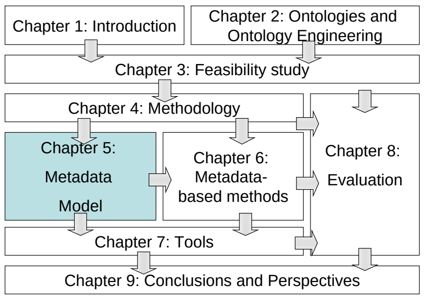
5 Ontology Reuse Metadata Model


Chapter 1: Introduction
Chapter 2: Ontologies and Ontology Engineering
Chapter 3: Feasibility study
Chapter 4: Methodology
Chapter 5: Metadata Model
Chapter 6: Metadata-based methods
Chapter 8: Evaluation
Chapter 7: Tools
Chapter 9: Conclusions and Perspectives

In this chapter we concentrate on the description of the conceptual model used to represent contextual information about Web ontologies and about the process of reusing them in new application scenarios. Section 5.1 describes the methodology we applied to create the model. Section 5.2 presents its main components from a conceptual and an implementation point of view. The second part of the chapter is dedicated to the design of methods enabling the deployment of the proposed metadata model within various activities of an ontology reuse process. Section 5.3 briefly approaches how this reuse-related information can be created and managed automatically and manually. Some of these ideas are prototypically implemented within the PROMI framework (Chapter 7). We conclude with a short summarization of the contents of the chapter in Section 5.4.
References*: This chapter is based on the publications [94, 168, 170].*

5.1 Model Engineering

In order to ensure a rich, but formal representation of the ambiguous semantics of the ontology metadata information and to enable the integration and exchange of contextual behavior within and across Semantic Web applications we represented this information in form of a *Semantic Web ontology*.

The ontology was developed in accordance with established ontology engineering methodologies [80], while the empirical findings acquired during the feasibility study (cf. Chapter 3) were the main input for the elaboration of the competency questions underlying the conceptual model.

Building an ontology is realized according to a pre-defined workflow (which is similar to that depicted in Figure 2.2 in Chapter 2). A team consisting of experts in the domain of interest and ontology engineers come together to design, implement and evaluate an ontology. After specifying the requirements and designing the conceptual model, this model is implemented in a specific representation language fitting the application requirements and the expressivity constraints induced by the conceptualization. The preliminary ontology is evaluated against the specified requirements and evolves in conformity with the needs of its users. Optionally, the ontology might be instantiated/populated, which means that application data is introduced to the ontology in form of instances. We describe each of these process step for the particular case of developing an ontology to capture reuse-relevant information about Semantic Web ontologies (i.e. a meta-ontology).[1]

5.1.1 Requirements Analysis

In this step we elaborated an inventory of requirements for the metadata model as a result of a systematic survey of the state of the art in the area of ontology reuse with the purpose to identify the real-world needs of the ontology engineering community with respect to a descriptive metadata format for ontologies. Further on, the domain analysis phase was complemented by a comparative study of existing (ontology-independent) metadata models and of tools such as ontology repositories and libraries making use of alike information (cf. Sections 1.4.2 and 2.2.2). A third source of information was the requirements specification associated with our ontology reuse methodology (cf. Chapter 4.5).

The domain analysis phase consisted of the following activities:

1. **Analysis of the domain to be modelled**: the meta-ontology should allow a general-purpose and in the same time comprehensive description of Semantic Web ontologies. It should notably cover information which is required as input in the activities recorded by our ontology reuse methodology:
   a) **Ontology discovery**: in this step the engineering team uses core metadata information related to the domain of the ontology, its availability or its creation date.
   b) **Ontology evaluation**: the evaluation of the content makes use of information related to the domain, the main concepts modelled within an ontology or the number of ontological primitives. Further on, the technical evaluation benefits from the availability of information related to the size, the structure and the development of the ontology. The knowledge representation evaluation concentrates on the language in which the ontology is formalized and its semantics. The application evaluation relates to more pragmatic aspects such as the settings in which the ontology was deployed before or the scope it was originally built for. Finally, the availability dimension is interested in information about the costs, the licence policies, the development status and the maintenance of the ontology.
   c) **Ontology merging and integration**: this technological step foresees the usage of diverse syntactic and semantic information about ontologies to optimize the selection of appropriate tools and methods. Graph-oriented metrics such as the total size of the ontology, the number of classes, properties, axioms and instances are correlated with the performance and scalability of the merging process. Further on, some tools are explicitly targeted at a restricted set of ontological primitives or at a particular input language.
2. **Identification of knowledge sources**: as knowledge sources relevant for the realization of the ontology we identified previous context and metadata models described in

[1]The evaluation step is briefly mentioned in this chapter for the sake of completeness. It will be thoroughly addressed in Chapter 8.

various literature studies (cf. Section 1.4). The Dublin Core metadata can be utilized for the description of facts which are not ontology-specific, such as the name of the ontology document or authorship information.[2] In the same time the FOAF model could form the basis for the description of persons and organizations involved in ontology engineering tasks.[3] A general-purpose taxonomy of topics such as the Open Directory can be re-used to provide a detailed, and in the same standardized inventory of ontology domains and their modeler's views. For the elaboration of a taxonomy of scopes and purposes and for the classification of typical ontology based applications we used available studies in the literature [115, 215], as well as the application scenarios accounted by our methodology. Classifications such as the North American Industry Classification System NAICS[4] or software taxonomies like the TROVE Software Map[5] might be useful if a more detailed description of ontology applications in terms of economical areas and functionality is required.

3. **Specification of domain and usability requirements**: The prospected model should capture ontology-related information which has been specified during the requirements analysis operated in relation to our methodology (cf. Section 4.5). Beyond this context, our focus was on the wide scale usability of the outcomes. This requirement states for the necessity of building a metadata model which allows proprietary extensions and refinements for particular application scenarios. From a content perspective, usability can be maximized by taking into account multiple metadata types, which correspond to specific viewpoints on the ontological resources, and are applied in various application tasks. Despite the broad understanding of the metadata concept and the use cases associated to each definition, several key aspects of metadata information are already established across computer science disciplines [153]:

   Structural metadata relates to statistical measures on the (graph) structure underlying an ontology.

   Descriptive metadata relates to the domain modelled in the ontology in form of keywords, topic classifications, textual descriptions of the ontology contents etc.

   Administrative metadata provides information to help managing ontologies, such as when and how it was created, rights management, file format and other technical information.

   Complementary to this metadata-specific classification, the features resulting from the cases study analysis as influencing the reusability of current Semantic Web ontologies can be classified from a content-oriented perspective in *syntactic*, *semantic* and *pragmatic* as follows [168, 170]:

   Syntactic features offer quantitative and qualitative information about the external form/syntax of the ontology and its underlying (graph) topology [39, 127]. Examples of syntactic features include the number of concepts and properties for

[2] `http://dublincore.org/` last visited in December, 2005
[3] `http://xmlns.com/foaf/0.1/` last visited in May, 2006
[4] `www.census.gov/epcd/www/naics.html` last visited in September, 2005
[5] `http://sf.hti.bfh.ch/softwaremap/trove_list.php` last visited in May, 2006

each class, the depth of an inheritance tree, the number of incoming properties, the number of concept instances, the average path length, the number of connected components. Since ontologies are published in an open network like the Semantic Web, it is also important to consider the links a particular ontology has to other networked information sources [155]. Finally, there is qualitative, representation language-dependent information like the representation language itself, the number and the type of syntax constructs used and syntactic correctness (validity).

Semantic features are related to the formal semantics of the representation language and the meaning of the ontology content:

- consistency (i.e. as measured by a reasoner),
- correctness (i.e. whether the asserted information is true as measured by human judgement),
- readability (i.e. the non-ambiguous interpretation of the meaning of the concept names with respect to a lexicon, the usage of human-readable concept names)
- level of formality (e.g., highly informal, semi-informal, semi-formal, rigorously formal as classified by [223]).
- type of model regarding the generality of the ontological content or the level of formality of the ontology.
- ontology domain (i.e. the domain modelled by the ontology, e.g., biology)
- view upon the modelled domain (i.e. medicine from the point of view of health insurances vs. medicine from an academic perspective)
- ontology representation paradigm (i.e. the class of representation languages with respect to expressivity e.g., a specific Description Logic)
- ontology representation language (i.e. the language in which an ontology is implemented)
- ontology natural language (i.e. the natural language used in denominating ontological primitives and for articulating comments and definitions e.g., Italian)

Pragmatic features are useful for managing or deploying the ontology. They refer to information about the history of the ontology, for example when, by whom and to which purpose it was developed, whether multiple versions are available or about the engineering process the ontology originally resulted from. The latter topic is relevant for ontology engineers intending to (partially) re-use the ontology within an information system: the original engineering methodology, tools used during the development process and the input information sources. As input information sources one can mention extern ontologies which are, partially or in a modified form, included to the current ontology. Another example is a domain-relevant document corpus used by ontology learning programs to generate the ontology.

The two classifications are complementary. The first one highlights the metadata character of the proposed model, while the semiotic classification accounts for the particular nature of the described objects i.e. Semantic Web ontologies which are intended to behave as meaning mediators between interacting agents.

5.1.2 Conceptualization

On the basis of the aforementioned analysis we designed the core structure of the metadata model in terms of classes and properties/attributes of these classes. Starting from the main class `Ontology`, the majority of its syntactic properties are conceptualized as attributes pointing to numerical values, while the remaining categories are modelled by means of auxiliary classes and pre-defined instances. Different types of ontologies as differentiated in various classifications in the ontology engineering literature (see for example the one in [138]) are introduced as sub-classes of the class `Ontology`, while their particularities are defined explicitly by means of property constraints.

The list of features capturing reuse-relevant information about Semantic Web ontologies can be ontologically conceptualized in at least two different ways. Following the tradition of metadata models, these features can be modelled as attribute-value pairs: the attributes are labeled by the names of the features and reference to numerical, boolean or string values:

- **Numerical values**: capture syntactic features of Semantic Web ontologies (e.g., the number of classes)
- **Boolean values**: can be used to capture qualitative syntactic and semantic features (e.g., the syntactic validity or the semantic consistency)
- **String values**: can be used to represent information about the remaining dimensions of the model (e.g., the formality level, the generality of the domain, the type of the ontology). In order to increase interoperability the usage of the values might be controlled by means of pre-defined vocabularies for each of the ontology features represented in this way.

However, the predominance of string-based metadata representation would contradict with the initial decision of modelling this type of information in ontological form, as the semantics of the corresponding concepts (i.e. the features of the described ontologies) is represented implicitly in the model and can be deduced only by humans interpreting the labels of the attributes and their values. By contrast, the usage of classes and instances for capturing ontology-related information provides the fundamental advantage of machine-understandability. The meaning of the metadata is represented explicitly in the model, which can be therefore used to compare and exchange ontologies with reduced semantic loss. As a consequence, the majority of ontological features introduced above were modelled as classes in the meta-ontology, while some of them were associated to a (closed) set of pre-defined instances.

Further on, every metadata entity was labeled in accordance to a pre-defined naming schema and carefully documented.

In order to increase the usability of the model with respect to its extendability we assigned the metadata entities to three usage categories (in the style of XML Schema):

- **Required**: These metadata facts are mandatory. Missing elements lead to incomplete metadata descriptions of ontologies and are handled accordingly by metadata management tools.

- **Optional**: The specification of optional metadata elements, though not mandatory, increases the reusability of the corresponding ontology.
- **Extensional**: This class of metadata elements is not represented in detail in the core model, but can be further elaborated in extension modules.

This distinction was only partially taken into consideration at conceptual level and is expected to be taken into account by metadata management tools in order to ensure the acquisition of a minimal set of metadata entries for Semantic Web ontologies. Whether a given metadata entry is mandatory or optional is explicitly represented by means of constraints on the corresponding ontological classes or properties.

5.1.3 Implementation

Due to the high accessibility and interoperability requirements, as well as the nature of the metadata—intended to describe Semantic Web ontologies—the conceptual model designed in the previous step was implemented in the OWL language. An implementation as XML-Schema or DTD was estimated to restrict the functionality of the ontology management tools using the metadata information (mainly in terms of retrieval capabilities) and to impede metadata exchange at semantic level. Further on, a language such as RDFS does not provide a means to distinguish between required and optional metadata properties. The implementation was performed manually by means of a common ontology editor.

Implementing the meta-ontology by means of the OWL language induced problematic modelling decision with respect to the separation/distinction between classes and instances. Due to limitations of OWL DL and to the character of the domain to be modelled (i.e. metadata on ontologies) the implementation is required to differentiate between two class-instance levels:

- the meta level which describes ontologies and their features. At this level the majority of the metadata entities can be interpreted as instances of a particular class. For example, when modelling ontology tasks, one can distinguish between the class `OntologyTask` and one of its instances (ontology-driven) `SemanticMatching`.
- the data level which describes concrete ontologies in terms of the schema. At this level, one creates an instance of the class `Ontology` and specifies the values of its slots, which are (at least in OWL DL) instances.

This distinction has proven to be non-trivial to follow within the implementation process, as some of the class instance separations were not intuitive. Further on, the lack of means to constrain datatype properties in OWL DL should be compensated by type checking routines at metadata management tool level.

5.1.4 Evaluation

The metadata model was evaluated by conducting structured interviews with a group of six experts from industry and academia. The evaluation framework complied to [79], who proposed a set of four general-purpose criteria for estimating the quality of ontological structures: *consistency*, *completeness*, *conciseness* and *extendability*. We added a fifth dimension, *readability*, accounting for the usage of intuitive labels to denominate metadata entities. We provide more details on the evaluation procedure, its outcomes and implications in Chapter 8.

5.1.5 Evolution

The real added value of the proposed metadata vocabulary is fundamentally determined by the representativeness of its content and its dissemination across the Semantic Web community. This aspect is addressed in our work in the context of the OMV (**O**ntology **M**etadata **V**ocabulary) project.

Within OMV the participant institutions initiated DEMO (**D**esign **E**nvironment for **M**etadata about **O**ntologies), whose main objective is the provision of an organizational and methodological setting for the evolution of the metadata ontology towards standardization [94]. For the realization of this goal DEMO takes advantage of the results already available in the ontology engineering community w.r.t collaboratively building ontologies in distributed environments. Based on the long-standing tradition of argumentation and conflict mediation research in knowledge management, agent-based computing or linguistics, approaches such as [26, 116, 181] provide a deep analysis of the challenges of such engineering settings, and means to operationalize it. For the collaborative refinement of the current release of the metadata ontology, the OMV consortium decided to apply the DILIGENT methodology [181], because, compared to alternative approaches, it provides a more fine-grained description of the underlying process model, whose validity has been tested in several case studies. The tools supporting the evolution activity are provided by the OntoWare platform.[6]

5.2 Metadata Ontology

In the following we present the core conceptual model and its realization in OWL in detail. For each of the main classes we give an overview of its properties, sub-classes and instances. We further exemplify the cases in which the mapping from the conceptual model to OWL (DL) was not straight forward. The implementation is available at `http://swpatho.ag-nbi.de/context/meta.owl`.

The first 3 levels of the is-a hierarchy underlying the metadata model contains the following classes:

- `owl:Thing`
  - `wikipedia:Application`
    - `OntologyApplication`
  - `dmoz:Topic`
  - `FormalityLevel`
  - `naics:IndustrialSector`
  - `MethodsMethodologiesTools`
    - `EngineeringMethodology`

[6] `http://omv.ontoware.org` last visited in May, 2006

    * Method
        · ConsistencyCheckingMethod
        · OntologyLearningMethod
        · OntologyMergingMethod
        · SyntacticalValidationMethod
        · KnowledgeRepresentationTranslationMethod
        · NaturalLanguageTranslationMethod
    * Tool
        · OntologyDevelopmentTool
        · ReasoningTool
        · ValidationTool
- OntologyType
- Person
- RepresentationLanguage
- RepresentationParadigm
    * DescriptionLogics
    * FrameLogics
- Role
    * PersonRole
        · DomainExpertRole
        · OntologyEngineerRole
        · OntologyUserRole
    * OntologyRole
        · VocabularyRole
        · IndexRole
        · ModelRole
        · FilterRole
- SemanticWebResource
    * GroupOfOntologicalPrimitives
        · ABox
        · TBox
    * OntologicalPrimitive
        · Concept
        · Instance
        · Property
        · Relation
        · Restriction
    * Ontology
        · Catalog
        · Classification
        · Glossary
        · SemanticWebOntology
        · Taxonomy
        · Thesaurus
        · ControlledVocabulary
    * WebDocument
- Task
    * OntologyDevelopment
    * OntologyLearning
    * OntologyMerging
    * OntologyTask
        · AnnotationTask
        · ConfigurationTask

· `FilteringTask`
· `IndexingTask`
· `IntegrationTask`
· `MatchingTask`
· `MediationTask`
· `PersonalizationTask`
· `QueryFormulationTask`
· `QueryRewritingTask`
· `SearchTask`

The implementation is divided into four modules, with the central module `meta` importing the remaining ones:

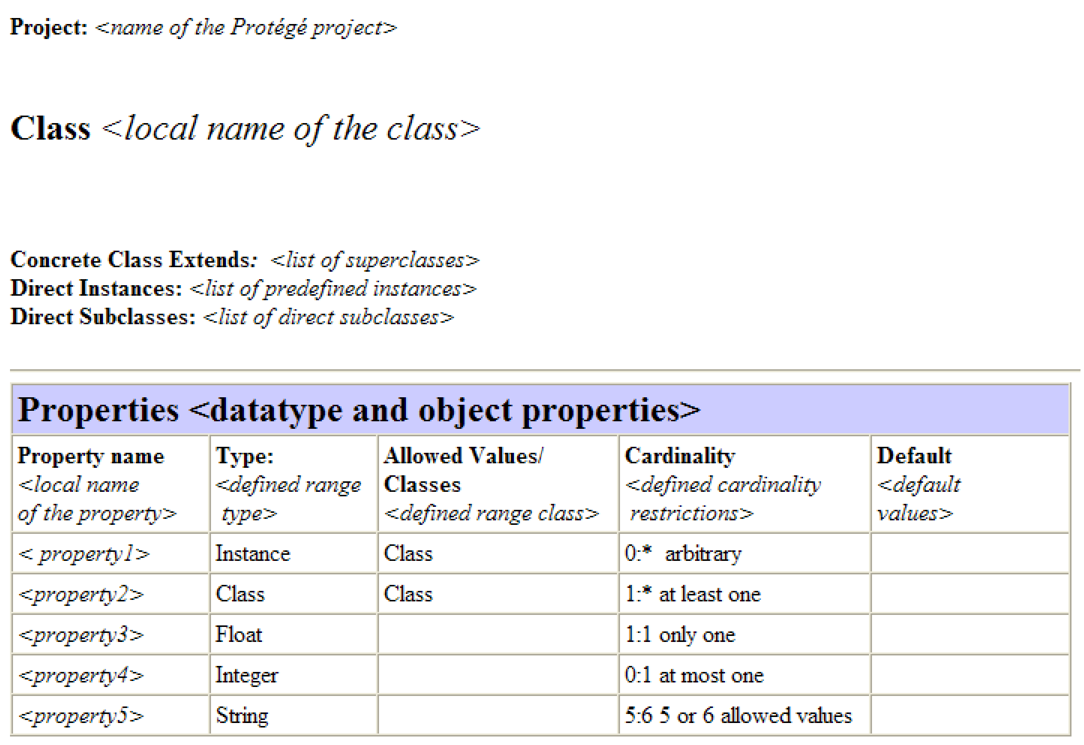
- `meta`: the core module of the ontology comprising the metadata entries.
- `wikipedia`: this module contains a classification of software applications as provided by the English Wikipedia in May, 2006. The classification does not claim for completeness or correctness, its main goal being to define a controlled framework in which these kind of items are described and to which metadata entries can uniformly refer.
- `dmoz`: the Open Directory topic hierarchy modelled as an OWL ontology. This is used for specifying the domain of the ontology and the view of the modelers upon this domain in a controlled manner.
- `naics`: self-developed OWL version of the NAICS standard on industrial sectors, which is used to describe application scenarios ontologies might be employed in.

In the following we describe the scope and the properties associated with each class. Every class is summarized using the subsequent documentation pattern (Figure 5.1).[7]

On the top the pattern we capture information about class at hand and its position in the specialization/generalization hierarchy. If no instances or sub-classes are available, the corresponding entries are missing. Per default the super-class of each domain-specific class is defined in OWL to be `owl:Thing`. The second half of the pattern is concerned with the datatype and object properties defined at the class. These are described in a 5-column table as follows:

- `property name`: the local name of a datatype or object property.
- `type`: the type of range defined. As per the syntax of the OWL language, object properties can be ranged at instances or classes. In the first case the values of a property are located at the set of instances of a given class. In the second class the value of the property is the class itself. The latter is not allowed in OWL DL, which differentiates between classes and instances. Datatype properties can be valued at common data types, as those specified by XML Schema. The example above includes strings, floats or integers, but other types are allowed as well.

[7]The documentation has been generated using the Protégé ontology engineering environment. For a given ontology it is possible to generate a class-centered Javadoc-style documentation which comprises basic information about the position of the class in the hierarchy and its main properties and their features. Refer to `http://protege.stanford.edu` last visited in August, 2006.



Project: *<name of the Protégé project>*

Class *<local name of the class>*

Concrete Class Extends: *<list of superclasses>*
Direct Instances: *<list of predefined instances>*
Direct Subclasses: *<list of direct subclasses>*

Properties <datatype and object properties>

Property name *<local name of the property>*	**Type:** *<defined range type>*	**Allowed Values/ Classes** *<defined range class>*	**Cardinality** *<defined cardinality restrictions>*	**Default** *<default values>*
< property1>	Instance	Class	0:* arbitrary	
<property2>	Class	Class	1:* at least one	
<property3>	Float		1:1 only one	
<property4>	Integer		0:1 at most one	
<property5>	String		5:6 5 or 6 allowed values	

Figure 5.1: Documentation Pattern for Ontology Classes

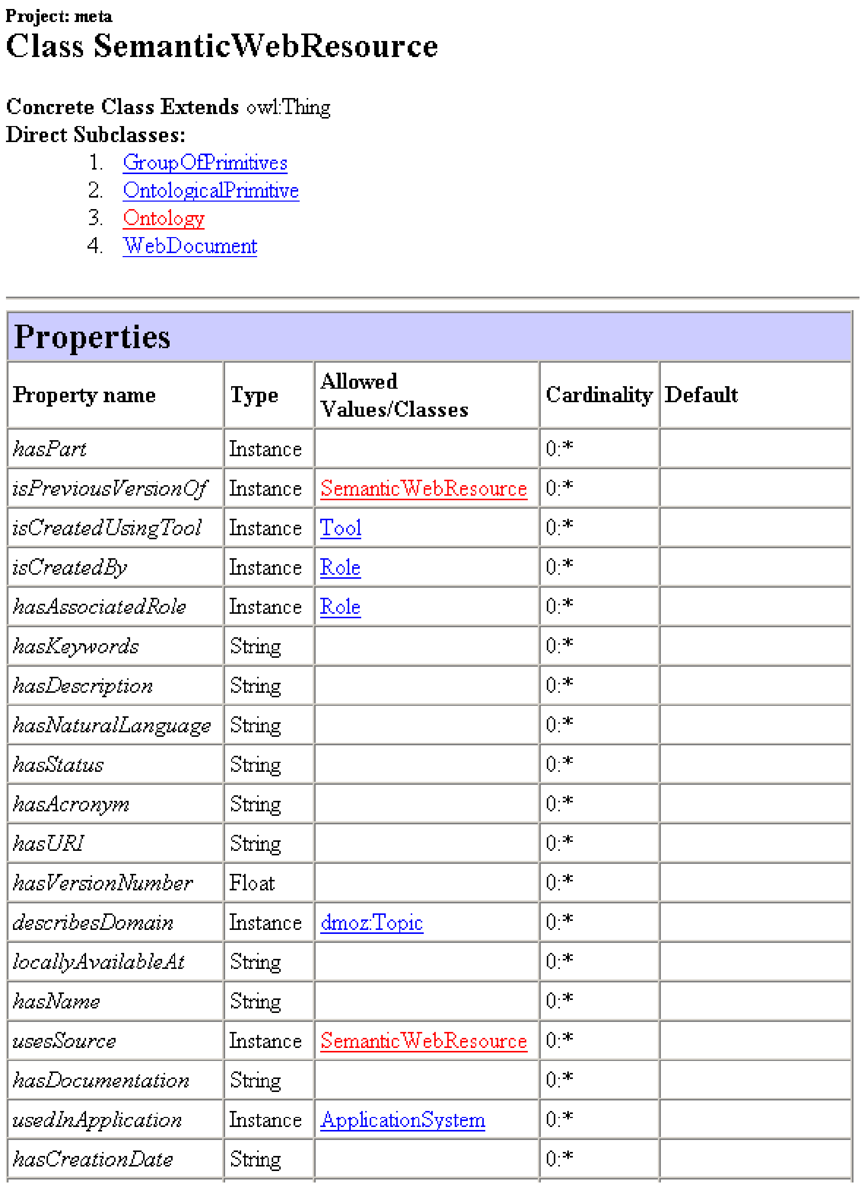
- `allowed classes/values`: in the column one specifies the concrete range class of an object property, no matter what the type of range is (i.e. class or instance).
- `cardinality`: the cardinality restrictions as allowed in OWL DL. The decision upon a particular restriction is reflected by the quality of a metadata entry of being "required" or "optional".
- `default`: default values for datatype or object properties.

A comprehensive example for the usage of the meta-ontology is provided in Appendix A.

5.2.1 Class `SemanticWebResource`

`SemanticWebResource` is the class representing all types of resources available on the Semantic Web, such as ontologies of various kinds, but also Web documents or fragments of ontologies. Ontologies and their parts (e.g., single ontological primitives and groups of primitives) are characterized by a set of features which are modelled by the remaining classes of the hierarchy. The associated properties are listed in Figure 5.2 and defined as follows:

- `hasName`: an informal name under which the resource is known.
- `hasURI`: the identifier of this Web resource.



Project: meta

Class SemanticWebResource

Concrete Class Extends owl:Thing

Direct Subclasses:

1. GroupOfPrimitives
2. OntologicalPrimitive
3. Ontology
4. WebDocument

Properties

Property name	Type	Allowed Values/Classes	Cardinality	Default
hasPart	Instance		0:*	
isPreviousVersionOf	Instance	SemanticWebResource	0:*	
isCreatedUsingTool	Instance	Tool	0:*	
isCreatedBy	Instance	Role	0:*	
hasAssociatedRole	Instance	Role	0:*	
hasKeywords	String		0:*	
hasDescription	String		0:*	
hasNaturalLanguage	String		0:*	
hasStatus	String		0:*	
hasAcronym	String		0:*	
hasURI	String		0:*	
hasVersionNumber	Float		0:*	
describesDomain	Instance	dmoz:Topic	0:*	
locallyAvailableAt	String		0:*	
hasName	String		0:*	
usesSource	Instance	SemanticWebResource	0:*	
hasDocumentation	String		0:*	
usedInApplication	Instance	ApplicationSystem	0:*	
hasCreationDate	String		0:*	

Figure 5.2: Overview of the Class `SemanticWebResource`

- `hasDocumentation`: a link to additional Semantic Web resources describing the actual resource.

- `hasPart`: generic property capturing information about the structure (as opposed to content) of the Semantic Web resource. Parts could be for instance sections of a Web document, or sub-graphs of a Semantic Web ontology.

- `isCreatedUsingTool`: the software used to generate the Semantic Web resource (e.g., text editors, ontology editors, translators etc.)

- `isCreatedBy`: authoring information. Authors are defined as roles referring to persons or organizations.
- `hasCreationDate`: the date when the resource has been created.
- `isPreviousVersionOf`: versioning information. The distinction between individual versions is not further specified and should be clarified at application level.
- `hasVersionNumber`: the number of the actual version of the resource, as specified by its authors.
- `hasStatus`: a string denoting the status of development of the current release of the resource.
- `usesSource`: external sources used as input for the generation of the Semantic Web resource at hand. For example a Web service might use an ontology to describe its contents, while an ontology might use various text files to extract domain-relevant ontological primitives.
- `isCompatibleWith`: those sources the ontology officially commits to. This applies in particular for standardized inputs. For example a business ontology might be compliant with the UNSPSC product classification.
- `describesDomain`: the topic of a Semantic Web resource in terms of a DMOZ category (see below).
- `usedInApplication`: instances of application systems using the Semantic Web resource for specific purposes. In case of ontologies this is further detailed within the dedicated class `Ontology`.
- `hasNaturalLanguage`: if the resource covers human-readable content, the natural language used for this purpose.

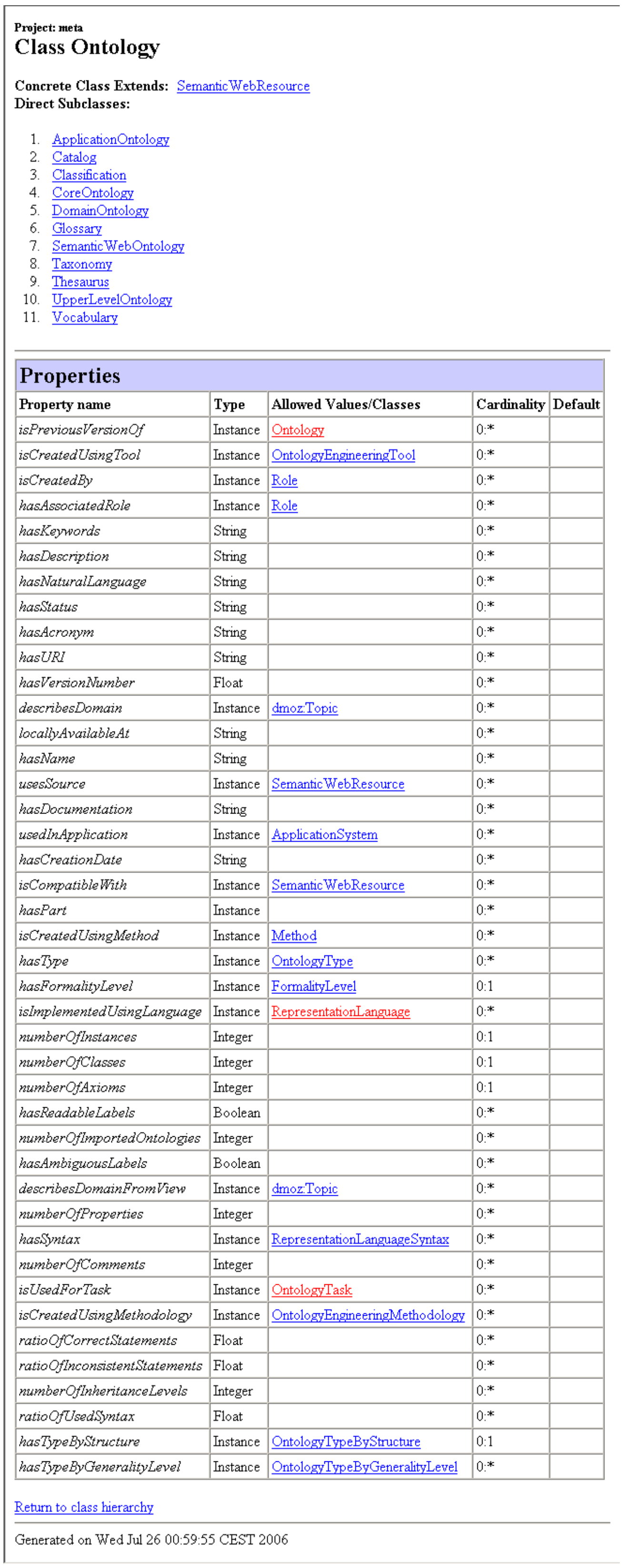
Except the `describesDomain` property referring to the subject addressed by a resource, the remaining properties capture pragmatic features of Semantic Web resources, which can be easily mapped to general-purpose metadata standards such as Dublin Core.

5.2.2 Class `Ontology`

`Ontology` is the class representing various types of ontologies currently available on the Web. Accounting for the ambiguous nature of the ontology concept the class unifies the various perspectives encountered in the ontology engineering literature of the last decades when defining this concept (see Chapter 2 for a discussion of this issue).

The subsequent datatype and object properties are defined for the class `Ontology`:

- `numberOf<*>`: this group of single-valued datatype properties capture graph-related metrics. The meaning of terms such as “class”, “instance” or “property” is defined through the OWL language.



Project: meta

Class Ontology

Concrete Class Extends: SemanticWebResource
Direct Subclasses:

1. ApplicationOntology
2. Catalog
3. Classification
4. CoreOntology
5. DomainOntology
6. Glossary
7. SemanticWebOntology
8. Taxonomy
9. Thesaurus
10. UpperLevelOntology
11. Vocabulary

Properties

Property name	Type	Allowed Values/Classes	Cardinality	Default
isPreviousVersionOf	Instance	Ontology	0:*	
isCreatedUsingTool	Instance	OntologyEngineeringTool	0:*	
isCreatedBy	Instance	Role	0:*	
hasAssociatedRole	Instance	Role	0:*	
hasKeywords	String		0:*	
hasDescription	String		0:*	
hasNaturalLanguage	String		0:*	
hasStatus	String		0:*	
hasAcronym	String		0:*	
hasURI	String		0:*	
hasVersionNumber	Float		0:*	
describesDomain	Instance	dmoz:Topic	0:*	
locallyAvailableAt	String		0:*	
hasName	String		0:*	
usesSource	Instance	SemanticWebResource	0:*	
hasDocumentation	String		0:*	
usedInApplication	Instance	ApplicationSystem	0:*	
hasCreationDate	String		0:*	
isCompatibleWith	Instance	SemanticWebResource	0:*	
hasPart	Instance		0:*	
isCreatedUsingMethod	Instance	Method	0:*	
hasType	Instance	OntologyType	0:*	
hasFormalityLevel	Instance	FormalityLevel	0:1	
isImplementedUsingLanguage	Instance	RepresentationLanguage	0:*	
numberOfInstances	Integer		0:1	
numberOfClasses	Integer		0:1	
numberOfAxioms	Integer		0:1	
hasReadableLabels	Boolean		0:*	
numberOfImportedOntologies	Integer		0:*	
hasAmbiguousLabels	Boolean		0:*	
describesDomainFromView	Instance	dmoz:Topic	0:*	
numberOfProperties	Integer		0:*	
hasSyntax	Instance	RepresentationLanguageSyntax	0:*	
numberOfComments	Integer		0:*	
isUsedForTask	Instance	OntologyTask	0:*	
isCreatedUsingMethodology	Instance	OntologyEngineeringMethodology	0:*	
ratioOfCorrectStatements	Float		0:*	
ratioOfInconsistentStatements	Float		0:*	
numberOfInheritanceLevels	Integer		0:*	
ratioOfUsedSyntax	Float		0:*	
hasTypeByStructure	Instance	OntologyTypeByStructure	0:1	
hasTypeByGeneralityLevel	Instance	OntologyTypeByGeneralityLevel	0:*	

Return to class hierarchy

Generated on Wed Jul 26 00:59:55 CEST 2006

Figure 5.3: Overview of the Class `Ontology`

- `describesDomainFromView`: this property complements the more generic `describesDomain` property defined at a higher level in the hierarchy with additional contextual information about the perspective of the modeler upon the topic of the ontology. For example, if the declared domain of an ontology is "cars" one can imagine different models covering different concepts for views such as "insurances", "supply industry" or "car drivers".

- `hasReadable/AmbiguousLabels` these dichotomous properties have the value 1 in case the natural language denominators of ontological primitives are human-readable (as stated by a lexicon, cf. Section 5.3) or have an unique meaning, respectively.

- `ratioOfUsedSyntax`: is a float value indicating how many syntactic constructs have been used in the implementation.

- `isImplementedUsingLanguage`: the language used for representing the knowledge, independently on any level of machine-understandable formality.

- `isCreatedUsingMethod/Methodology`: accounts for the various ways of creating ontologies (cf. class `MethodsMethodologiesAndTools` below).

- `hasType`: refers to a type of ontology, as described by the class `OntologyType`. Well-known types of ontologies are also pre-defined as sub-classes of the main class, while their meaning is refined using constraints. For example an informal taxonomy `hasPart` concepts and is-a relationships and `hasFormalityLevel` informal.

- `hasFormalityLevel`: refers to pre-defined formality levels explained below.

- `usedForTask`: the tasks an ontology is designed to be used for in a particular context.

More general-purpose properties are inherited from the class `SemanticWebResource`.

5.2.3 Class `OntologyType`

This class models a classification of Semantic Web ontologies with respect to the type of the represented domain and its generality and application-independence. As mentioned in Chapter 2 various, though overlapping classifications of ontologies w.r.t this dimension emerged in the last years in the literature. In this ontology we adopted the classification of [86] due to its popularity and compatibility with the majority of remaining proposals and modelled the four ontology types as instances of the class `OntologyType`: `application`, `task`, `domain` and `upperlevel`. Further intermediary types are `core` and `knowledgerepresentation` [80]. The meaning of the types employed has been introduced in Chapter 2:

- `upperlevel` ontologies describe general-purpose domains of interest, which can be roughly ordered to the discipline of philosophy.

- `domain` ontologies are used to model specific domains of reality.

- `core` ontologies are those ontologies which model the most important concepts within a concrete domain. These concepts are likely to be relevant for any application in that domain using ontologies.
- `task` ontologies focus on domain-specific processes, tasks or activities which take place in a particular domain of interest, be that abstract or concrete.
- `application` ontologies reflect, as the name says, the view of a particular application scenario upon the domain being modelled. This leads to specific modelling decisions, which are not directly related with the facts which hold true within the domain, but with the requirements of the application setting in which the ontology is used.
- `knowledgerepresentation` ontologies describe the core concepts of a formal representation paradigm and their properties.

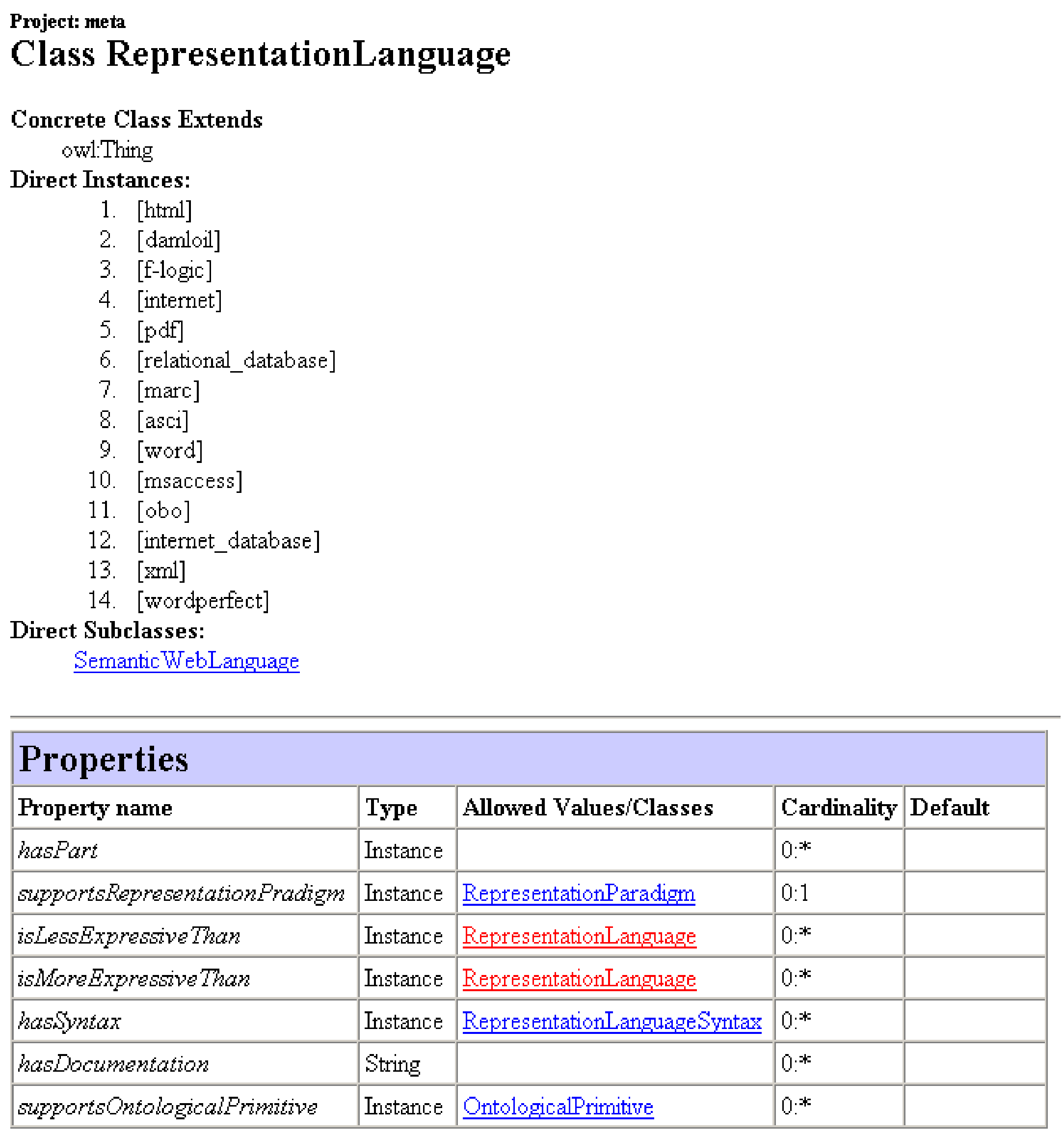
5.2.4 Class `FormalityLevel`

This class states for a classification of levels of formality compiled from the current state of the art in the area of Semantic Web ontologies. The metadata model resorts to the 4-level classification of [223]. The four formality levels are modelled as instances of the class `FormalityLevel`, which in turn is described extensively by this classification: `highly-informal`, `semi-informal`, `semi-formal` and `rigorously-formal`. Again, these formality levels have been previously defined in Chapter 2:

- an ontology can be considered `highly-informal` if it describes the domain of interest in a loose form in natural language, whose precise semantics can be captured by humans, at most.
- in `semi-informal` ontologies the meaning of the modelled entities is less ambiguous by the usage of a restricted language, which is still human-readable.
- by contrast, `semi-formal` ontologies make use of a formal knowledge representation language.
- in `rigorously-formal` ontologies the meaning of the representation language is defined in detail, with theorems and proofs for soundness or completeness, while the domain of interest is modelled precisely using properties and axioms.

5.2.5 Class `RepresentationLanguage`

Information about the language in which an ontology is implemented is captured by the class `RepresentationLanguage` (Figure 5.4). Every representation language is characterized by several syntactic alternatives (i.e. the class `Syntax`) and by a particular `RepresentationParadigm`. Expressivity relationships between languages are modelled in terms of properties such as `isLessExpressiveThan` and its counterpart `isMoreExpressiveThan`. Further on, one can provide detailed information about the constructors in knowledge representation language by indicating which `OntologicalPrimitives` are supported.



Project: meta

Class RepresentationLanguage

Concrete Class Extends
owl:Thing

Direct Instances:

1. [html]
2. [damloil]
3. [f-logic]
4. [internet]
5. [pdf]
6. [relational_database]
7. [marc]
8. [asci]
9. [word]
10. [msaccess]
11. [obo]
12. [internet_database]
13. [xml]
14. [wordperfect]

Direct Subclasses:
SemanticWebLanguage

Properties

Property name	Type	Allowed Values/Classes	Cardinality	Default
hasPart	Instance		0:*	
supportsRepresentationPradigm	Instance	RepresentationParadigm	0:1	
isLessExpressiveThan	Instance	RepresentationLanguage	0:*	
isMoreExpressiveThan	Instance	RepresentationLanguage	0:*	
hasSyntax	Instance	RepresentationLanguageSyntax	0:*	
hasDocumentation	String		0:*	
supportsOntologicalPrimitive	Instance	OntologicalPrimitive	0:*	

Figure 5.4: Overview of the Class `RepresentationLanguage`

5.2.6 Class `RepresentationParadigm`

Different knowledge representation language may share a common knowledge representation paradigm. This can be formally defined as a class of representation languages with similar properties. Prominent examples are Description Logics languages or Frames. The distinction between representation paradigms is important as ontologies implemented in languages belonging to different paradigms can not be directly translated from one language to another without semantic loss.

5.2.7 Class `dmoz:Topic`

The class `dmoz:Topic` relates to a standard taxonomy of general-purpose subjects, which can be used to specify the domain and the view upon this domain of an ontology. As classification structure we re-used the DMOZ taxonomy, which was customized and translated to OWL for this purpose.[8]

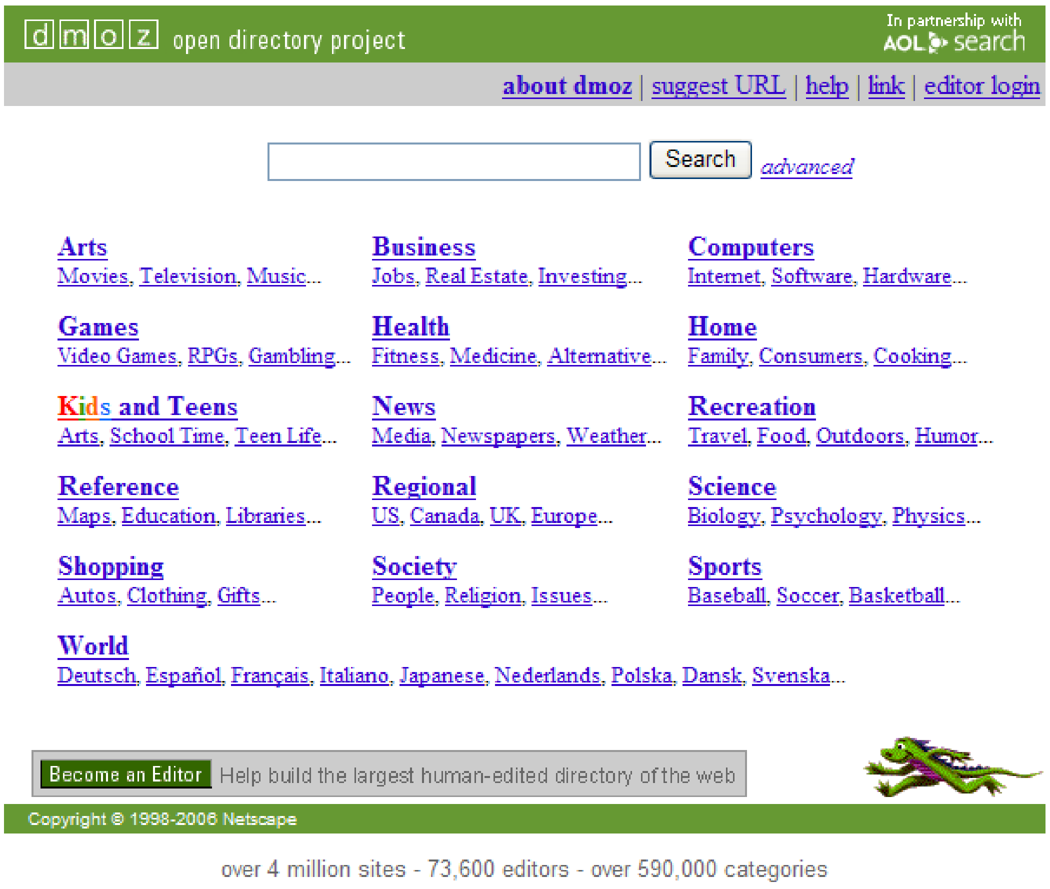
DMOZ is the most prominent human-edited directory of the Web. It is constructed and maintained by a vast community of volunteer editors within the scope of the Open Directory project and employed by a variety of search engines and other applications. The topics are organized in 14 content categories and two geographic ones, while individual terms may occur at several nodes in the taxonomical tree (cf. Figure 5.5). For example, the topic *"Software"* is listed under *"Business/Information technology"* and *"Computers"*. Linguistically this multiple inheritance structure provides a simple means to identify linguistically plurivalent topic labels. From a conceptual point of view the meaning of the topics multiply ordered in the taxonomy can be interpreted in a context-sensitive manner. Going back to the previous example, if an ontology is declared to model the domain *"Software"*, it might be useful to add context-related information to this entry in order to disambiguate its meaning. According to the DMOZ taxonomy *"Software"* can be interpreted in the context of *"Business"* or *"Computers"*. This additional information allows humans, but also machines using ontology metadata, to have a more clear understanding on the contents of the underlying ontology. An ontology modelling software-related concepts in business context is likely to contain concepts like *"Price"* or *"Licence"*, while the same domain in technical context will focus on concepts related to software components and functionality.

In our ontology metadata the core of the topic classification is the class `Topic` and the associated transitive property `hasSubTopic`, which is used to express taxonomical relationships. Individual nodes of the DMOZ tree are mapped to OWL instances in order to avoid the definition of a `describesDomain` property with a `Class` range (i.e. instead of an `Instance` range), which would imply a complexity at the level of OWL Full in the ontology. For the purpose of our metadata model, we re-used the first 5 levels of DMOZ, resulting in over 3,000 individual topics. The path information corresponding to each topic instance can be computed as the transitive closure of the `hasSubTopic` relationship. The context information is per default assigned to the direct super-topics of a certain topic. This means that for an instance `x`, all instances `y` which are connected to `x` by means of a `hasSubTopic` relationship ($\langle$y hasSubTopic x$\rangle$) can be interpreted as context information.

5.2.8 Class `wikipedia:Application`

The class `wikipedia:Application` is targeted at providing a classification of the most popular contemporary software applications. Due to the lack of representative software taxonomies freely available, we decided to prototypically re-use the one emerging in the Wikipedia collection, which is depicted below. We do not claim for the completeness or the unconfined correctness of the re-used classification. Its goals in relation to our work are two fold: firstly, it defines a vocabulary which can be used to control the metadata creation task;

[8]`http://dmoz.org/about.html` last visited in May, 2006


dmoz open directory project
In partnership with AOL search
about dmoz | suggest URL | help | link | editor login
Search
advanced

Arts
Movies, Television, Music...

Business
Jobs, Real Estate, Investing...

Computers
Internet, Software, Hardware...

Games
Video Games, RPGs, Gambling...

Health
Fitness, Medicine, Alternative...

Home
Family, Consumers, Cooking...

Kids and Teens
Arts, School Time, Teen Life...

News
Media, Newspapers, Weather...

Recreation
Travel, Food, Outdoors, Humor...

Reference
Maps, Education, Libraries...

Regional
US, Canada, UK, Europe...

Science
Biology, Psychology, Physics...

Shopping
Autos, Clothing, Gifts...

Society
People, Religion, Issues...

Sports
Baseball, Soccer, Basketball...

World
Deutsch, Español, Français, Italiano, Japanese, Nederlands, Polska, Dansk, Svenska...

Become an Editor Help build the largest human-edited directory of the web

Copyright © 1998-2006 Netscape

over 4 million sites - 73,600 editors - over 590,000 categories

Figure 5.5: Overview of the Top Categories of DMOZ Taxonomy

secondly, though still necessitating revisions especially with respect to the taxonomical structure, the classification has emerged as a collaborative effort and is representative for a wide community of Web users. This is an important feature given the absence of a standardized and thoroughly built taxonomy of software.

- `SoftwareApplication`
  - `AnalyticsSoftware`
    - `DataMiningSoftware`
    - `DecisionSupportSoftware`
  - `BankingSoftware`
    - `CreditCardsSoftware`
      - `AutomaticTellerMachines`
      - `ChequeProcessingSoftware`
  - `BusinessSoftware`
    - `AccountingSoftware`
  - `CommerceSoftware`
    - `AuctionsSoftware`
    - `BarCodeScanningSoftware`

    * ReverseAuctionsSoftware
    * TradeSoftware
- CommunicationSoftware
    * CalendarSoftware
    * ContactManagerSoftware
    * EmailSoftware
    * InstantMessagingSoftware
    * VOIPSoftware
- CompilerSoftware
    * CompilerOptimizationSoftware
    * InterpreterSoftware
    * LinkerSoftware
    * LoaderSoftware
    * ParserSoftware
- ComputerGraphicsSoftware
    * AnimationSoftware
    * GraphicsEditorSoftware
    * GraphicsPostprocessingSoftware
    * SpecialEffectsSoftware
- CryptographySoftware
- DatabaseSoftware
- EmbeddedSystems
    * AutomotiveSoftware
    * AvionicsSoftware
    * HVACSoftware
    * MedicalDeviceSoftware
    * TelemetrySoftware
    * TelephonySoftware
- EngineeringSoftware
    * CADSoftware
    * EDASoftware
    * NumericalAnalysisSoftware
    * SimulationSoftware
- FileManagementSoftware
    * FileSharingSoftware
    * FTPSoftware
- FinanceSoftware
    * BondMarketSoftware
    * FuturesMarketSoftware
    * StockMarketSoftware
- GamesSoftware
- InformationManagementSoftware
    * LISSoftware
    * MiSSoftware
- LogisticsSoftware
    * SuplyChainManagementSoftware
- ManufacturingSoftware
    * CAMSoftware
    * DCSSoftware
- MusicSoftware
    * MusicSequencerSoftware

    - SoundEffectsSoftware
    - SynthetizerSoftware
  - NetworkSoftware
    - DNSSoftware
    - ProtocolSoftware
    - RoutingSoftware
  - OfficeSoftware
    - PresentationSoftware
    - TableCalculationSoftware
    - TextProcessingSoftware
  - RoboticsSoftware
  - SignalAnalysisSoftware
    - HandwritingRecognitionSoftware
    - ImageProcessingSoftware
    - SignalProcessingSoftware
    - SpeechProcessingSoftware
    - TextRecognitionSoftware
  - TrafficControlSoftware
  - TrainingSoftware
  - VizualizationSoftware
  - VotingSoftware
  - WWWSoftware
    - BrowserSoftware
    - WebServerSoftware

An individual software application is characterized by an `IndustrialSector` and additional documentation and authoring information.

5.2.9 Class `naics:IndustrialSector`

The main goal of this class is to complement the description of ontology-specific software applications with information about sectors and industries. Our intention was to re-use existing classification systems for this field in order to increase the usability and interoperability of the meta-ontology. By contrast to the software area, interoperable classifications of industries are available from prestigious European and international standardizations organizations. For our purpose we customized the NAICS system, as the most popular industrial sector classification currently available. The most important categories are subsequently listed:

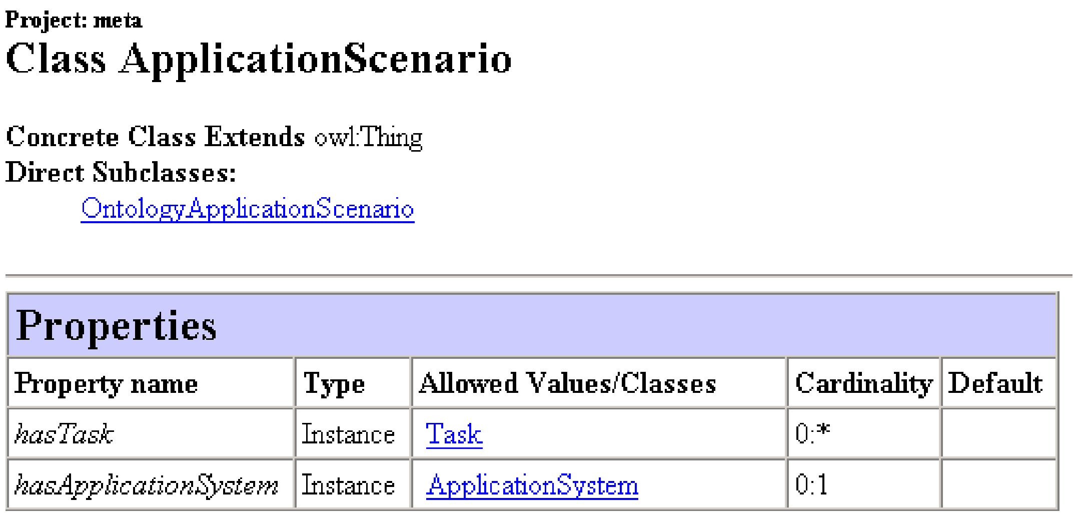
- AgricultureForestryFishingAndHunting
- ArtsEntertainmentAndRecreation
- AccomodationAndFoodServices
- EducationalServices
- HealthCareAndSocialAssistance
- Information
- AdministrativeAndSupportAndWasteManagementAndRemediationServices
- FinanceAndInsurance
- ManagementOfCompaniesAndEnterprises
- ProfessionalScientificAndTechnicalServices

- `RealEstateAndRentalAndLeasing`
- `Manufacturing`
- `Construction`
- `Mining`
- `Utilities`
- `Services`
- `PublicAdministration`
- `WholesaleTrade`
- `RetailTrade`
- `TransportationAndWarehousing`

The ontology was generated programmatically from textual description of the NAICS classification, followed by minor re-naming of categories repeatedly used. This occurs because of the particular structure of the NAICS hierarchy, which foresees a fixed number of inheritance levels. If an inheritance path does not contain a sufficient number of distinct entries, the one situated at the lowest level is replicated on the remaining hierarchy levels.

5.2.10 Class `ApplicationScenario`

The class `ApplicationScenario` is motivated by the lack of n-ary properties in OWL. In order to establish a link between an ontology, the application system it is integrated into, and the particular task the ontology is used for, we introduced a separate class modelling the relationship between the three, as recommended by the W3C Working Group on Best Practices and Deployment.[9]


Project: meta

Class ApplicationScenario

Concrete Class Extends owl:Thing
Direct Subclasses:
OntologyApplicationScenario

Properties

Property name	Type	Allowed Values/Classes	Cardinality	Default
hasTask	Instance	Task	0:*	
hasApplicationSystem	Instance	ApplicationSystem	0:1	

Figure 5.6: Overview of the Class `ApplicationScenario`

The class possesses two important properties referring to the class `Task` and `wikipedia:Application`, respectively.

[9]`http://www.w3.org/2001/sw/BestPractices/` last visited in May, 2006

5.2.11 Class `OntologyTask`

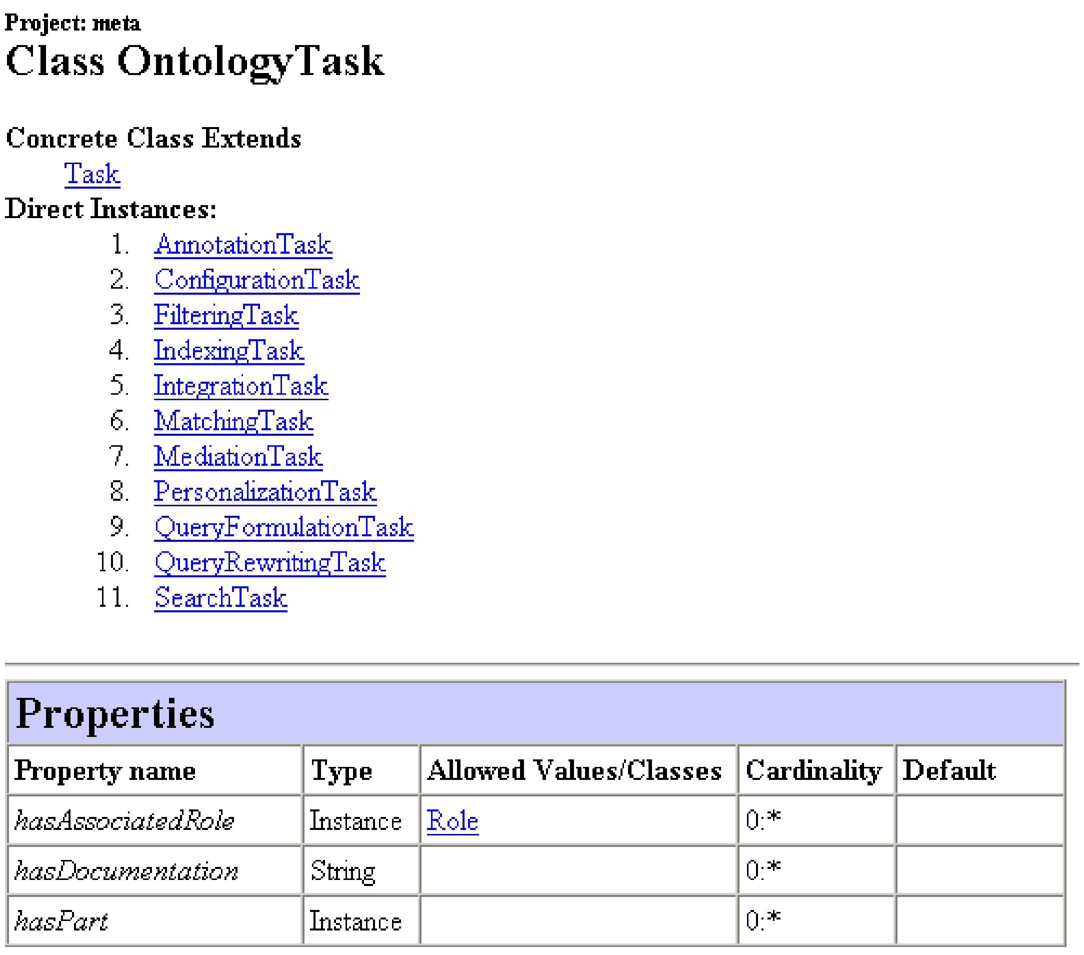
The class of `OntologyTask` is targeted at providing a (basic) representation of the most common activities ontologies have been used for in the Semantic Web community. Far from pretending to realize a fully-fledged task or process modelling in the sense of problem solving methods or more recently Semantic Web Services, the task-oriented ontology sub-fragment can be used to capture usage-related information about ontologies. The concrete tasks were compiled as a result of an extensive literature research process, in which we analyzed some of the most comprehensive surveys regarding ontology applications in the last decades.

The different purposes of an ontology are classified by [223] in 3 high-level categories:

1. **Communication**: ontologies are built to reduce conceptual and terminological ambiguity between interacting agents, be that humans of machines.
2. **Interoperability**: similar to the previous context, ontologies are designed to provide a common basis for tool interaction.
3. **System engineering**: the ontology is used as a means to unambiguously represent diverse specifications for a software system.

The classification of Uschold and Grüninger was not used in our metadata model due to its poor applicability to the Semantic Web context, and because of the coarse granularity level. More fine-grained classifications have resulted from several recent European projects in the area of Semantic Web. For the realization of the metadata model we merged the most recent ones: a classification of ontology-based applications provided by the European project SWAD (Semantic Web Applications and Development) [215] and a taxonomy of knowledge processing tasks emerged in the European network of excellence KnowledgeWeb [115]. The result was a list of 11 ontology tasks (see Figure 5.7), which are defined as follows:

- `AnnotationTask`: the ontology is used as a controlled vocabulary to annotate Semantic Web resources. This task includes the usage of a semantically rich ontology for representing arbitrarily complex annotation statements on these resources. The task can be performed manually or (semi-)automatically.
- `ConfigurationTask`: the ontology is designed to provide a controlled and unambiguous means to represent valid configuration profiles in application systems. As the aim of the ontology is to support the operationalization of particular system-related processes; this task is performed automatically in that the ontology is processed in an automatic manner by means of reasoners or APIs.
- `FilteringTask`: the task describes at a very general level how ontologies are applied to refine the solution space of a certain problem, such as information retrieval or personalization. The task is targeted at being performed semi-automatically or automatically.
- `IndexingTask`: in this scenario, the goal of the ontology is to provide a clearly defined classification and browsing structure for the information items in a repository. Again, the task can be performed manually by domain experts or as part of an application in an automatic or semi-automatic way.



Project: meta

Class OntologyTask

Concrete Class Extends
Task

Direct Instances:

1. AnnotationTask
2. ConfigurationTask
3. FilteringTask
4. IndexingTask
5. IntegrationTask
6. MatchingTask
7. MediationTask
8. PersonalizationTask
9. QueryFormulationTask
10. QueryRewritingTask
11. SearchTask

Properties

Property name	Type	Allowed Values/Classes	Cardinality	Default
hasAssociatedRole	Instance	Role	0:*	
hasDocumentation	String		0:*	
hasPart	Instance		0:*	

Figure 5.7: Overview of the Class `OntologyTask`

- `IntegrationTask`: the task characterizes how ontologies provide an integrating environment, an inter-lingua, for information repositories or software tools. In this scenario the ontology is applied (semi-)automatically to merge between heterogeneous data pools in the same or in adjacent domains.

- `MatchingTask`: the goal of matching is to establish links between semantically similar data items in information repositories. In contrast to the previous task, matching does not include the production of a shared final schema/ontology as a result of aggregating the matched source elements to common elements. With respect to the automatization level the range varies from manual to fully-automatical execution.

- `MediationTask`: the ontology is built to reduce the ambiguities between communicating human or machine agents. It can act as a normative model which formally and clearly defines the meaning of the terms employed in agent interactions. In the context of programmed agents, the task is envisioned to be performed automatically.

- `QueryFormulationTask`: the ontology is used in information retrieval settings as a controlled vocabulary for representing user queries. Usually the task is performed automatically in that the concepts of the ontology is are listed in a query formulation front-end in order to allow users to specifies their queries.

- `QueryRewritingTask`: complementary to the query formulation dimension, this task applies ontologies to semantically optimize query expressions by means of the domain knowledge (constraints, subsumption relations etc.) The task can be interpreted as a particular art of filtering information. The task is performed automatically; however, it assumes the availability of patterns describing the transformations at query level.

- `PersonalizationTask`: the ontology is used mainly for providing personalized access to information resources. Individual user preferences with respect to particular application settings are formally specified by means of an ontology, which, in conjunction with appropriate reasoning services, can be directly integrated into a personalization component for filtering purposes. The usage of ontologies in personalization tasks might be carried out in various forms, from a direct involvement of the user who manually specifies ontological concepts which optimally describe his preferences, to the ontological modelling of user profiles.

- `SearchTask`: the task characterizes how ontologies are used to refine common keyword-based search algorithms using domain knowledge in form of subsumption relations. Ontology-driven search is usually performed automatically by means of reasoning services handling particular aspects of an ontology representation language.

In achieving these tasks Semantic Web ontologies may play various roles, which are also modelled as classes:

- `VocabularyRole`: ontologies provide a controlled means to describe and communicate about a specific domain. This role is characteristic to tasks such as annotation, configuration, mediation, query formulation and rewriting. It assumes a very simple ontological structure i.e. a list of pre-defined domain- or application-relevant terms modelled as concepts, relationships or instances.

- `IndexRole`: the ontology is used as a classification structure e.g., for indexing or query rewriting tasks. With respect to the richness of the domain representation, indexing implies the availability of a taxonomical structure.

- `ModelRole`: the ontology is used as a formal or at least semi-formal representation of the application domain, specifying the meaning of the domain concepts by means of properties, attributes and logical constraints. The role can be associated with tasks which imply ontology-based reasoning services, such as search/retrieval, annotation, configuration, personalization, indexing, query rewriting, mediation, matching and integration.

- `FilterRole`: in this role the ontology is applied to refine the results of a specific algorithm, usually in scenarios like information retrieval or personalization.

The correspondences between certain types of roles and ontology-related tasks are represented at implementation level using `owl:allValuesFrom` range restrictions on the property `hasAssociatedRole` at the class `OntologyTask`. They are summarized in Table 5.1.

	Vocabulary	Formal model	Index	Filter
Annotation	✓	✓	✓	-
Configuration	✓	✓	-	✓
Filtering	-	-	-	✓
Indexing	✓	✓	✓	-
Integration	✓	✓	-	-
Matching	✓	✓	-	✓
Mediation	✓	✓	-	-
Query formulation	✓	-	-	✓
Query rewriting	✓	✓	✓	✓
Personalization	-	✓	-	✓
Search	✓	✓	✓	✓

Table 5.1: Alignment of Ontology Roles to Ontology Tasks

5.2.12 Class `MethodsMethodologiesAndTools`

The aim of the class `MethodsMethodologiesAndTools` is to capture pragmatic information about the engineering process an ontology resulted from. We distinguish between three sub-classes of this class in accordance with the IEEE Standard Glossary of Software Engineering Terminology [103]:

- `EngineeringMethodology`: a methodology describes an inventory of methods and techniques to create a general system theory on how a particular class of challenging tasks should be performed.
- `Method`: a method describes the procedure necessary to be carried out when engineering a product or performing. The terms *"method"* and *"technique"* are often used alternatively, while the latter focuses rather on technical aspects.
- `Tool`: a tool is a piece of software which provides means for tool users to apply methods and methodologies in practice.

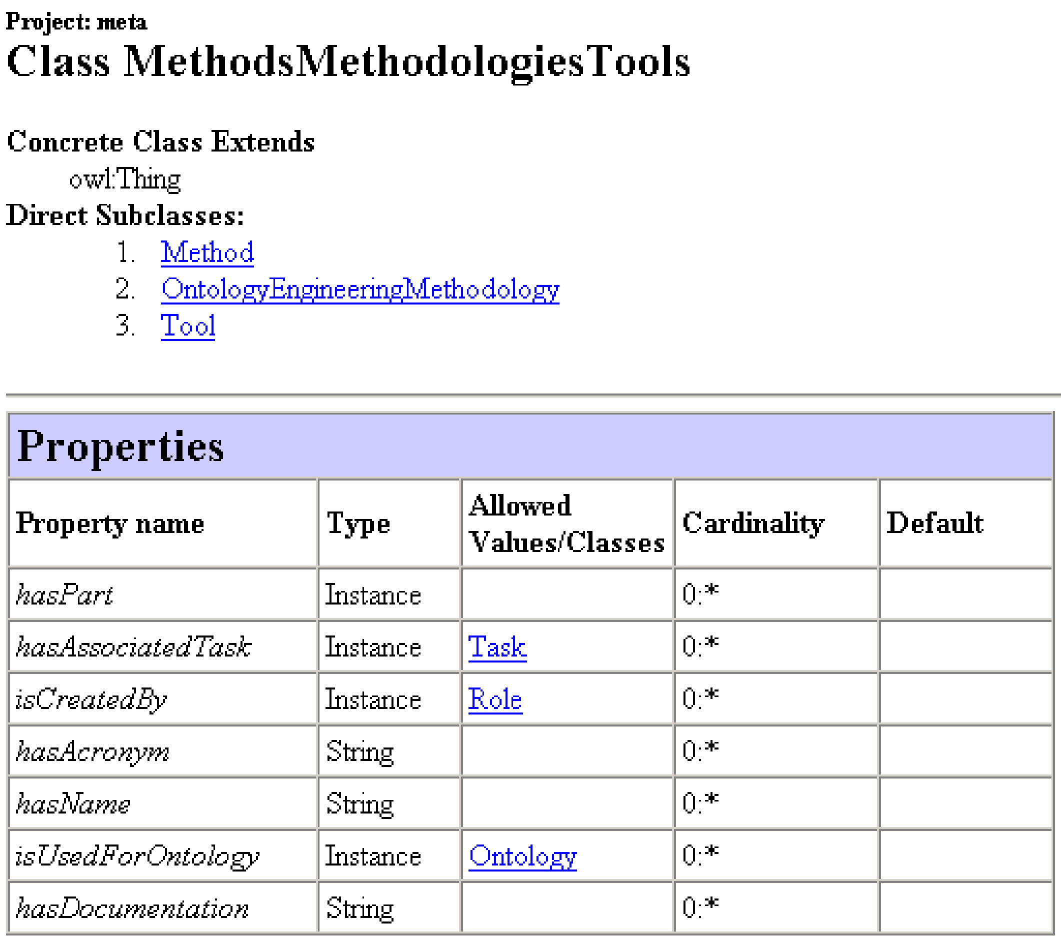
Authoring and documentary information about these ontology engineering aspects can be specified in terms of the properties `hasName`, `isCreatedBy` and `hasDocumentation`, whose meaning is self-explanatory. Every method, methodology or tool may be used to support or control the execution and results of a particular `Task`, such as `OntologyDevelopment` or, more relevant for the ontology reuse context, `OntologyIntegration`, `OntologyMerging` and `OntologyEvaluation`. The methodological and technical support in the area of reuse is represented explicitly in the ontology, while general-purpose engineering issues can be easily added to the model. Particular proposals in one of these areas are modelled as instances.

5.3 Methods for Metadata Creation and Management

In addition to metadata representation, the issue of creating, maintaining and using ontology metadata is equally important for providing real added value to current ontology reuse processes. Once the valid form and correct usage of the metadata entities has been established (e.g., by means of an ontology) existing and newly built Semantic Web ontologies should be uniformly described by metadata records in order to foster their reusability. This can be ensured by making ontology development and management tools such as ontology editors or ontology repositories aware of the ontology metadata model in that they provide means to capture, store and update this additional information. Independently of the environment in which a metadata-driven behavior is integrated into, this environment should incorporate a series of functional components as depicted in Figure 5.9 [135].

Orthogonal to the core services for creating, revising, storing and proving the quality of the metadata records, the system should also take into account issues related to metadata authoring, providing an access policy mechanism which specifies the roles of the metadata providers and the way they interact with the metadata repository.

This thesis discusses how the enumerated core metadata services (cf. Figure 5.9) could be realized in the current technological landscape. A comprehensive design or an implementation of a complete metadata management system are out of the scope of this work,



Project: meta

Class MethodsMethodologiesTools

Concrete Class Extends
owl:Thing

Direct Subclasses:

1. Method
2. OntologyEngineeringMethodology
3. Tool

Properties

Property name	**Type**	**Allowed Values/Classes**	**Cardinality**	**Default**
hasPart	Instance		0:*	
hasAssociatedTask	Instance	Task	0:*	
isCreatedBy	Instance	Role	0:*	
hasAcronym	String		0:*	
hasName	String		0:*	
isUsedForOntology	Instance	Ontology	0:*	
hasDocumentation	String		0:*	

Figure 5.8: Overview of the Class `MethodsMethodologiesAndTools`

which, however, focuses on one of the most challenging questions regarding metadata provision, i.e. on methods to *automatically* generate specific fragments of the proposed metadata model. The remaining services (i.e. human-driven metadata provision, quality checking, metadata updating and persistent storage) are briefly addressed, as they are currently under development in external projects such as Onthology and Oyster with our collaboration.[10] The compatibility between the schemas underlying these ontology management tools and our metadata ontology is ensured as part of the ontology evolution strategy presented in Section 5.1. An integration of the technologies presented here with these environments is planned for the near future.

[10]`http://www.onthology.org:8080/index.jsp`, `http://oyster.ontoware.org/oyster/oyster.html` last visited in May, 2006

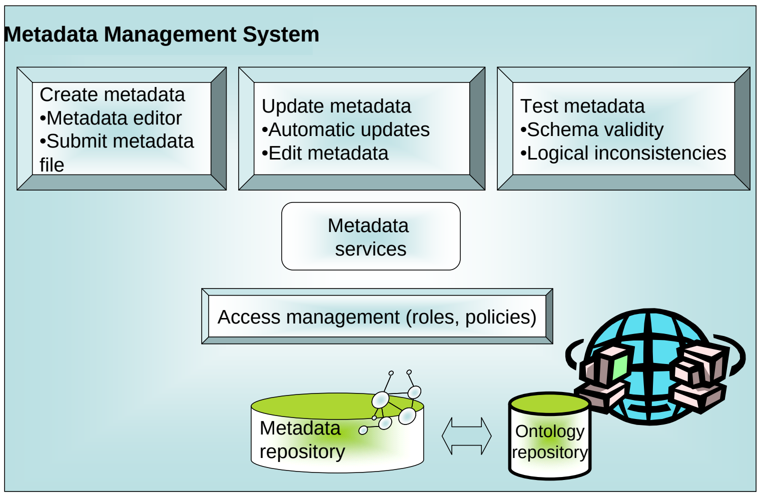


Figure 5.9: Core Metadata Services

5.3.1 Metadata Creation

Creating new metadata entries can be realized in various ways: firstly, ontology development tools, as well as ontology repositories can be extended with components allowing humans to manually annotate ontologies with metadata information. Secondly, these platforms can be enhanced with methods to automatically acquire (at least a fragment of) the metadata at run-time, during the implementation of the ontology, or at post-implementation time. The human-driven metadata creation is directly related to the benefits induced by the availability of this information for the metadata authors, be that ontology developers or users, as this task naturally implies resources to understand the metadata schema and to provide the required information. The computer-aided metadata acquisition tries to explicitly cope with this cost benefit ratio by generating parts of the metadata entries automatically on the basis of user profile information or special heuristics (see below), thus decreasing the amount of effort invested by humans for this purpose. The process of metadata provision is reduced to the human-driven quality assurance, which has the aim of testing the accuracy of results of the automatic metadata acquisition. An operationalization of the automatic acquisition task is, however, directly related to the Semantic Web setting and the semantic conceptualization of the metadata information. Under these circumstances, it is possible to access publicly available information sources in order to derive specific metadata entries, and to adjust the results of this discovery by means of ontological background knowledge.

Automatic Metadata Creation

Following the classification of the metadata entries in terms of the semiotic framework by [206], we discuss possible methods to automatically acquire syntactic, semantic and pragmatic metadata, which is represented using the ontology introduced in the previous sections. This theoretical analysis forms the basis for the realization of the OntoMeta tool introduced in Chapter 7.

Automatic Generation of Syntactic Metadata The syntactic metadata captures information about the extrinsic properties of a Semantic Web ontology: its underlying graph structure and the syntax of the implementation language. The metadata elements listed in this category can be derived programmatically for every ontology represented in a (semi-)formal representation language:

Number of classes/instances : this can be computed by means of an API which is able to map the syntactic representation of the ontology into a graph structure. The number of classes is the number of labeled class nodes in the graph, while the number of instances are calculated as the number of labeled instance nodes.

Number of properties : this entry can be calculated similarly to the number of classes and instances. Provided an API which maps properties between concepts and concepts and literals, respectively to edges in a graph, the number of properties is computed as the number of labeled edges between labeled class nodes.

Number of axioms : the algorithm for the calculation of this metric strongly depends on the representation of the axioms in the ontology implementation language and on the way the ontology management API is handling these axioms. In OWL, for instance, ontologies axioms are represented as sub-classes of a generic class `owl:Restriction`. Ontologies implemented using the Frames formalism represent axioms in form of specially designed rule constructs. Independently of the implementation language, the number of axioms is calculated as the number of occurrences of these elements in the ontology.

Number of imported ontologies : this metadata element returns the number of directly imported ontological sources. The granularity of the imported sources is considered at file level.

Syntax : this entry captures the particular syntax of the implementation language used in the ontology. It can be derived using specific methods of the ontology management API.

Ratio of used syntax : this metric aims at capturing information about the degree of language usage covered by a particular ontology. It is calculated as the ratio between the number of distinct syntactic constructs used in the ontology and the total number of syntactic constructs contained by the vocabulary of the knowledge representation language.

$$RUS = \frac{numberofusedconstructs}{totalnumberofconstructsinthelanguage}$$

Used syntax : this element contains a list of the syntactic language constructs concretely used in the ontology. The list can be derived in parallel to the computation of the used syntax ratio.

Ratio of invalid constructs : the value of this metadata entry is computed in conjunction with a particular validation tool. The value is set by the result of the validation procedure, while a null value indicates syntactic validity.

$$RInvC \quad = \quad \frac{number of invalid statements}{total number of statements in the ontology}$$

Acquisition of Semantic Metadata Semantic metadata elements provide descriptive information about the (formal) meaning of the contents of the ontology. They can be divided into two sub-categories: metadata related to the ontology domain, and metadata related to the formal semantics of the ontology representation language.

By contrast to the syntactic metadata, parts of this information are often considered of subjective nature, and can not be automatically captured by means of standard calculations or tools. The majority of the elements contained in this category (in particular the ones related to the modelled domain) can, however, be computed using specific heuristics and methods targeted at mapping the contents of the ontology to pre-defined values of the metadata elements:

Readability : the readability element indicates whether the ontological primitives (especially concepts and relationships) are denominated by human-readable unambiguous labels. This metadata element can be quantified only in relation to a reference source for human readability such as a dictionary or a thesaurus. Further on, the ambiguity of the labels can be detected using word disambiguation heuristics as those emerged in the last 20 years in the Natural Language Processing field [24, 70, 102, 230]. Deriving readability information on ontologies is, however, a non-trivial task. Certain types of ontologies, in particular those aimed at human consumption and at linguistic processing (e.g., glossaries, catalogues, vocabularies, classifications, thesauri) can be per default considered to be human-understandable. Highly-formal ontologies do not necessarily denominate ontological primitives by natural language labels, as the meaning of the conceptualization is specified using the model-theoretic semantics of the knowledge representation language. In this case ontologies usually provide concept labels in form of (abbreviated) word expressions potentially constrained by (implicit) naming conventions. Heuristics aiming at extracting readability and ambiguity information from this type of ontologies should be able to handle composed concept labels, naming conventions and abbreviations.

Number of comments and definitions : this element is adjacent to the readability dimension, as the availability of documented conceptualizations considerably eases the interaction between human users and the ontology. As comments and definitions are usually uniquely labeled at syntax level, their number can be easily calculated in the same manner as the syntactic metadata entries.

Ontology domain : this metadata element aims at providing information about the topics or subjects covered by an ontology in terms of a specific classification or a controlled vocabulary. As the range of the metadata element is pre-defined by the metadata ontology (i.e. the class `dmoz:Topic`), acquiring the domain(s) modelled by an ontology results in finding a method to map the contents of the ontology to one or more topics in the Open Directory taxonomy. However, this information can be automatically captured only if we assume the usage of human-readable labels and the availability of a correspondence between the semantics implicitly carried by the assigned primitives' names and the real-world meaning of the modelled domain.

View of the conceptualization upon the domain : this metadata element offers complementary information to the previous one. Based on the empirical finding that the reusability of the domain-overlapping Semantic Web ontologies is significantly dependent on the context the ontologies originally emerged from, the metadata describes the general point of view of the engineering team. Technically, the view is represented as a topic in the Open Directory taxonomy as well. In fact, it is calculated as the most representative super-class of the topic representing the domain of the ontology, as foreseen by the DMOZ hierarchy (cf. Chapter 7).

Type of ontology : in order to automatically acquire information about the type of an existing ontology (i.e. whether we are dealing with a domain, upper-level, application or knowledge representation ontology, see Section 5.2) the metadata management component requires classification heuristics, which are able to order an ontological source in (at least) one of the mentioned classes of ontologies on the basis of a pre-defined set of characteristic features and rules. Theoretical approaches, as well as feasible practical implementations for the realization of this task are available in the artificial intelligence (clustering, classification) field.

Formality level : an automatic generation of the value of this metadata element results in assigning one of the four pre-defined formality levels ranging from highly informal to highly formal to existing ontologies. Similarly to the automatic typing of ontologies, one needs characteristic features and heuristic rules to apply them in order to perform the desired classification task.

Natural language : this element aims at providing information about the natural language used in the ontology vocabulary. The automatic generation of its value requires techniques like language guessing, which have been developed for some time in the area of computer linguistics.

Metadata describing the knowledge representation language underlying the ontology can be automatically acquired, similarly to the syntactic elements in the previous category:

Knowledge representation language : information about the knowledge representation language is usually provided by the language parser. For Semantic Web languages it is furthermore important to derive not only the representation language, but also the exact sub-fragment of the language, as this has implications on the applicable reasoning services and further on the complexity of particular computations. Due to the ontological representation of the metadata schema, this information can be automatically inferred by a reasoner, provided that a complete description of the features supported by particular fragments of ontology representation languages are defined explicitly.

Knowledge representation paradigm : this information can be automatically extracted if the metadata acquisition component can resort to knowledge about the relationship between languages and knowledge representation paradigms. For example, the information that OWL belongs to the class of Description Logics is explicitly encoded in the metadata ontology and can be used to acquire this missing link if the language is known in advance to the system.

Ratio of inconsistent constructs : the value of this metadata entry is computed using particular reasoning tool. The value is set by the result of the consistency checking procedure, while a null value indicates semantic consistency. Consistency checking could theoretically be carried out by humans as well. Nevertheless, it is a time-consuming and error-prone activity, which requires deep knowledge of the language employed and its model-theoretic semantics.

$$RIncC \quad = \quad \frac{number of reported inconsistencies}{total number of ontological primitives}$$

Acquisition of Pragmatic Metadata Without any human intervention the extraction of pragmatic metadata information from existing ontological sources can be performed with limited success. Given its intrinsic irregular nature 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 activities. This requirement is not fulfilled by the majority of ontology engineering tools to a satisfactory extent so far. In consequence, available metadata information is restricted to the following entries at most:

Ontology engineering tool : the name of the tool used to perform a particular ontology engineering task or activity.

Authoring and provenance information : the name of the authors and contributors. Usually this information is located in the header of the ontological resource and can be directly captured.

Versioning information : this metadata element aims at providing information about the current version of a particular ontology. Just as in the previous case versioning metadata is located in the header of the ontology file(s) and can be (if available) easily extracted using automatic methods.

Documentation : the documentation metadata is conceived as a list of textual resources describing the content, the application scenario, the engineering process of a specific ontology. Due to the lack of standardized documentation templates in the new field of ontology engineering, these resources—which are, moreover, not directly linked to the actual ontological content—can not be discovered solely by means of Web-based search heuristics, which aim at identifying relevant documents from the list of results returned by pre-defined queries.

The remaining elements can not be acquired in an automatic manner without considerable pre- and post-processing efforts which might under circumstances outweigh the benefits of an operationalized approach to metadata creation. They are expected to be generated with the help of humans, provided an increasing awareness of the Semantic Web community with respect to the importance of metadata for feasibly reusing ontologies.

The enumerated considerations have been prototypically implemented for Semantic Web ontologies represented in the RDFS and OWL languages. Details on the implementation and the evaluation of its results are presented in Chapters 7 and 8, respectively.

User-driven Metadata Creation

A definitely more trustworthy, but in the same time more resource-intensive alternative to create metadata about ontologies is the manual one. In this case ontology developers, reviewers or users are provided with a pre-defined framework to capture the ontology-related information and to manage and use it systematically for reuse purposes.

The manual metadata creation can be operated at development time with the help of ontology management tools recording metadata about the engineering process and the intrinsic properties of the ontology being built. Ontology evaluation tools can embed metadata components which allow the creation of entries related to the results of a particular evaluation procedure (e.g., the name of the reviewers, the methods employed, links to the documentation).

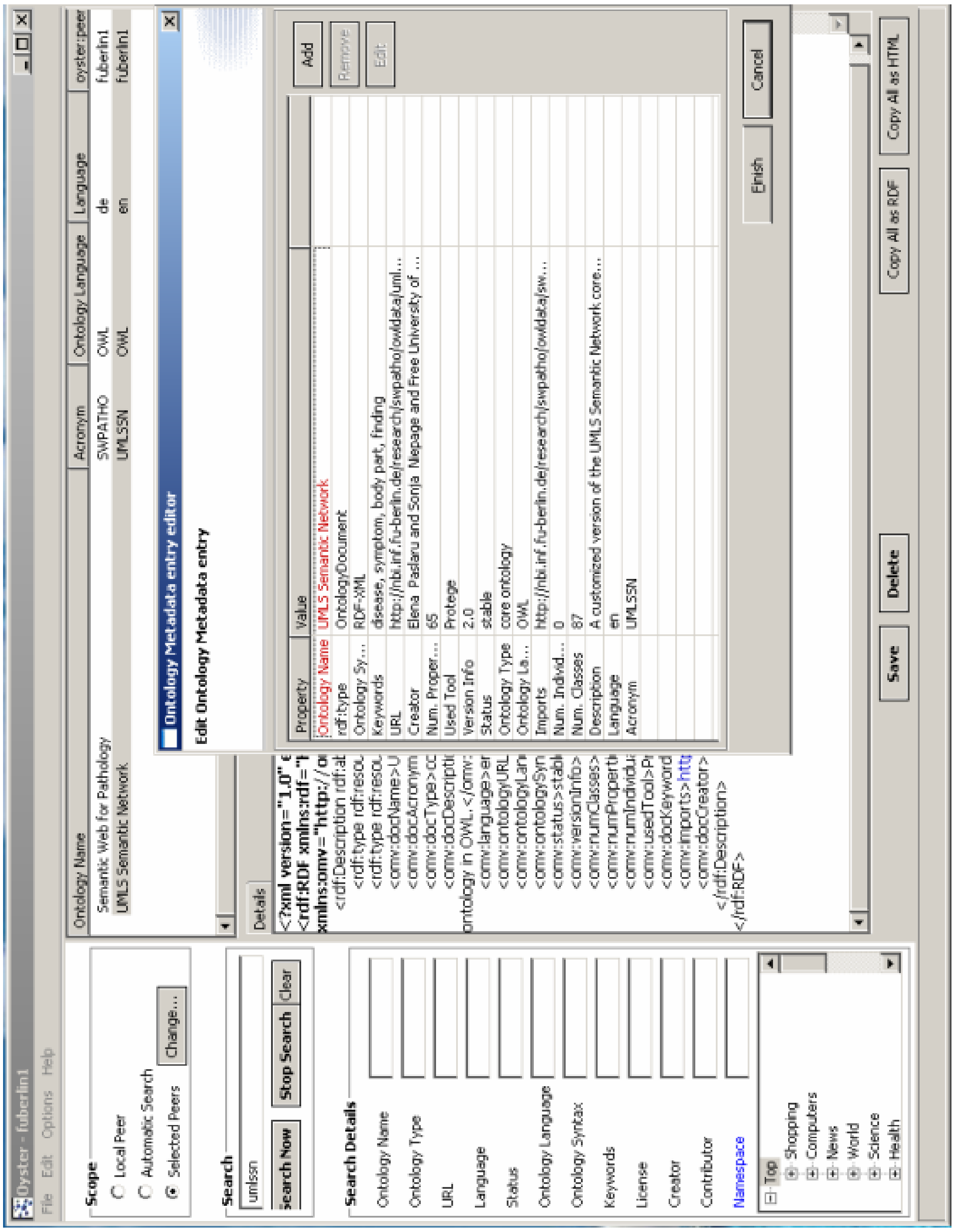
Technologically this task can be easily supported by simple questionnaires structuring this task and providing default or permitted values and value ranges for the entries defined by the metadata ontology. The Oyster system, primarily developed at the Universidad Politecnica de Madrid, offers such functionality in the context of a distributed ontology development scenario, in which ontology engineers create and exchange ontology metadata information, thus facilitating reuse (cf. Figure 5.10). Oyster uses the OMV vocabulary to manage ontology metadata information.[11] Each peer-developed ontology is accompanied by ontology metadata information which can be created and revised by ontology authors. It can be exported to RDF as instance data of the OMV schema and forms the basis for structured distributed search techniques, which have the aim to discover remotely developed ontologies fulfilling certain criteria.

A second example is the Onthology project from the University of Karlsruhe which manages a centralized ontology repository and the associated metadata. Similarly to Oyster it integrates basic metadata editing functionality, as illustrated in Figure 5.11. In addition to manual metadata creation Onthology provides a prototypical metadata update service, which will be described in the next section, and an ontology rating component.

5.3.2 Metadata Management

The metadata creation is only the first step in a fully-fledged metadata-based ontology reuse environment. Prior to submitting the new entries to the system the metadata might be subject to a quality assurance procedure whose aim is to identify potential errors, inconsistencies or missing mandatory information. As the metadata is conceptualized by means of an ontology, the validity of the provided information w.r.t. the schema is easily ensured. Further logical inconsistencies can be detected if the system has access to deep background knowledge on

[11]`http://oyster.ontoware.org/` last visited in May, 2006


Oyster - fuberlin1

Ontology Name	Acronym	Ontology Language	Language	oyster:peer
Semantic Web for Pathology	SWPATHO	OWL	de	fuberlin1
UMLS Semantic Network	UMLSSN	OWL	en	fuberlin1

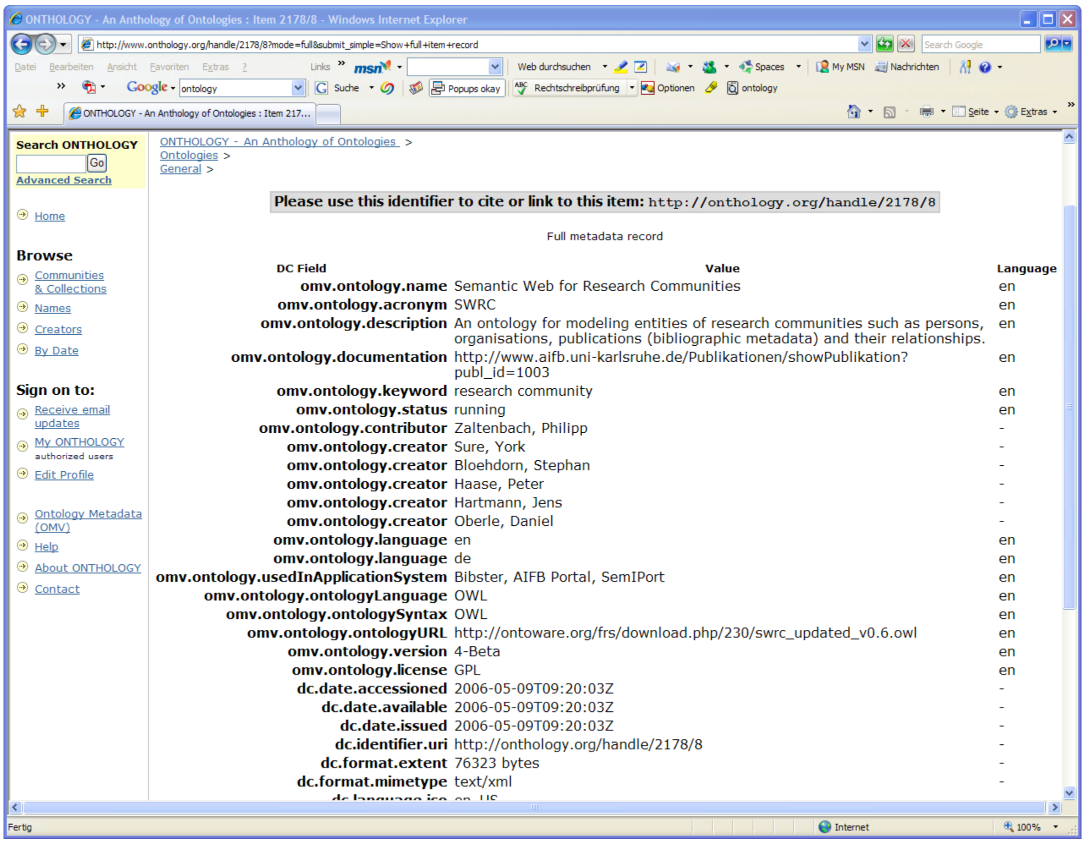
Ontology Metadata entry editor

Edit Ontology Metadata entry

Property	Value
Ontology Name	UMLS Semantic Network
rdf:type	OntologyDocument
Ontology Sy...	RDF-XML
Keywords	disease, symptom, body part, finding
URL	http://nbi.inf.fu-berlin.de/research/swpatho/owldata/uml...
Creator	Elena Paslaru and Sonja Nlepage and Free University of ...
Num. Proper...	65
Used Tool	Protege
Version Info	2.0
Status	stable
Ontology Type	core ontology
Ontology La...	OWL
Imports	http://nbi.inf.fu-berlin.de/research/swpatho/owldata/sw...
Num. Individ...	0
Num. Classes	87
Description	A customized version of the UMLS Semantic Network core...
Language	en
Acronym	UMLSSN

```
<?xml version="1.0" e
<rdf:RDF xmlns:rdf="h
xmlns:omv="http://o
  <rdf:Description rdf:ab
    <rdf:type rdf:resou
    <rdf:type rdf:resou
    <omv:docName>U
    <omv:docAcronym
    <omv:docType>cc
    <omv:docDescripti
ontology in OWL.</omv:
    <omv:language>er
    <omv:ontologyURL
    <omv:ontologyLan
    <omv:ontologySyn
    <omv:status>stabl
    <omv:versionInfo>
    <omv:numClasses>
    <omv:numProperti
    <omv:numIndividu
    <omv:usedTool>Pr
    <omv:docKeyword
    <omv:imports>http
    <omv:docCreator>
  </rdf:Description>
</rdf:RDF>
```

Figure 5.10: Metadata Creation Support in Oyster


ONTHOLOGY - An Anthology of Ontologies : Item 2178/8 - Windows Internet Explorer
http://www.onthology.org/handle/2178/8?mode=full&submit_simple=Show+full+item+record
Search Google
Datei Bearbeiten Ansicht Favoriten Extras ?
Links msn Web durchsuchen Spaces My MSN Nachrichten
Google ontology Suche Popups okay Rechtschreibprüfung Optionen ontology
ONTHOLOGY - An Anthology of Ontologies : Item 217...
Seite Extras
Search ONTHOLOGY
Go
Advanced Search
Home
Browse
Communities & Collections
Names
Creators
By Date
Sign on to:
Receive email updates
My ONTHOLOGY
authorized users
Edit Profile
Ontology Metadata (OMV)
Help
About ONTHOLOGY
Contact
Fertig
Internet
100%

ONTHOLOGY - An Anthology of Ontologies >
Ontologies >
General >

Please use this identifier to cite or link to this item: `http://onthology.org/handle/2178/8`

Full metadata record

DC Field	Value	Language
omv.ontology.name	Semantic Web for Research Communities	en
omv.ontology.acronym	SWRC	en
omv.ontology.description	An ontology for modeling entities of research communities such as persons, organisations, publications (bibliographic metadata) and their relationships.	en
omv.ontology.documentation	http://www.aifb.uni-karlsruhe.de/Publikationen/showPublikation?publ_id=1003	en
omv.ontology.keyword	research community	en
omv.ontology.status	running	en
omv.ontology.contributor	Zaltenbach, Philipp	-
omv.ontology.creator	Sure, York	-
omv.ontology.creator	Bloehdorn, Stephan	-
omv.ontology.creator	Haase, Peter	-
omv.ontology.creator	Hartmann, Jens	-
omv.ontology.creator	Oberle, Daniel	-
omv.ontology.language	en	en
omv.ontology.language	de	en
omv.ontology.usedInApplicationSystem	Bibster, AIFB Portal, SemIPort	en
omv.ontology.ontologyLanguage	OWL	en
omv.ontology.ontologySyntax	OWL	en
omv.ontology.ontologyURL	http://ontoware.org/frs/download.php/230/swrc_updated_v0.6.owl	en
omv.ontology.version	4-Beta	en
omv.ontology.license	GPL	en
dc.date.accessioned	2006-05-09T09:20:03Z	-
dc.date.available	2006-05-09T09:20:03Z	-
dc.date.issued	2006-05-09T09:20:03Z	-
dc.identifier.uri	http://onthology.org/handle/2178/8	-
dc.format.extent	76323 bytes	-
dc.format.mimetype	text/xml	-
dc.language.iso	en_US	

Figure 5.11: Metadata Creation Support in Onthology

ontologies and ontology engineering. This implies a fundamental refinement of the core metadata model in terms of a more detailed representation of the ontology engineering field and in terms of a deep axiomatization of the underlying knowledge. By the knowledge of the author there is currently no tool supporting this type of quality assurance behavior. The aforementioned systems implement schema validity checking in a superficial manner, at most.

Updating metadata entries is directly related to the technical realization of the metadata management environment [18]. In our case, the metadata is modelled as an OWL ontology; every metadata entry corresponds to a (particular version or fragment) of an URI-identified Semantic Web ontology. Tools managing the metadata (e.g., an ontology repository) can provide techniques to detect changes at the corresponding locations and to subsequently trigger metadata actualization procedures. If the metadata component is embedded in an ontology development tool, the update procedure is triggered by the completion of a new version of the same ontology (e.g., when the ontology file is saved). Updates should, however, take into consideration a series of aspects related to the semantics of the ontology metadata model and to the particularities of ontology development and management:

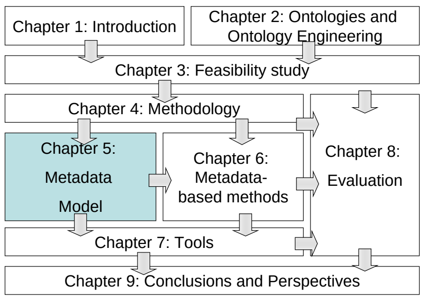
- **Resource identification**: creating and maintaining a link between data and metadata and the platforms managing these types of information is a non-trivial problem across various fields of research in computer science. While each metadata record needs to refer to the described resource by means of an identifier, its uniqueness and consistency require a reliable synchronization mechanism.

- **Metadata record granularity**: in our case metadata applies to both ontologies, but also fragments or groups of them. The model assumes the creation of a new metadata record (for ontologies or groups of ontological primitives) for each new version of an ontology, denominated by a new URI. Consecutive versions are interrelated by means of a dedicated relationship `isPreviousVersionOf`. Its range could be automatically set to the corresponding resource once a new version of the ontology is available.

- **Volatility of metadata elements**: the range of the update activities also depends on the volatility of the metadata entries in relation to particular ontology modifications. A new version of an ontology causes the creation of a new metadata record, which, however, shares metadata entries with the ones associated to previous versions of the same ontological resources. Further more, within the same release of an ontology, a significant contingent of metadata does not change: the domain and the view upon the domain, the authors and contributors, as well as the original scope and purpose, the implementation language and the knowledge representation paradigm. By contrast, numerical entries like the number of ontological primitives and their labels are continuously changing; these values can, however, be computed automatically. If an ontology is utilized in a new application setting, information regarding this setting are recorded in addition to the existing metadata.

Current ontology repositories provide solely primitive means to address the metadata update problem. The majority of them (such as the DAML Ontology Library or SchemaWeb) rely on the vigilance of the metadata providers or ontology developers to maintain the records consistently. Onthology implements a simple update mechanism which relies on internal identifiers for ontologies (cf. Figure 5.11 top). Once the information about an ontology has been inserted to the system, it can be stored persistently, and can be revised and modified if new facts related to the ontology become available. The storage of the metadata information can be handled in the same way as every Semantic Web ontology, by using relational databases or native RDF stores such as Sesame [23] or 3store [93].

5.4 Summary


Chapter 1: Introduction
Chapter 2: Ontologies and Ontology Engineering
Chapter 3: Feasibility study
Chapter 4: Methodology
Chapter 5: Metadata Model
Chapter 6: Metadata-based methods
Chapter 8: Evaluation
Chapter 7: Tools
Chapter 9: Conclusions and Perspectives

This chapter was dedicated to the metadata model capturing information about the context in which ontology reuse processes are performed. The model includes entries associated with intrinsic properties of ontologies, but also process-driven information which might enable its reusability in concrete settings. The second part of the chapter approached methods to create and manage ontology metadata. In particular we addressed the topic of automatic metadata generation, which is vital for the real added value and the impact of metadata in general. Aspects of metadata management and manual metadata provision were briefly exemplified by means of two applications already making use of our metadata model, which have been implemented by external parties.