification, so that the possible views upon a domain D are those topics T for which $D\ isSubTopic\ T$ holds true.

For each pair of ontologies O_1 and O_2 we defined their content-oriented resemblance as follows:

APPLICATION SYSTEM	SIMILAR APPLICATION SYSTEM	SIMILARITY VALUE
Information retrieval	Information extraction	0.6
	Content management	0.6
	Knowledge acquisition	0.6
	Multimedia retrieval	0.6
	Multimedia generation	0.4
Multimedia generation	Multimedia retrieval	0.8
Computer graphics	Image Processing	0.8
Community support	Data sharing	0.6
Data sharing	Groupware	0.8
	Electronic publishing	0.4
	Groupware	0.6
Content management	Electronic publishing	0.8
	Desktop publishing	0.8
Language generation	Machine translation	0.8
	Linguistic processing	0.8
	Text analysis	0.8
Machine translation	Linguistic processing	0.8
	Text analysis	0.8
Text analysis	Linguistic processing	0.8

Table 6.6: Similarities between Application Systems

$$sim_{contents}(O_1, O_2) \quad = \quad sim_{domain}(O_1, O_2) * sim_{view}(O_1, O_2) \tag{6.7}$$

As both are defined as topics in the DMOZ directory, the domain- or view-driven resemblance between two ontologies can be calculated using the same measure. For this purpose we can apply various established approaches which make use of the labels denominating topics and the ones exploiting the graph structure of the DMOZ directory [42, 178]. An analysis of the DMOZ wording reveals that the most relevant string similarity functions in this case are those based on edit distances, as the classification distinguishes between various parts of speech sharing the same stem. For example, we terms such as “Medical” and “Medicine” are used to denominate *different* nodes of the DMOZ directory. Moreover graph-based functions should consider both path distances, but also the inheritance level the compared nodes are ordered to, as topics placed at a higher level in the hierarchy tend to refer to more generic concepts. In Chapter 7 we will present the prototypical implementation of our ontology evaluation method and provide more details on the similarity measures we used and their performance.

Similarities between engineering methods Engineering methods are relevant for the application evaluation dimension. The idea underlying this assumption is that the method an ontology has been originally developed through inherently correlates with specific features of the outcomes and fits better to particular purposes people usually use ontologies for. The ontology metadata model describes engineering methods in terms of methodologies and techniques, which are executed manually or automatically to construct ontologies on the basis of

various inputs. This includes classical ontology engineering methodologies delivering process descriptions by which engineers are guided towards their final goal, but also methods which generate ontologies from other ontologies or from other information structures.

As the aforementioned parameters strongly correlate, we designed an aggregated similarity measure for engineering methods which combines the effects of similar input types and automatization level information:

$$sim_{engineering}(O_1, O_2) = sim_{input}(O_1, O_2) * sim_{automatization}(O_1, O_2) \quad (6.8)$$

Input types are pre-defined in the ontology metadata as direct or indirect instances of the generic class `SemanticWebResource`. This subsumes not only ontologies, but also Web documents, databases or named fragments of ontologies. Their mutual similarities have been assigned by an ontology engineering expert based on features such as formality and structure of their contents as depicted in Table 6.7.

	Ontology	Ontology Fragment	Web Document	Database
Ontology	1	1	0.4	0.8
Ontology Fragment	1	1	0.4	0.8
Web Document	0.4	0.4	1	0.4
Database	0.8	0.8	0.4	1

Table 6.7: Similarities between Semantic Web Resources as regards Engineering Methods

Ontologies and ontology fragments can be considered to be equivalent in this context, as an engineering method accepting as input one of them does produce the same results. Further on, the higher similarity between databases and the former is justified by their commonality with regard to formality.

Further on, the automatization level is represented in the metadata model as a set of instances of the class `AutomatizationLevel`: `manual`, `semi-automatical` and `automatical`. If we map these fixed individuals to the range $\{0, 0.5, 1\}$ the function $sim_{automatization}$ in the equation 6.8 is reduced to

$$sim_{automatization}(O_1, O_2) = 1 - |automatization(O_1) - automatization(O_2)| \quad (6.9)$$

In this way manual methods are considered more close to semi-automatical ones, which are further related to the full automatization.

Similarities between natural languages Natural languages are modeled within the metadata ontologies as strings and are referenced by the datatype property

`hasNaturalLanguage`. In the ontology management field differences at language level are important usability constraints for manual tasks, while scenarios using ontologies internally, as for configuration or mediation are not affected by such representation issues. In the latter the natural language is not relevant as evaluation criterion. In the former different languages have a similarity value of 0, because of the mentioned limitations such incompatibilities impose.

Similarities between developmental states The state of development of an ontology is an important criterion for the availability evaluation. Currently it can not be a clear quality indicator, as the Semantic Web and the published ontologies are usually results of academia projects which imply additional consolidation in order to become usable in real world scenarios. However, if updates are relevant for the target application scenario, an unstable release (just as dynamically evolving ontologies) might lead to additional efforts to maintain the target ontology in accordance to its initial sources.

In our metadata ontology we model developmental states in a very simple way, as fixed instances. We distinguish between `alpha`, `beta` and `stable` releases, which are considered similar in a same way the assigned these measures to the automatization level of engineering methods in Section 6.1.4.

Similarities between availability states As the metadata model distinguishes between two states of this variable, naming freely available ontologies, and those which can be used only under license constraints, the similarity is defined dichotomously:

$$\begin{aligned} sim_{availability}(O_1, O_2) &= 1 \; if \; both \; ontologies \\ &\quad\; have \; the \; same \; availability \; state, \; and \\ &= 0 \; otherwise. \end{aligned} \tag{6.10}$$

Ranking Function

Every metadata-described ontology O can be interpreted as a vector of features

$$\overrightarrow{O} = O_1 \ldots O_i \tag{6.11}$$

The rank of an ontology with respect to its reusability in the actual reuse context can be calculated as

$$rank(O, \overrightarrow{q}) = \frac{\sum w_i * sim(O_i, F_i)}{\sum w_i} \tag{6.12}$$

The function $sim(O_i, F_i)$ is intended to compute partial feature-oriented relatedness. As every query feature F_i can be associated to multiple expected values (connected by default disjunctively), we differentiate in equation 6.12 between two cases:

- all values of the feature F_i are covered by a particular ontology (and the corresponding feature Q_i). Hence we define the similarity between the query and the ontology feature to be the maximal similarity among the individual values:

$$sim_{jk}(O_i, F_i) = max\ \{sim(O_{ij}, F_{ik})\} \tag{6.13}$$

where j and k range over the values of the corresponding features.

- otherwise we define this function as an average measure.

6.1.5 Result Selection

The results of the ranking are presented to the ontology evaluator in form of a list of ontologies, together with the obtained scores and the available metadata. The evaluator can revise the outcomes of the method and in case they are unsatisfactory, re-iterate the process at the feature selection step.

Note that the introduction of method-internal weights does not have any impact on the overall recall. By this means we are potentially causing changes in the ranking of the results without a deprecation of the total hit list.

To summarize in this section we demonstrated how ontology metadata can be used to operationalize the ontology usability evaluation. We described a method which makes intensive use of background knowledge on the inherent contextual dependencies influencing the success of a reuse process. From a technical point of view the method is based on a vector model which is evaluated on an existing ontology metadata repository using semantic means. Details on the implementation of the method are given in the next chapter. Chapter 8 discusses the evaluation of the method and the implications of the evaluation outcomes.

6.2 Ontology Merging and Integration Method

This section addresses the usage of contextual information about ontology reuse processes as an optimization factor for ontology merging/integration. In this step an ontology reuse endeavor is finalized by customizing the positively evaluated ontological sources and integrating the revised versions in a broader ontology engineering setting (cf. Chapter 4). From a methodological perspective our ontology reuse methodology recommended the following workflow:

1. **Ontology customization**: execute basic customization operations as foreseen during the previous reuse stages.

2. **Tool and method selection**: specify tools and integration strategy.

3. **Integration preparation**: specify the concrete integration workflow.

4. **Integration execution**: execute integration in the pre-defined way.

5. **Integration evaluation**: evaluate integration results.

The feasibility study revealed that the most challenging and time-consuming task at this stage is the discovery and selection of the tools and methods which can be adequately utilized to perform the merging and integration of multiple sources. This task is currently completed manually and assumes the availability of additional expert knowledge. Ontology engineers are required to possess expertise with respect to ontology matching methods in order to select an appropriate merging/integration strategy. Programmers should complement this work with the capability to configure and use the implemented algorithms. If these pre-conditions are satisfied, the engineering team should compare the functionality of various matching services with the particularities of the ontologies to be processed and to rank the results of this comparison. The selection process implicitly builds upon a series of dependencies between service capabilities to properties of the application context in which the integration takes place.

The remaining tasks do impose further limitations:

- **Ontology customization**: summarizes a series of transformations of the original ontologies w.r.t. their content, representation language or natural language articulation. These tasks are intended to be accomplished automatically and first tools for this purpose are already available, though their quality needs further improvement.
- **Integration preparation**: primarily implies configuration operations performed by humans on the tools and methods to be applied. These tasks can not be further optimized at method level.
- **Integration execution**: stays for the concrete operation of the tools and the collection of the results. Again, this task is not subject to further optimizations.
- **Integration evaluation**: the integration evaluation can the fundamentally understood as a technical check whether the integration execution has produced the expected results. Note that this is not the same as the evaluation of the ontology reuse outcomes, which is a topic situated outside of the scope of this thesis. The evaluation of the target ontology is intended to be based upon the requirements specified in the ontology requirements specification document and be performed using the methods recorded there. Approaches coping with this different type of evaluation are introduced for example in [19, 85, 114, 203].

We argue that an explicit representation of the context-driven optimizations and the automatic execution of the emerging dependency rules can be a prolific first step towards the operationalization of this non-trivial task within and beyond ontology reuse processes. The remainder of this section elaborates on these preliminary considerations. We give an overview of a metadata-based method for selecting tools and methods for ontology integration, which was co-developed by the author as backbone for the MOMA framework [148, 149, 150].[9]

6.2.1 Method Overview

The method utilizes additional information about ontologies (*ontology metadata*) and available matching services (*matcher metadata*) in order to determine which of the latter are appropriate in a given application context. The ontology metadata captures information about

[9] `http://moma.ag-nbi.de/` last visited in June, 2006

matching-relevant ontology features such as the size of the model or the language used for labeling ontological primitives. In turn the matcher metadata describes the most important characteristics of the matching services: input and output parameters, applied heuristics etc. Service providers are expected to provide the required descriptive metadata for the subscribed resources, as this guarantees a higher visibility of their products in respect to incoming inquiries. Ontology metadata is specified at run-time, as ontology engineers aiming at discovering methods or tools appropriate for their purposes define their preferences in this form. A rule repository comprises dependency rules linking service capabilities to contextual parameters. The rules are triggered automatically in order to speed-up the selection process.

From a process-driven perspective the integration method consists of four steps (cf. Figure 6.3):

1. **Context specification**: in this step the ontology engineer specifies those characteristics of the application setting which are relevant to the integration task. The context is particularly concerned with the ontologies involved in the process.

2. **Feature selection**: after specifying the context the user is expected to provide details about the integration process itself, in terms of inputs, outputs, as well as quality of service criteria.

3. **Usability computation**: the criteria specification is translated to a query which is executed on the knowledge base containing matching metadata and rules.

4. **Result assessment**: the user selects the appropriate method or tool from the computed hit list.

6.2.2 Context Specification

The main objective of this first step is to clarify the needs of the merging and integration task with respect to the methods and tools to be applied. The user is expected to provide information about the context of the actual task. At this level the context is primarily limited to information about the input ontologies and the expected outputs, but it can be extended to further issues such as non-functional quality of service criteria or costs.

Context Ontology

We differentiate between two types of contextual information for merging and integration purposes. Consequently the context model is confined to (cf. [166]):

- **Task-related information**: includes information about the methods and tools applied to perform the process. We designed a metadata model capturing this type of information in a controlled and formal way.

- **Target-related information**: strictly relates to integration-relevant features of the processed ontologies. For the representation of this information we resort to our ontology metadata model.

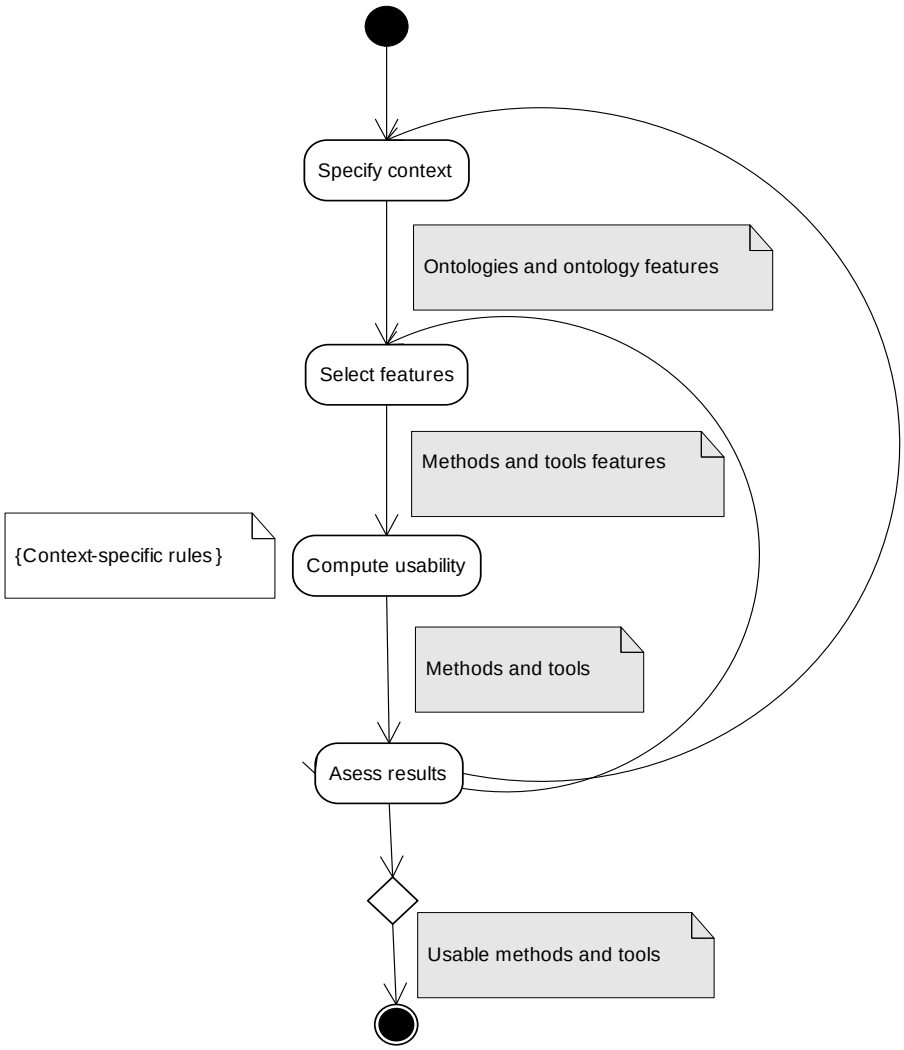


Figure 6.3: Ontology Merging and Integration Method

This model can be refined with additional environment-related information, with the objective of inducing an application-specific behavior to the integration strategy. These aspects are, however, not covered by our method so far. They are further discussed within the MOMA framework by Mochol in [148, 150].

The task-related contextual information is extracted from a matching metadata model, which describes fundamental ontology merging and integration services. The core of the model refers to matching algorithms, as these are a essential pre-requisite for the operation of this reuse step. The way we classified the matching algorithms strongly relies on [184] (see Fig. 6.4).

[184] distinguishes between *individual matchers* which compute a mapping based on a single matching criterion and *combining matchers* which use multiple individual matchers [149]:

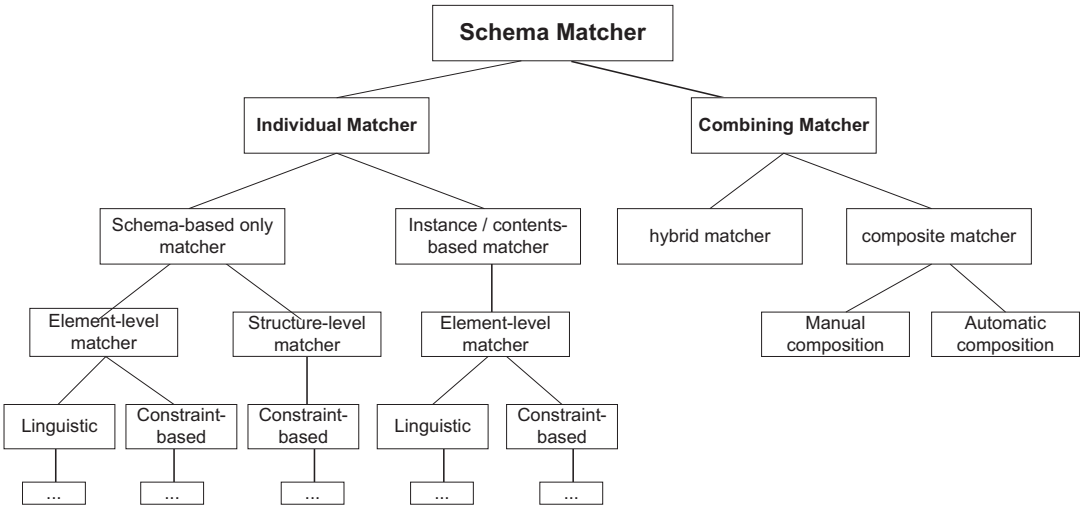


Figure 6.4: Taxonomy of Schema Matching Approaches cf. [184]

- **Individual matchers**: can work on instance data (*instance/contents-based matchers*) or consider only structure information, be that relationship types, datatypes and schema structures (*schema-only based matchers*). Both algorithms can be applied on individual schema elements such as attributes or concept labels (*element-level matchers*). In addition, schema-only based approaches can deal with combinations of these schema elements such as complex schema structures, thus computing mappings by analyzing subgraphs (*structure-level matchers*). A single element-level matcher uses linguistic, as well as constraint-based techniques, while a schema-only based matcher is considered to use only the latter.

- **Combining matchers**: are divided into two categories. *Composite matchers* combine the different results of independently executed matchers whereupon the order of the execution of the individual matchers can be assign manually (*manual composition*) or (semi-)automatically (*automatic composition*). By contrast, a *hybrid matcher* does not allow such manual intervention.

Beside the aforementioned classification, the matching metadata distinguishes among different *matching results* (mappings, value) and includes several *matching characteristics*:

- **Input type**: instances or schemas, numerical values.

- **Cardinality**: specifies whether a matcher compares one or more elements of one schema with one or more elements of another schema; we differentiate between global cardinality (w.r.t different mapping elements) and local cardinality (w.r.t individual mapping elements).

- **Matching level**: *atomic level*, e.g., attributes in an XML schema and *higher (non-atomic) level* e.g., XML elements,

- **Completeness**: a *full match* considers all elements of the two schemas, in contrast to a *partially match*.

Further on, properties of the ontologies to be matched such as type, formality level, domain type, representation languages, supported natural language, supported used primitives etc. are defined in the ontology metadata model and are referenced in the matching metadata model to refine the description of the matching inputs.

The matcher classification supplemented by the aforementioned features was conceptualized in form of an ontology and implemented in OWL. Matcher types are defined as OWL classes within a hierarchical structure with specialization/generalization relationships between them. By means of OWL constraints we specified the characteristics of each type of matching algorithm (for example that the input of an instance-based matcher can not be a schema without instance data).

Matching algorithms can not be applied with the same success expectations independently of all the dimensions of the mentioned ontology information model. In particular, we identified ontology features which are relevant for matching tasks:

- **Syntactic features**: such as number of specific ontological primitives influence the matching execution performance and the quality of the structured-based matchers, which usually perform better on simple graph structures. Further on, the implementation language might impose further constraints on the selection of tools, as most of them are explicitly built for a set of languages.
- **Semantic features**: such as
  - **Readability** (i.e. the usage of human-readable concept names)
  - level of formality (e.g., highly informal, semi-informal, semi-formal, rigorously formal [223]).
  - **Type of model** (upper-level, domain ontology, taxonomy, thesaurus etc.)
  - **Ontology domain** (i.e. the domain modeled by the ontology, e.g., medicine)
  - **Ontology representation paradigm** (i.e. the class of representation languages with respect to expressivity)
  - **Ontology natural language** (i.e. the natural language used in denominating ontological primitives e.g., English)

Figures 6.5 and 6.6 show an excerpt of the matching ontology which captures information about the most significant approaches in the contemporary literature in the field classified in individual and combined algorithms.[10]

Impact of Context on the Usability Assessment

For a given pair of ontologies to be merged, the matching engine has to decide which matching algorithms can be applied to obtain the desired outputs. The engine is aware of background information describing the available matching services and the properties of the input

[10]The screenshots show the matching ontology in the Protégé ontology engineering environment.

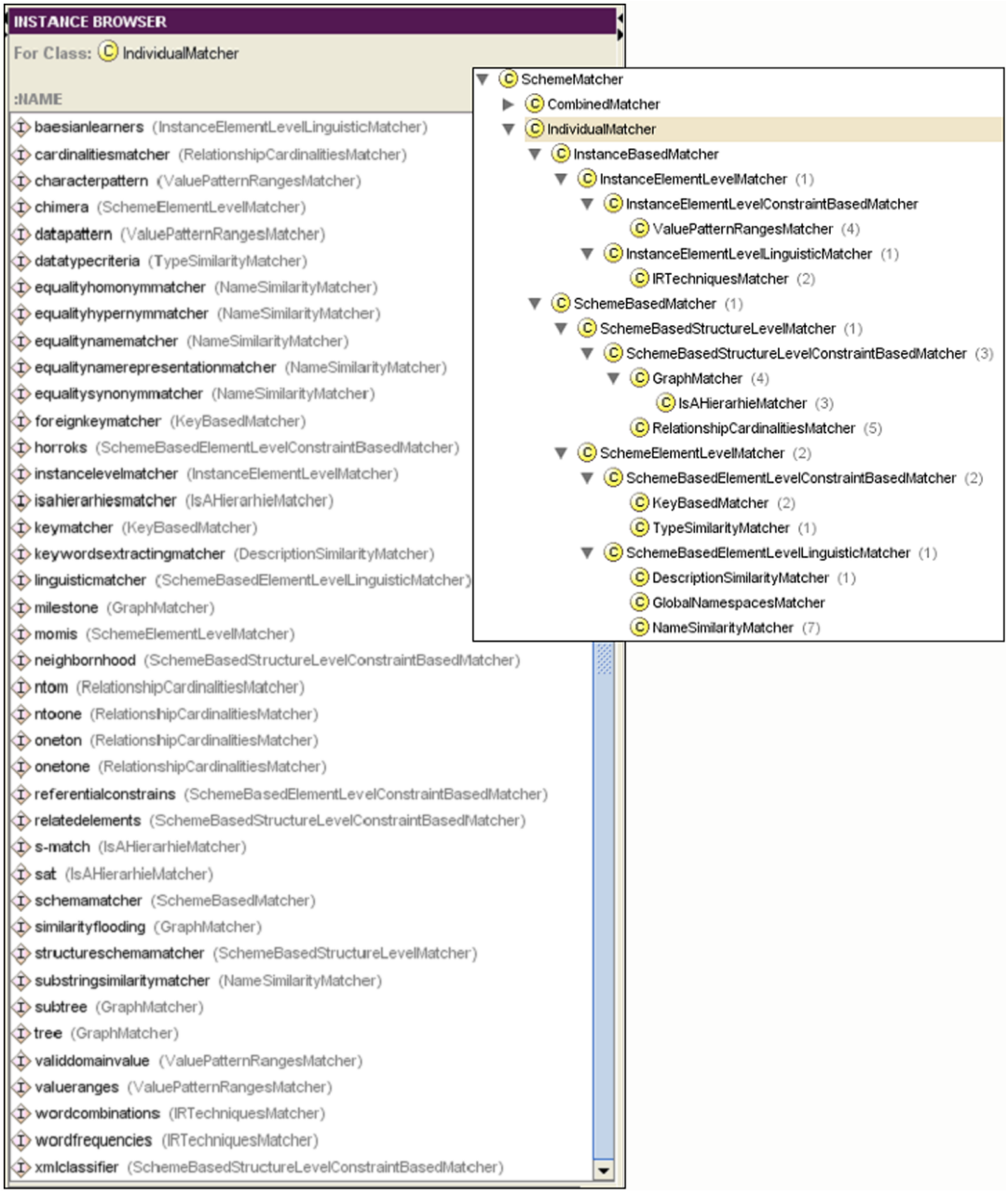


Figure 6.5: Classification of Individual Matchers with Some Instances

ontologies. However, in order to automatically infer which algorithms are suitable to these inputs, it needs explicit knowledge about the dependencies between them and the structures they operate on. We formalize this knowledge in terms of *generic dependency rules*—statements that determine which elements (in this case which matchers) are to be used or excluded:

- Apply only instance matchers for a single ontology.

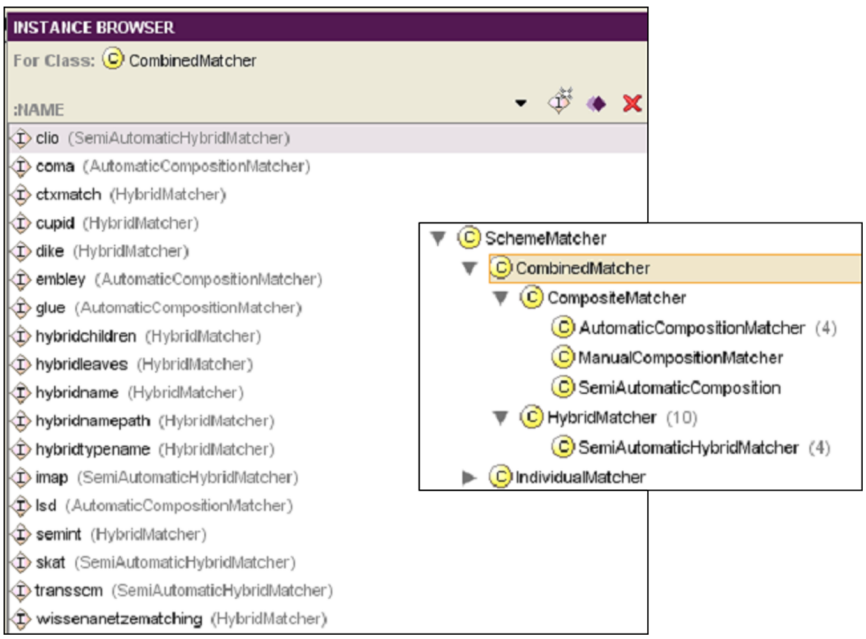


Figure 6.6: Classification of Combined Matchers with Some Instances

- Apply only matchers which are able to deal with the representation language of the inputs.
- Use only linguistic matchers for informal and semi-formal ontologies.
- Use structure-based matchers for ontologies with different natural languages.
- Use constraints-based matchers only for formal ontologies and only if ontologies contain axioms.
- Match upper-level to domain ontologies using linguistic matchings.
- Match only ontologies in similar domains.
- Apply only schema matchers if no instance data is available.
- Do not apply linguistic matchers for ontologies with incompatible concept names.

These rules are the result of analytical studies and literature research and were applied within the projects "KnowledgeNets" and "A Semantic Web for Pathology" [13, 151, 173], which required ontology matching techniques to merge and integrate existing ontologies in different Semantic Web application settings. A discussion of the empirical findings of these two experiments is provided in [150]. Further rules, especially relating the syntactic features of ontologies with specific performance and accuracy parameters, are addressed in detail in the context of the MOMA framework beyond the scope of this thesis.[11]

[11] `http://moma.ag-nbi.de` last visited in June, 2006

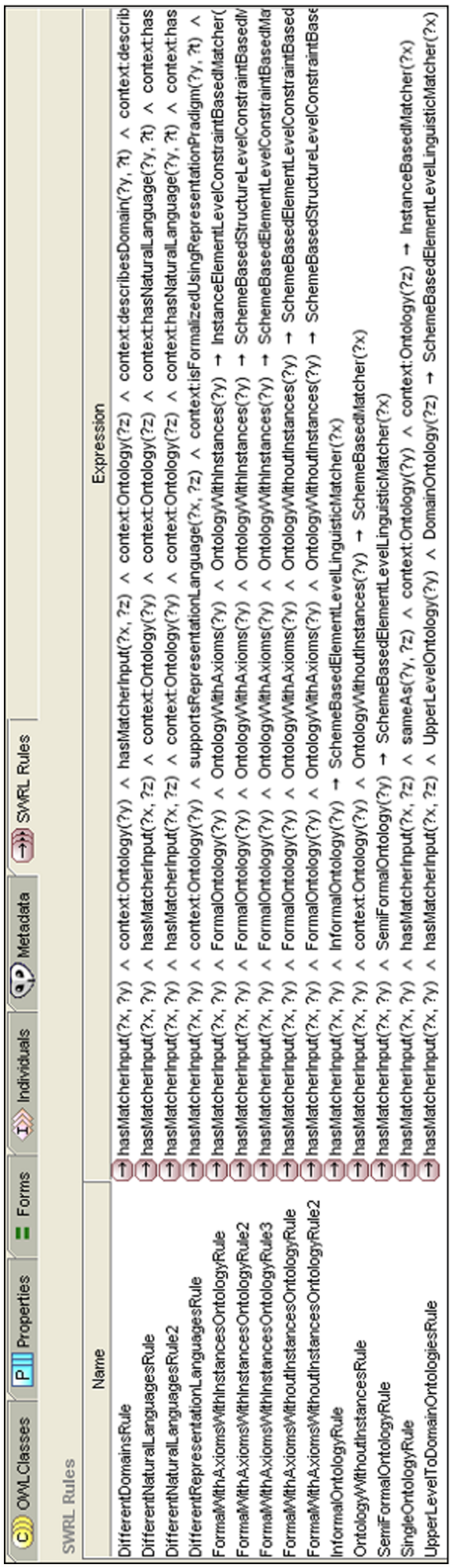


Figure 6.7: Implementation of Contextual Dependencies in Protégé

Formalizing Context Dependencies

The matching rules were implemented in SWRL [100], a rule language for the Semantic Web, which allows us to formalize them in terms of the concepts defined in the two metadata models. The implementation was carried out in Protégé, using a dedicated plug in. An overview of the rules is depicted in Figure 6.7. Appendix C provides some examples of contextual dependency rules serialized to RDF/XML, while a complete description of the implementation is provided in [149].

6.2.3 Usability Computation

The usage of the SWRL rules in decision making processes requires a reasoning engine which is able to operate on OWL ontologies and SWRL rules—an issue which is still subject of active research in the Semantic Web community. We analyzed several tools which address this topic [141, 171] and, for proof-of-concept purposes, employed the SWRL rule engine developed by Jing Mei at the Free University of Berlin, Networked Information Systems.[12] This tool provides reasoning over RDF Schema taxonomies plus SWRL rules and is built upon the Sesame repository [140, 142]. In a first step the ontologies (describing ontologies and ontology management services) have been stored (as RDFS) to the Sesame repository. Further on, we proceeded with the provision of SWRL dependency rules, aiming to restrict the number of potentially usable matching candidates. Once the entire knowledge base was stored to Sesame, we were able to carry out the method and tool selection task. The query specified the features of the input ontologies, while the SWRL engine was responsible for generating the results in accordance to the formalized restrictions.

The results of the rule-based query processing could be subject to further ranking operations aiming to capture the degree to which a particular matching service fulfills the user-defined goal. This issue is not addressed in more detail in this thesis, whose focus is on the reusability of ontologies more than service matchmaking.

6.2.4 Result Selection

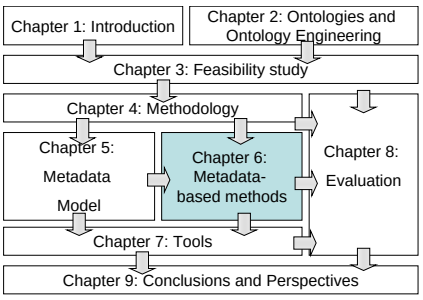
The engineering team analyzes the results delivered by the rule engine and selects the methods and tools which suit their goals. If the suggestions of the tool and method selection are not considered adequate, the process can be re-iterated at step 1, so that the criteria which should be satisfied by the integration services are adjusted. The current implementation of the service selection method clearly concentrates on the elimination of unappropriate matching candidates without studying in more depth means to rank the ones which could be applicable to this task. These issues are out of the scope of this thesis and are central to the research carried out in the MOMA framework.

In summary, the second part of this chapter sketched a high-level solution of the problem of choosing the right merging and integration strategy within the scope of ontology reuse. The method presented here does clearly not qualify for a complete account for the service matchmaking question, but rather for a proof-of-concept of the hypothesis that context aspects

[12] `http://www.ag-nbi.de/research/swrlengine/` last visited in June, 2006

play a non-negligible role in the quality and the success of various stages of ontology reuse. These preliminary ideas will be pursued by Mochol in the scope of the MOMA framework [148].

6.3 Summary



In this chapter we showed two methods which contribute to the operationalization of specific ontology reuse activities. We described a metadata-based ontology evaluation method that takes into account the impact of contextual dependencies in aiding ontology developers to decide upon application-relevant ontological sources. We further addressed the question of identifying appropriate tools and techniques for the execution of ontology merging and integration. The implementation and evaluation of these methods are discussed in the next chapters.