D1.2.2 Semantic Web Framework Requirements Analysis

Coordinator: Wolf Siberski (L3S)

Jérôme Euzenat (INRIA), Jens Hartmann (UKARL), Alain Léger (FT), Diana Maynard (USFD), Jeff Pan (VUM), Maria del Carmen Suárez-Figueroa (UPM)

with contributions from:
Maud Cahuzac (FT), John Domingue (OU), Rafael González-Cabero (UPM), Shishir Garg (FT), Asunción Gómez-Pérez (UPM), Manuel Lama (Universidade de Santiago de Compostela), Angel López-Cima (UPM), Miguel Rodríguez-Hernández (UPM), Dumitru Roman (DERI Ireland), Pavel Shvaiko (UniTn), Michael Stollberg (DERI Ireland), Farouk Toumani (LIMOS-University of Clermont-Ferrand)

Abstract.
EU-IST Network of Excellence (NoE) IST-2004-507482 KWEB
Deliverable D1.2.2 (WP1.2)
This document reports on requirements for the Semantic Web Framework from industry usage point of view. The framework is split into components for which requirements, characteristics and available solutions are described.
Keyword list: Semantic Web Framework, Semantic Web Components

Document Identifier	KWEB/2005/D1.2.2/v1.3
Project	KWEB EU-IST-2004-507482
Version	v1.3
Date	June 30, 2005
State	draft
Distribution	public

Knowledge Web Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2004-507482.

University of Innsbruck (UIBK) - Coordinator
Institute of Computer Science
Technikerstrasse 13
A-6020 Innsbruck
Austria
Contact person: Dieter Fensel
E-mail address: dieter.fensel@uibk.ac.at

École Polytechnique Fédérale de Lausanne (EPFL)
Computer Science Department
Swiss Federal Institute of Technology
IN (Ecublens), CH-1015 Lausanne
Switzerland
Contact person: Boi Faltings
E-mail address: boi.faltings@epfl.ch

France Telecom (FT)
4 Rue du Clos Courtel
35512 Cesson Sévigné
France. PO Box 91226
Contact person : Alain Leger
E-mail address: alain.leger@rd.francetelecom.com

Freie Universität Berlin (FU Berlin)
Takustrasse 9
14195 Berlin
Germany
Contact person: Robert Tolksdorf
E-mail address: tolk@inf.fu-berlin.de

Free University of Bozen-Bolzano (FUB)
Piazza Domenicani 3
39100 Bolzano
Italy
Contact person: Enrico Franconi
E-mail address: franconi@inf.unibz.it

Institut National de Recherche en Informatique et en Automatique (INRIA)
ZIRST - 655 avenue de l'Europe -
Montbonnot Saint Martin
38334 Saint-Ismier
France
Contact person: Jérôme Euzenat
E-mail address: Jerome.Euzenat@inrialpes.fr

Centre for Research and Technology Hellas / Informatics and Telematics Institute (ITI-CERTH)
1st km Thermi - Panorama road
57001 Thermi-Thessaloniki
Greece. Po Box 361
Contact person: Michael G. Strintzis
E-mail address: strintzi@iti.gr

Learning Lab Lower Saxony (L3S)
Expo Plaza 1
30539 Hannover
Germany
Contact person: Wolfgang Nejdl
E-mail address: nejdl@learninglab.de

National University of Ireland Galway (NUIG)
National University of Ireland
Science and Technology Building
University Road
Galway
Ireland
Contact person: Christoph Bussler
E-mail address: chris.bussler@deri.ie

The Open University (OU)
Knowledge Media Institute
The Open University
Milton Keynes, MK7 6AA
United Kingdom
Contact person: Enrico Motta
E-mail address: e.motta@open.ac.uk

Universidad Politécnica de Madrid (UPM)
Campus de Montegancedo sn
28660 Boadilla del Monte
Spain
Contact person: Asunción Gómez Pérez
E-mail address: asun@fi.upm.es

University of Karlsruhe (UKARL)
Institut für Angewandte Informatik und Formale
Beschreibungsverfahren - AIFB
Universität Karlsruhe
D-76128 Karlsruhe
Germany
Contact person: Rudi Studer
E-mail address: studer@aifb.uni-karlsruhe.de

University of Liverpool (UniLiv)
Chadwick Building, Peach Street
L697ZF Liverpool
United Kingdom
Contact person: Michael Wooldridge
E-mail address: M.J.Wooldridge@csc.liv.ac.uk

University of Manchester (UoM)
Room 2.32. Kilburn Building, Department of Computer
Science, University of Manchester, Oxford Road
Manchester, M13 9PL
United Kingdom
Contact person: Carole Goble
E-mail address: carole@cs.man.ac.uk

University of Sheffield (USFD)
Regent Court, 211 Portobello street
S14DP Sheffield
United Kingdom
Contact person: Hamish Cunningham
E-mail address: hamish@dcs.shef.ac.uk

University of Trento (UniTn)
Via Sommarive 14
38050 Trento
Italy
Contact person: Fausto Giunchiglia
E-mail address: fausto@dit.unitn.it

Vrije Universiteit Amsterdam (VUA)
De Boelelaan 1081a
1081HV. Amsterdam
The Netherlands
Contact person: Frank van Harmelen
E-mail address: Frank.van.Harmelen@cs.vu.nl

Vrije Universiteit Brussel (VUB)
Pleinlaan 2, Building G10
1050 Brussels
Belgium
Contact person: Robert Meersman
E-mail address: robert.meersman@vub.ac.be

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to writing parts of this document:

Centre for Research and Technology Hellas
École Polytechnique Fédérale de Lausanne
France Telecom
Free University of Bozen-Bolzano
Freie Universität Berlin
Institut National de Recherche en Informatique et en Automatique
Learning Lab Lower Saxony
National University of Ireland Galway
The Open University
Universidad Politécnica de Madrid
University of Innsbruck
University of Karlsruhe
University of Liverpool
University of Manchester
University of Sheffield
University of Trento
Vrije Universiteit Amsterdam
Vrije Universiteit Brussel

Changes

Version	Date	Author	Changes
0.1	2004-09-25	Wolf Siberski	creation
0.2	2004-10-18	Wolf Siberski	changed according to results of Sep 29 meeting
0.3	2004-11-15	Wolf Siberski	integrated partner contributions
0.4	2004-11-21	Wolf Siberski	modified according to Berlin meeting (Nov, 17-19)
0.5	2004-12-01	Wolf Siberski	added contributions from FT, UPM, USFD
0.6	2004-12-08	Wolf Siberski	added contributions from INRIA
0.7	2004-12-23	Wolf Siberski	completed, except conclusion
1.0	2005-01-21	Wolf Siberski	changed according to feedback, added conclusion
1.1	2005-05-31	Wolf Siberski	updated introduction, added updates from USFD and INRIA
1.2	2005-06-28	Wolf Siberski	added updates from FT, UKARL and VUM, updated conclusion
1.3	2005-06-30	Wolf Siberski	minor corrections and additions

Executive Summary

To foster the use of Semantic Web Technology within commercial software projects, solutions covering all aspects of semantic applications have to be provided. Of course all of these solutions have to fit together, thus forming a framework for application development and deployment. This deliverable describes requirements for the most important ontology-based tools and components. The collected requirements will serve as input for the definition of the Semantic Web Framework, and can also be used as source for evaluation criteria. This work builds on the results of the use-case analysis conducted in WP1.3 which yielded a list of high-level knowledge-processing tasks and components. All primary tasks identified in deliverable D1.1.3 "'Knowledge Processing Requirements Analysis"' are covered in this document.

Requirements for the the following tools and components are collected:

- Ontology Development
  - Ontology Editor
  - Ontology Integration
  - Ontology Transformation
  - Ontology Extraction/Mining
- Data Layer Components
  - Semantic Query Processor
  - Reasoner
  - Wrappers to existing Information Sources
- Interface Layer Components
  - Annotation and Instance Editor
  - Semantic Web Service Infrastructure

The main characteristics and requirements for each of these potential framework components are presented. For illustrative purposes, some existing approaches, solutions and systems which realize these components are also described.

Contents

Chapter 1

Introduction (by L3S)

A crucial factor for industry acceptance of a new IT technology is the availability of stable and interoperable tools and components for application development. The goal of task 1.2.2 is to identify these components and describe their functionality and requirements. We describe our findings here. The component identification and structuring is based on three sources:

- A use-case analysis conducted in work package 1.3 (see 1.1)
- The application of the common three-tier application architecture (see 1.2).
- The survey of ontology-based tools and components conducted in the OntoWeb project (see 1.3

We chose a uniform pattern for component/tool description, consisting of three subsections:

- **Expected Functionality** lists all functions this component needs to offer
- **Requirements** describes the specific requirements this component has to fulfill
- **Existing Solutions** contains descriptions of some existing component implementations for illustrative purposes.

To show how this modularization can be applied to an application, we describe how a semantic portal is assembled of the components listed (chapter 5).

In chapter 6 we summarize our findings and present some conclusions.

1.1 Knowledge Processing Tasks and Components according to Use Case Analysis

WP 1.3 collected and analyzed several use-cases of knowledge-processing applications. In deliverable D1.1.3, for each such use case the knowledge processing tasks it requires were identified. They are structured as primary and secondary tasks according to their influence on the architecture of a system. The following knowledge processing tasks were derived from the use cases (the descriptions were adapted here to remove dependencies to use case specifics):

- **Data translation** is a task of translating data from different information sources into RDF exploiting methods which are able to preserve semantics of an information source. *In 2.3.4.1, this term is used to denote the transformation of the data to a global schema. But that task should be defined separately as Ontology mapping*
- **Ontology management** is a task of maintaining the ontologies used in the application. *called 'Global schema management' in 2.3.4.1*
- **Matching** is the task of discovering "correspondences" between different semantic informations (encoded as RDF grphs), e.g. between vacancies and jobseekers in a recruiting system.
- **Ranking matching results** is a task of ordering matching results according to a desired criterion.
- **Content annotation** is the task of supporting the user in adding/editing semantic information for a resource.
- **Reasoning** is the task of evaluating a formalized semantic query based on available ontology and instance information.
- **Semantic query processing** the translation of a user query (specified in an end-user-friendly way to a formal query relating to concepts of the underlying ontologies.
- **Intelligent search and retrieval** is the combination of semantic search and and text/multi-media based retrieval methods to identify most relevant resources.
- **Planning of web services** the process of constructing ad hoc a composite service based on user request, rules and available (atomic) services.
- **Mapping rules definition** the task of determining semantic relations between different ontologies.
- **Result reconciliation** is the process of merging results from different sources to a consistent unified result which can be presented to the end-user.

- **Metadata generation** is the automatic creation of semantic information about a resource based on its content (e.g. keywords)
- **Searching for content providers** the identification of relevant information providers from a query and provider self-descriptions.
- **Content provider's directory management** A provider directory for management of providers. *This is not a task description, but a component description.*
- **Schema/Ontology Merging** is a task of integrating other ontologies into an existing ontology. This task can be considered as an ontology management subtask.
- **Producing explanations.** This is the task of creating a human-readable explanation how a search result was determined to be relevant.
- **Personalization and media adaptation** Adaptation of resource presentation according to user's profile and context (e.g. used device).

It is clear that some of these tasks are highly use-case specific. In chapter 3 of D1.1.3, they were joined to ten general knowledge processing tasks . For each task group, one responsible component was defined. To avoid a bloated requirements analysis, we focus on the following list of primary high level taks and components and omit the rather specific secondary tasks and components for this deliverable (descriptions are directly cited from D1.1.3):

- **Data Translation and Wrapper** This task and component are in charge of translating/exchanging instances between heterogeneous information sources storing their data in different formats (e.g., RDF, SQL DDL).
- **Ontology Management, Schema/Ontology Matching, Merging and Ontology Manager** These tasks and component are in charge of ontology maintenance with respect to (evolving) business case requirements.
- **Matching, Matching Results Analysis, Producing Explanations and Match Manager** These tasks and component are in charge of determining mappings between the entities of multiple schemas/ontologies. The mappings might be ordered according to some criteria. In addition, explanations of the mappings might be also produced.
- **Content Annotation and Annotation Manager** This task and component are in charge of automatic production of content metadata.
- **Reasoning and Reasoner** This task and component are in charge of logical reasoning.

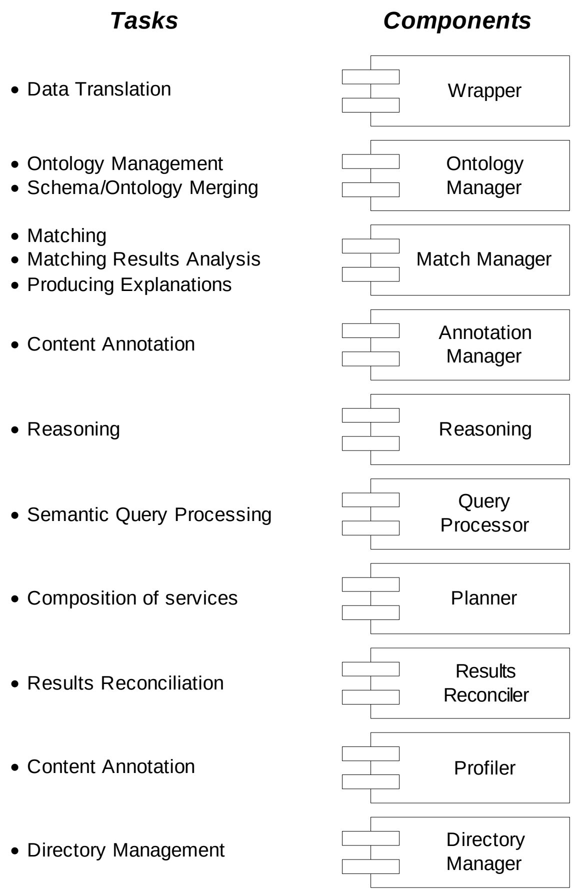


Figure 1.1: Tasks and associated high-level components

- **Semantic Query Processing and Query Processor** This task and component are in charge of interpreting (rewriting) a query by using terms which are explicitly specified in a model of the domain.

- **Composition of Web Services and Planner** This task and component are in charge of automated composition of web services into executable processes.

- **Results Reconciliation and Results Reconciler** This task and component are in charge of determining an optimal solution, in terms of contents (no information duplication, etc.), for returning results from the queried information sources.

The use-case analysis presented here didn't take any architectural considerations into account, because they were out of scope for that work. However, as this document serves as basis for a framework specification, architectural design issues are essential. Therefore it isn't sufficient to just adopt the componentization described above.

1.2 Three-Tier Architecture for Information Systems

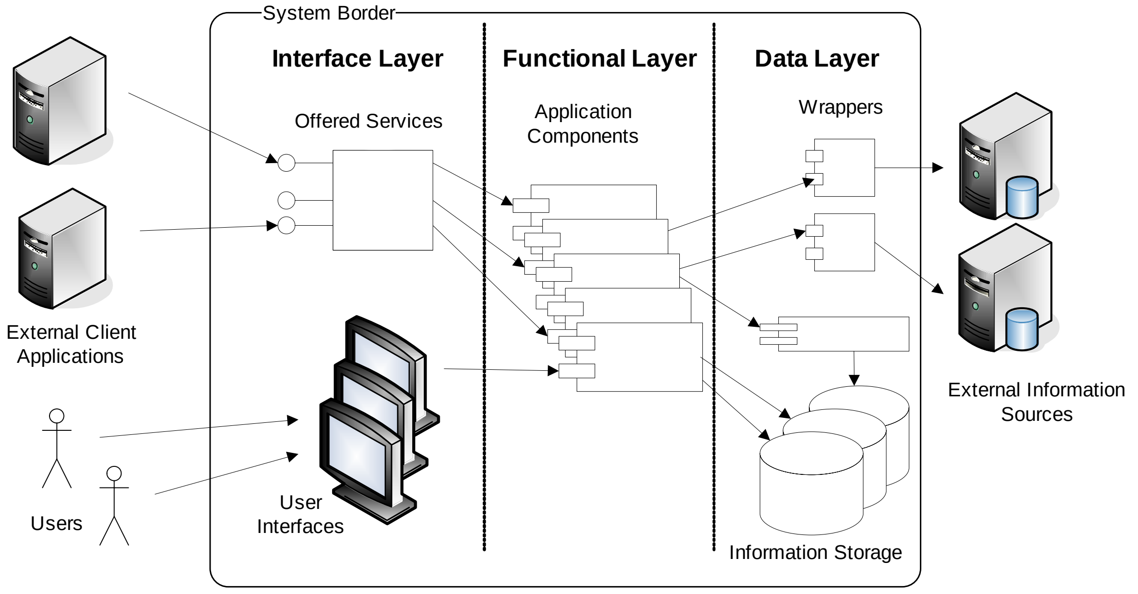


Figure 1.2: Three-tier application

Information systems often are structured according to the common three-tier architecture (e.g. [Züllighoven, 2005]) which consists of the following three layers:

- **Interface Layer** This layer contains all interfaces to the system, user as well as service interfaces. It is introduced to decouple application logic from interaction logic with the system. In case of end-user interfaces, the presentation logic is built using generic user-interface frameworks, e.g. GUI frameworks. In case of interfaces for other applications the modern approach is to provide services, which are built based on a generic service framework (like Web Services).

- **Functional Layer** The application logic layer comprises of domain-specific models and processes. It is the layer with most application-specific code. In case of server-based applications this code is written so that it can be executed within an application server. However, the use of generic components is rather limited in this layer.

- **Data Layer** This layer is responsible for storage and retrieval of the data/information hold within the application. Typically, one or several generic components are used to provide storage and querying functionality. This may be a relational database, an instance store or a repository specialized for some kind of resources like documents. In complex systems, we often find a combination of several such stores. The data layer also may contain wrappers to external information sources which translate requests from the functional layer to the form required by the external systems.

When applying this architecture to ontology-based applications, some modifications are in order. In this case, a large part of the application logic is not programmed in an imperative language, but specified within the used ontologies (and optionally additional rules). Therefore the distinction between functional and data layer becomes blurred. This blur manifests particularly in the fact that a reasoner component is necessary for application logic (functional layer) as well as for query processing (data layer).

1.3 Ontoweb Ontology Tool Survey

In the Ontoweb deliverable D1.3 [Gómez-Pérez, 2004b], the following types of tools have been identified:

- **Ontology development tools** Tools for creating and maintaining ontologies
- **Ontology merge and integration tools** Often, the need arises to combine ontologies or to translate between them. These tools support such activities.
- **Ontology evaluation tools** help in assesing the quality and suitability of an ontology.
- **Ontology-based annotation tools** applications and components which allow end-users to create ontology instances.
- **Ontology storage and querying tools** generic components for storage and querying of ontologies and ontology instances.

A similar classification is found in [Gómez-Pérez, 2004a].

1.4 High-level Tools and Components

Not all taks identified in D1.1.3 are responsibilities of the application system. Some, like ontology maintenance or ontology mapping are rather done at development time, using development tools. Therefore we split our component list in tool components for development and application components for deployment.

For the modularization of development tool components, we follow the existing pattern documented in the Ontoweb tool collection and split into the following components:

- Ontology Editor

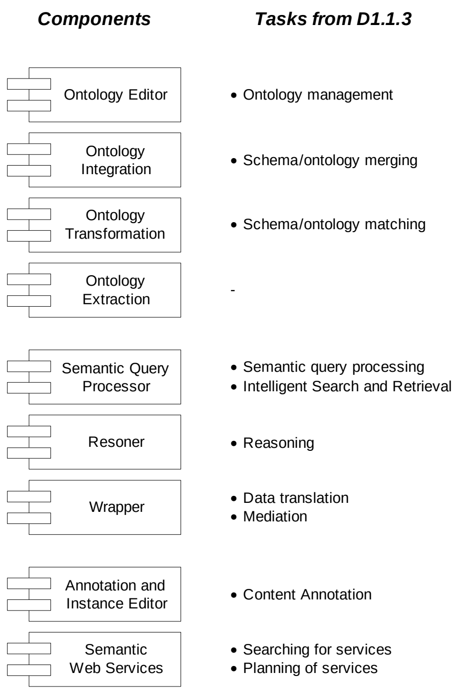


Figure 1.3: Components and their responsibilities

- Ontology Integration
- Ontology Transformation
- Ontology Extraction/Mining [1]

For the identification and classification of application components, we take the use-case analysis results and the architectural considerations presented in 1.2 into account. When comparing the components listed in 1.1 with the architecture described in 1.2, we notice that all of these components belong either to the interface or data layer. We split into the following components (some Components are renamed in compared to D1.1.3, as indicated below):

- Data layer components
  - Semantic Query Processor
  - Reasoner [2]
  - Wrapper
- Interface layer components

[1]not identified in D1.1.3

[2]could have been in the functional layer as well

- Annotation and Instance Editor
- Semantic Web Service Infrastructure [3]
  - Service Directory
  - Service Discovery
  - Service Composition

Fig. 1.3 shows how the primary tasks identified in D1.1.3 and the components identified here are related.

[3]only service composition has been identified in D1.1.3 (as 'Planner'), but this doesn't work without directory and discovery components.

Chapter 2

Ontology Development

2.1 Ontology Editor (by UKARL)

2.1.1 Expected Functionality

Ontology Editors are applied by engineers modelling a specific domain. Therefore several functionalities are expected from an engineering environment which allows for inspecting, browsing, codifying and modifying ontologies.

Modelling ontologies means modelling at a conceptual level, viz. (i) as much as possible independent of a concrete representation language, (ii) using graphical user interfaces (GUI) to represent views on conceptual structures conceptual structures, i.e. concepts ordered in a concept hierarchy, relations with domain and range, instances and axioms, rather than codifying conceptual structures in ASCII.

Main functionalities which an ontology editor should provide are as follows:

Create (new) Ontology

Obviously, an ontology editor should provide methods for creating new ontologies. For doing so an ontology editor should provide this functionality in an intuitive and easy way for its users. Optional, additional help is given or even a wizard guides new users through the creation process asking necessary parameters.

Modify Ontology Entities

Working on ontolgies means modifying its structure (concepts, properties, instances and axioms) for which standard editing methods are needed, e.g. *add, delete* and *edit*.

Visualization

Engineering should be provided within a Graphical-User-Interface based on a visualisation technique for ontologies. One important aspect is to handle large ontologies. Their visual representation needs special attention, since a simple graph-based visualisation might be difficult, e.g. when browsing thousands of concepts.

Search and Query

For large engineering processes functionalities like searching and querying the ontological models are expected.

Undo / Redo Functionality

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

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

It is obvious that for each elementary change there is exactly one inverse change that, when applied, reverses the effect of the original change. Based on the infrastructure described in the previous section, it is not hard to realize the requirement for reversibility of ontology engineering actions and to provide an appropriate undo/redo functionality: To reverse the effect of some extended sequence of changes, a new sequence of inverse changes in reverse order needs to be created and applied.

In other words, reversibility means undoing *all* effects of some change, which is in general not the same as requesting an inverse change manually.

2.1.2 Requirements

Scalability

When speaking about working with an ontology, it is important to emphasize that the ontology may be stored in different ways depending on the intended application or usage scenario. If a *really* huge ontology with lots of concepts and instances shall be created, the usage of a database for ontologies is advisable, since it employs functionalities like a relational database system to store all entities involved. Hence supporting storage of ontologies in main memory and in large databases is a requirement for an ontology editor.

Extensibility and Evolution

Industrial and academic environments are very dynamic, inducing changes to application requirements. Using an ontology-based system, often the underlying ontology must be evolved in order to adapt to those changes. As ontologies grow in size, the complexity of change management increases, thus requiring a well-structured ontology evolution process. It employs so-called evolution strategies that encapsulate certain policies for evolution with respect to the user's demands (see [Stojanovic *et al.*, 2002a] and [Stojanovic *et al.*, 2002b]).

Interoperability

An ontology editor should be able to import and export different representation languages of ontologies, e.g. RDF(S) or OWL. It is obvoius that an editor has to support existing standards and ontology languages as many as possible.

Metadata Generation

An ontology editor should support metadata standards for sharing and re-use of ontologies. So that users are able to specify *manuallay* or *semi-automatically* metadata about their modeled ontologies. As far as possible most of the attributes should be filled automatically by an editor itself (e.g. editor name, creation date, etc.). The generated metadata can be inserted into the modeled ontology itself or be submitted to an ontology metadata repository, like Onthology[1].

Consistency

Necessary, modeling ontologies requires consitency of all operations.

Internationalization

Ontologies are shared across the World within different countries and different cultural groups. Dealing with different languages and character-encondings is a strong requirement.

[1] see Onthology at http://www.onthology.org

Usability

Usability aspects from an end-user point of view is essential for acceptence and usage. Indeed, usability often decides about wether an application is used or not. Hence, an editor should be designed carefully considering usability aspects while the complete development process of a particular editor and while a productive life-time as well.

Collaboration

Ontologies are generally developed by a group of users (maybe distributed among the world) collaborating together. This aspect has to be reflected and supported by an editor in such as it allows *several* users *transparently* working on an ontology over *time*.

Reusability

Ontology editors are required to be re-usable to be applied on different ontologies and ontology languages by different users.

2.1.3 Existing Applications

Ontology Editor OI-Modeler

OI-Modeler is KAON's tool for ontology creation and ontology maintenance[2]. The OI-Modeler's main goal is to allow scalability for editing large ontologies and to incorporate some basic usability issues related to ontology management and evolution.

The goal of this section is to introduce the reader to the OI-Modeler's main features. For further details on how to work with the OI-Modeler we refer to "OI-Modeler User's Guide" [Karlsruhe, 2002].

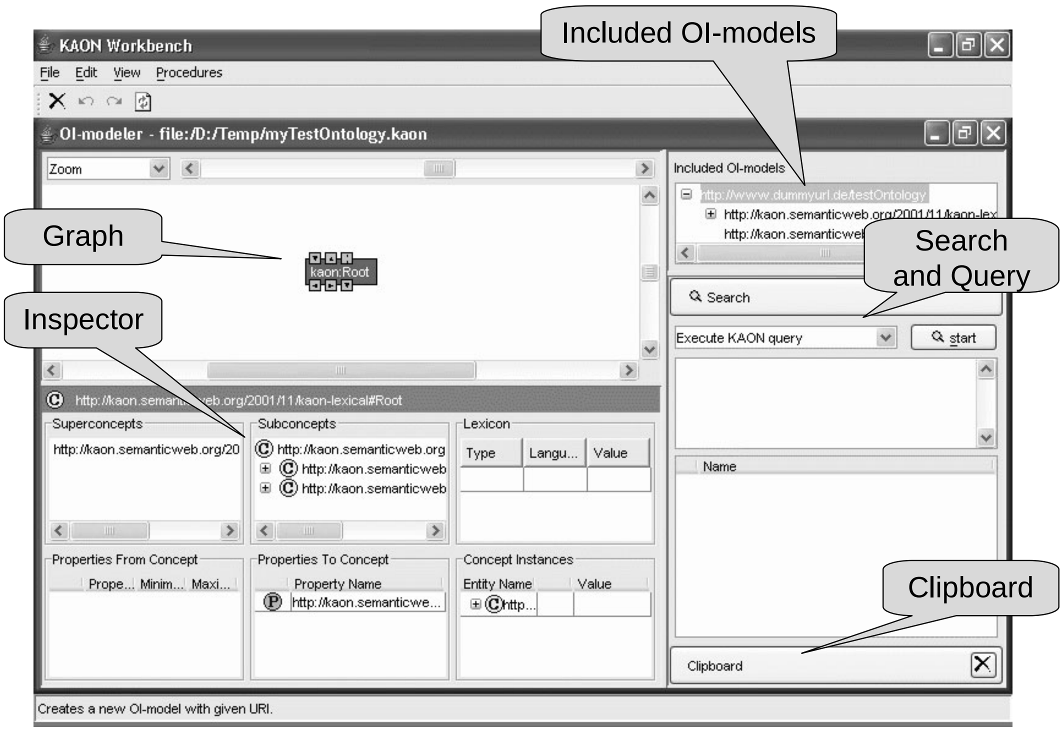
Create New OI-Model After having launched the KAON Workbench the user may work with the OI-Modeler, KAON's ontology editor.

As illustrated in Figure 2.1, the OI-Modeler provides different views on the Ontology and allows to inspect its components (concepts, instances, properties and lexicon).

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

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

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



Creates a new OI-model with given URI.

Figure 2.1: Main Window of the OI-Modeler

Each graph node features up to six little arrows (see Figure 2.2). By clicking on those arrows related entities can be expanded, so that the user can successively browse through the ontology. For example, for a concept the user may expand that concept's sub- and super-concepts, properties to and from this concept, the concept's instances as well as its spanning instances. Regarding the notion of spanning instances please refer to [Maedche *et al.*, 2002].

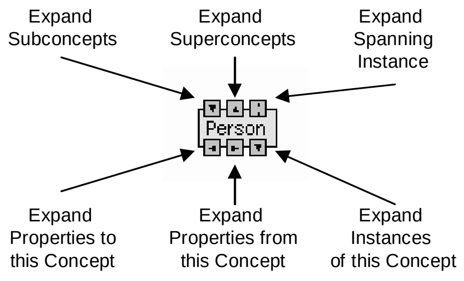

Expand Subconcepts
Expand Superconcepts
Expand Spanning Instance
Person
Expand Properties to this Concept
Expand Properties from this Concept
Expand Instances of this Concept

Figure 2.2: Characteristics of Nodes in OI-Modeler's Graph Visualization

Inspector In the inspector you can find all information about the ontology entity that is currently selected in the graph. Thus, the inspector's appearance adapts to the type of entity (concept, property, or instance) currently selected. If, for example, a concept is selected information about that concepts and its super- and subconcepts are displayed, also the properties to and from the concept and the concept instances. Furthermore, the inspector may be used to directly create new (sub-)concepts (see below).

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

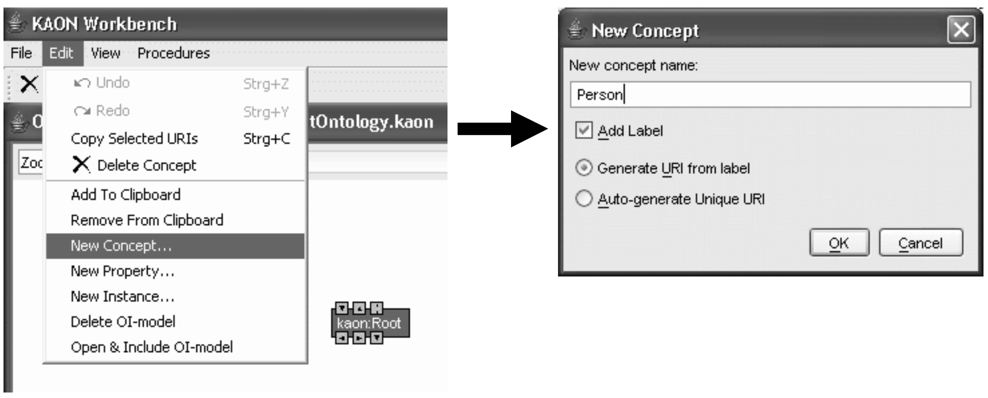
Search and Query With the search function, one can easily find different named nodes. It is possible to search for concepts, instances, and properties and to perform a keyword-based search for any matching item.

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

Exemplary: Add New Concept OI-Modeler provides three ways to create a concept. You can add new concepts by

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

Figure 2.3 illustrates the first of the three above-mentioned ways.



New concept name: Person

[✓] Add Label

[✓ Generate URI from label] Auto-generate Unique URI

Figure 2.3: Adding a Single Concept *Person*

Sub-concepts are thematic refinements of concepts. When intending to add a sub-concept c_s to an existing concept c, first concept c has to be selected – as a consequence, details regarding that concept are displayed in the Inspector. Now, the sub-concept can be added to c with one of the three possible ways mentioned before. In Figure 2.4 the third alternative (using the context menu in the Inspector) is shown.

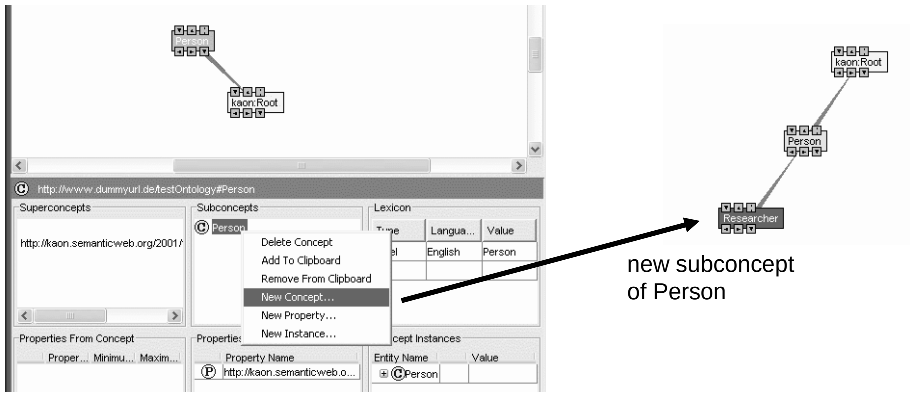


Figure 2.4: Adding Sub-Concept *Researcher* to Concept *Person*

To summarize, OI-Modelerhas been applied successfully in several research and industry projects and can be freely downloaded from kaon.semanticweb.org.

WebODE

WebODE is an ontological engineering workbench developed by the Ontology Engineering Group at Universidad Politécnica de Madrid (UPM) [Gómez-Pérez, 2004a]. The current version is 2.0. WebODE is the offspring of the ontology design environment ODE, a standalone ontology tool based on tables and graphs, which allowed users to customize the knowledge model used for conceptualizing their ontologies according to their KR needs. Both ODE and WebODE give support to the ontology building methodology METHONTOLOGY. Currently, WebODE contains an ontology editor, which integrates most of the ontology services offered by the workbench, an ontology-based knowledge management system (ODEKM), an automatic Semantic Web portal generator (ODESeW), a Web resources annotation tool (ODEAnnotate), and a Semantic Web services editing tool (ODESWS).

Architecture WebODE has been built as a scalable, extensible, integrated workbench that covers and gives support to most of the activities involved in the ontology development process (conceptualization, reasoning, exchange, etc.) and supplies a comprehensive set of ontology related services that permit interoperation with other information systems. WebODE is platform-independent as it is completely implemented in Java. To allow scalability and easy extensibility, it is supported by an application server so that services can be easily created and integrated in the workbench by means of a management console. One important advantage of using this application server is that we can decide which users or user groups may access each of the services of the workbench.

The core of the WebODEs ontology development services are: the cache, consistency and axiom services, and the ontology access service (ODE API), which defines an API for accessing WebODE ontologies. One of the main advantages of this architecture is that these services can be accessed remotely from any other application or any other instance of the workbench. WebODE ontologies are stored in a relational database so they can manage huge ontologies quite efficiently. WebODE also provides backup management functions for the ontologies stored in the server. The interoperability services import ontologies from XML, XCARIN, RDF(S), DAML+OIL, and OWL; and export ontologies to XML, FLogic, XCARIN, RDF(S), OIL, DAML+OIL, and OWL. Ontologies are also exported to languages that are not specifically created for defining ontologies such as Prolog, Jess [4], and Java. For instance, the Prolog export service is used as a basis of the WebODEs inference engine.

Knowledge model Ontologies in WebODE are conceptualized with a very expressive knowledge model. This knowledge model is based on the reference set of intermediate representations of the METHONTOLOGY methodology. Therefore, the following ontology components are included in the WebODEs knowledge model: concepts and their local attributes (both instance and class attributes); concept groups, which represent sets of disjoint concepts; concept taxonomies, and disjoint and exhaustive class partitions; ad-hoc binary relations between concepts, which may be characterized by relation properties (symmetry, transitiveness, etc.); constants; formal axioms, expressed in first order logic; rules; and instances of concepts and relations. In addition to the previous components, bibliographic references, synonyms, and abbreviations can be attached to any of the aforementioned.

The WebODEs knowledge model allows referring to ontology terms defined in other ontologies by means of imported terms. Imported terms are identified with URIs and these are of two types: those available in another WebODE ontology, either in the same or in a different WebODE server (referred to as webode://WebODE_host/ontologies/ontology#name), and those identified by a different type of URI. WebODE instances are defined inside instance sets. Thus, we can create different instantiations for the same ontology, which are independent from each other. For instance, we can instantiate our travel ontology in different instance sets, one for each travel agency using the ontology.

Ontology editor The WebODE ontology editor is a Web application built on top of the ontology access service (ODE API). The ontology editor integrates several ontology building services from the workbench: ontology edition, navigation, documentation,

[4]http://herzberg.ca.sandia.gov/jess/index.shtml

merge, reasoning, etc. Three user interfaces are combined in this ontology editor: an HTML form-based editor for editing all ontology terms except axioms and rules; a graphical user interface, called OntoDesigner, for editing concept taxonomies and relations graphically; and WAB (WebODE Axiom Builder), for editing formal axioms and rules. Other ontology building services integrated in the ontology editor are: the documentation service, ODEMerge, the OKBC-based Prolog inference engine, and ODEClean.

WebODE includes an inference engine that consists of a Prolog implementation of a subset of the OKBC protocol primitives. Since WebODE ontologies can be translated into Prolog, the inference engine obtains an ontology in Prolog from the Prolog export service and loads it into the Prolog interpreter. With this process, the implemented OKBC primitives can build more complex Prolog programs for being executed in the Prolog interpreter for any purpose. As the rest of WebODE services, the inference engine can be executed not only from the user interface of the ontology editor but also by means of its Java API.

Interoperability There are several ways of using WebODE ontologies inside ontology-based applications. First, they can be accessed from its Java API via a local service or application running on the same computer where the ontology server is installed. This API avoids accessing directly the relational database where ontologies are stored and it includes cache functions to accelerate the access to ontology terms. WebODE ontologies can be accessed not only from inside the local server but also remotely with RMI (Remote Method Invocation) and Web services. Second, ontology export services available in the workbench permit generating WebODE ontologies in XML and in several other ontology languages such as: RDF(S), OIL, DAML+OIL, OWL, XCARIN and FLogic. Translations into Prolog can be used similarly. Third, ontologies can be transformed into Protégé-2000. So we can use them inside the Protégé-2000 ontology editor or use the interoperability capabilities provided this tool. Finally, WebODE ontologies can be transformed into Java. In this process, concepts are transformed into Java beans, attributes into class variables, ad-hoc relations into associations between classes, etc., with their corresponding constructors and methods to access and update class variables, etc. This Java code can be used to create other Java applications, uploaded in rule systems like Jess, etc.

2.2 Ontology Integration (by INRIA)

2.2.1 Expected Functionality

Ontology integration consists in solving interoperability problems by using ontologies in a joint environment. This can be achieved by either merging the ontologies into one new ontology, ”bridging” the new ontologies by using axioms for expressing the relationships between element in both ontologies. In both cases, the correspondences between the two ontologies must have been found. This work is considered in depth in Knowledge web work package 2.2.

Merging ontologies can be used at design time when editing a new ontology as well as at run-time when data expressed under some ontology is to be used in another environment. Bridges can be used within another ontology or in mediators which will dispatch queries and answer to various resources. In this last case, the bridge is used as a transformation (see below).

Ontology integration can, as well cover the alignment process (finding the correspondences between the ontologies).

Main functionalities which ontology integration may provide are described below. These functionality are :

Matching ontologies

Matching ontologies consists in finding the correspondences between two ontologies. The result of matching (the alignment) can be exploited in a further step for merging, bridging or transforming ontologies. The specification of this operation is given in deliverable D2.2.1.

Current systems however are usually targeted towards one operation (merging, transforming, translating, etc.). They thus do not identify any specific matching step.

In order to be interoperable and plugable, it is necessary that the matching step provides the alignment in some standard format (e.g., the alignment API [Euzenat, 2004]) and that the other operations are able to take these as input. The Alignment API already provides many different generators that can be used in this further step.

The matching step is certainly the one that is the most difficult to perform automatically, for that purpose, it is often implemented as an interactive process by which the user is presented with one or several possible candidate alignments that the user can alter and ask the system to improve the resulting alignment.

Merging ontologies

Merging consists in taking two ontology and providing as input an ontology which merges both of these ontologies. Usually, this results consists in the import of the two previous ontologies and a set of bridge axioms expressing the alignment discovered in the previous step.

Transforming ontologies

Transforming ontologies take two ontologies and transform one of then in an ontology that is expressed in function of the other. This operation is more often used when ontologies are written in different ontology representation languages. However, it can also be used for replacing terms and identifiers of one ontology by those of another ontology. This can be helpful when adopting a newly standardised ontology or before merging the ontologies (in the last event, no bridge axiom would be needed).

Ontology transformation is moreover the topic of setion 2.3.

Translating statements

Translating statements consists in taking some assertions written in function of one ontology and rewriting them in function of the other ontology. This translation is based on the alignment which provides the rules for the translation program.

Generating mediators

Generating mediators and most often query mediators consists in generating two translation functions, the first one allows to translate queries from one ontology to another and the other to translate answers from the other to the first one.

2.2.2 Requirements

Scalability

Ontology integration should be able to support large ontologies. This is especially a problem when the ontologies are actually merged for providing a new ontology. However, this can reveal difficult as well when the use of bridging axioms leads to longer access time due to indirections (this can even be worse depending on the complexity of the axioms).

Scalability can also affect the alignment process itself where alignment algorithms working on toy problems do not scale real world one.

Distribution

The choice of merging ontologies can be wise when connectivity is not guarantee. In this case, it is better to upload the ontologies and merge them. On the contrary, axiom or mediator approaches have been especially designed for dealing with distributed systems: they leave the ontologies on the servers and only store the information for taking dynamically advantage of distributed ontologies.

Version/Configuration Management

Once an alignment has been found between two ontologies, and independently of whether it is used in merging or bridging ontologies, if one of the ontology is modified, then the alignment and its concrete form must be updated in order to reflect the new version of the ontology. This incremental service should be provided by ontology integration tools.

Consistency

Usually merging ontologies or generating bridge axioms from alignment is something relatively easy and most of the systems should comply some specification of these actions. Most difficult is the definition of the result of the alignment process. No particular consensus exists on how to formally specify the result of the alignment process which is very often controlled by a human user[5].

Interoperability

The ontology integration component should be able to interoperate with the other components. Fortunately the only requirements that comes to mind here is that they are able to communicate ontologies to each other. The best way to achieve this would be to be based on a common API (the worse would be to to use syntactic form and share the same parser).

It would be useful that the integration component be able to accept plug-ins so that new alignment components could be added to the system and that the system be able to accept externally made alignments. For that purpose a common alignment format have to be defined. Again, the aliready cited Alignment API has been designed for that purpose.

Reusability

The goal of alignments and their instanciation as merged ontologies and bridges is to be used many times. So these systems must not build bridges for answering one query or merge ontologies for just composing two web services.

This is the purpose of the Alignment API [Euzenat, 2004] to be able to reify, publish and reuse alignments in many different contexts.

[5]Knowledge web work package 2.2 is investigating these problems.

2.2.3 Existing systems

OntoMerge

In OntoMerge [Dou *et al.*, 2003], the merge of two ontologies is obtained by taking the union of the axioms defining them, using XML namespaces to avoid name clashes, and adding bridging axioms that relate the terms in one ontology to the terms in the other. Inferences can be conducted in this merged ontology in either a demand-driven (backward-chaining) or data-driven (forword chaining) way. The ontologies in OntoMerge are described in either DAML+OIL or RDFS. OntoMerge is developed on top of PDDAML (PDDL-DAML Translator) and OntoEngine (inference engine). These are built using Jena (from HP Research) and JTP (from SRI/KSL). OntoMerge does not provide any clue for alignment.

Alignment software

There are many alignment algorithms available, and quite a few tools among which Protégé and it Diff suite can be singled out. Please refer to deliverable 2.2.3 for more information about these tools.

2.3 Ontology Transformation (by INRIA)

2.3.1 Expected Functionality

Ontology transformation consists of transcribing an ontology from one form to another. This can be in a different ontology language, a reformulation in a restriction of a language (e.g., expressing automatically some non necessary OWL-Full ontology into OWL-DL) or with regard to a different vocabulary. Ontology transformation is useful for solving heterogeneity problems, when one wants to take advantage, in a particular context of an ontology that have been developed in another context (i.e., using a different language). There are various such situations: for interoperating systems developed under different ontologies like web services, peer-to-peer systems, etc.

Ontology transformation can be produced by tools requiring various power. Some of them can be mere lexical translator, there can be syntactic translators, but most of them will require the power of processing the ontology (i.e., inferring) in order to transform it. It is thus necessary to use the right tool.

The additional function of an ontology transformation system can be described by properties which depends on what is expected from the transformed ontology. Among these properties, one can cite the following:

- subsumption preservation

- consequence preservation
- round tripping

There is various work dedicated to this.

Ontology transformation can be processed as a one-shot pass (for instance when someone wants to import an ontology into her editing environment) or as a routine which can be used as a sub system (for instance in a system which dynamically compose web services).

Ontology transformation

The only function of ontology transformation is ontology transformation! It requires one ontology (written in a particular form) and a specification of a targeted form, and will generate the ontology under the specified form. The form specification can be as simple as the name of a language and as complicated as the set of transformation rules (with several intermediate steps).

It generally operates automatically. User intervention can be needed when identifier generation is involved.

2.3.2 Requirements

Scalability

The transformation systems must be able to cope with arbitrarily large amount of data. This is most often the case for simple systems (syntactical transformation or not very complicated semantic transformations). However, some system or some transformation might require arbitrary inference to be drawn. In such a case, scalability is not guarantee anymore.

In general, scalability is a property of the transformation to be implemented rather than that of the system itself. Any system should be expected to scale easily on a simple transformation.

Distribution

The ability of the system to run on a network of computers. In this context, the most important aspect is the ability to integrate information which is distributed in a network. Part of this requirement is the ability to *localize* relevant information in the network.

Version Management

Transformation systems are not version managers. However, transformations can be used for udgrading/downgrading an ontology (or data expressed with regard to an ontology)

Consistency

Consistency, is one of the most important features of transformation systems and transformations. Consistency can be measured with regard to properties that they must satisfy. We mentioned several such possible properties above. They are important in several respects: because the user must know what to expect when performing a transformation, and also because it allows to compose transformations

Another type of consistency for transformations is type-safety. The fact that an ontology translated from language A to language B is indeed an ontology of language B.

Internationalization

Internationalization affects transformation systems to the extent that they must be able to deal with as many character encodings as possible.

Usability

The degree of efficiency with which end-users can do their tasks with the product, and their overall satisfaction with that process. As usability is evaluated from an end-users point of view, user interface design and implementation are major factors.

Interoperability/Standard-Conformance

So far, transformation systems are tools for helping interoperability. There are no particular standard on that topics[6], so no real conformance to be expected. It should be very useful however, to use some common API allowing to compose transformations. This is espectially true when it is necessary to compose a syntactic language transformation with a semantic transformation provided as an alignment result.

In order to interoperate with other components, the ontology transformation component must be able to deal with the input and output ontologies. Again, the best way to achieve this would be to be based on a common API (the worse would be to to use syntactic form and share the same parser).

[6]However, Knowledge web work package 2.2 is working towards a format for expressing alignment that should be useful for the whole semantic web community and could become a standard

It would be useful that the transformation component be able to accept plug-ins so that new transformation opportunities are offered. For that purpose, the form specification must be clearly defined as well as a way to declare the capabilities of plug ins. No such standard exists so far, but defining one reasonable specification would be useful.

Reusability

It is expected that transformation are reusable. This is the case when they are expressed in a transformation language. Most of the tools starts from such a language.

2.3.3 Existing systems

There are few systems strictly devoted to ontologies. We give example of such systems below and their characteristics, particularly with regard to the capability to express semantic transformations and how these are articulated with semantics.

RDFT

The main goal of RDFT is to represent mappings that can be executed and imported in a transformation process [Omelayenko, 2002]. These mappings correspond to sets of pairs between simple entities (RDFS classes and properties) with a qualification of the relation holding. The ontology is expressed in DAML+OIL. Surprisingly, if the correspondences are generated by Bayes techniques no strength is retained. This format is aimed at using the mapping but no hints are given for adding alignment algorithms or extending the format.

MAFRA

MAFRA provides an explicit notion of semantic bridges modelled in a DAML+OIL ontology [Mädche *et al.*, 2002]. The MAFRA Semantic bridges share a lot with the mapping format presented here: they can be produced, serialised, manipulated and communicated through the web. Moreover, the semantic bridges are relatively independent from the mapped languages (though they can map only classes, attributes and relations). They have, however, been built for being used with the MAFRA system, not to be open to external uses, so the classes of the ontology are rather fixed and cannot easily be extended towards new relations or new kinds of mappings. This format is also tailored to the processing architecture used (with non declarative primitives in the transformations).

Transmorpher

Transmorpher [Euzenat and Tardif, 2002] is a software tool for defining and processing complex transformations of XML documents. It can accept external transformations (e.g., XSLT stylesheet) and provide a simple transformation language offering unit transformations (suppression, renaming, regular expression substitutions and query facilities). In addition to generating, transforming and serializing XML documents, it features constructors like merging, dispatching, querying, iterating, and composing transformations. These transformations can have several input and output streams. New implementation of these constructors can be plugged in Transmorpher. Transmorpher can be used as a compiler, an interpreter, a Ant task, a Servlet generator or embeded in another program.Transmorpher is thus not specific to ontology transformation. However, it has been explicitly designed with property enforcement in mind and preservation of these properties when composing transformations in its use. This has been exploited in the Family of languages approach

Family of languages

The 'family of languages' approach [Euzenat and Stuckenschmidt, 2003] provides a way to use a set of different knowledge representation languages and transformation among them satisfying some properties. The properties considered are mainly semantic properties.

More precisely the family of languages approach is based ona set of knowledge representation languages whose partial orderingdepends on the transformability from one language to another bypreserving a particular formal property such as logicalconsequence. For the same set of languages, there can be severalsuch structures based on the property selected for structuring thefamily. Properties of different strength allow performingpracticable but well founded transformations. The approach offersthe choice of the language in which a representation will beimported and the composition of available transformations betweenthe members of the family.

This approach has been implemented, with a simple family of description logic, on top of the Transmorpher system (see above).

WebODE

WebODE translation system [Corcho, 2004] is the set of import/export method embedded in the WebODE workbench. It is based on a four layer classification of the kind of interoperability problems arising in ontology mismatch: lexical, syntactic, semantic and semiotic. WebODE provides three different languages for expressing the necessary transformations according to the corresponding layer (semantic and semiotic problems are considered within the same layer).

ODELex, ODESyntax and ODESem reuses the way Lex and Yacc can be used as transformation system while offering features that are specific to ontology transformation. A number of actions are described at the level of the representation model (adding classes). These systems allow the use of various kind of ontology representation languages (expressed in their textual form).

The WebODE system offers enough power to consider semantically grounded transformations but cannot account for the semantic by itself.

OntoMorph

OntoMorph [Chalupsky, 2000] is a system of syntactic transformation of ontologies with a syntactic transformation language not very different from XSLT. It however is integrated with a knowledge representation system (PowerLoom, [Valente *et al.*, 1999]) which provides the opportunity to have semantically-grounded rules in the transformations. The system can query assertions for not only being syntactically in the source representation, but also for being a consequence of this initial representation (as soon as PowerLoom is semantically complete). This is a generic implementation of what is proposed in [Stuckenschmidt and Wache, 2000]. Of course, this option requires to use PowerLoom as an initial pivot language and the problem of translation arises when transforming from the source representation to the PowerLoom representation.

Since OntoMorph is able to take advantage of the inference power of PowerLoom, it can implement semantic transformations. These are implemented in an ad hoc manner where the transformation rules have to satisfy the semantic properties (however, these are expressed at a higher level than, say, Transmorpher).

2.4 Ontology Extraction/Mining (by USFD)

2.4.1 Expected Functionality

Systems for ontology extraction / mining should take a corpus of text and (optionally) an existing ontology and should be able to create semantic metadata, either by populating the texts with instances from the ontology or by modifying the ontology with instances from the text. In some cases they may also modify the existing ontology structure or content, for example by adding new concepts or reassigning instances. They tend to rely heavily on Information Extraction techniques.

Traditional IE is not, however, completely suitable for metadata creation, because semantic tags need to mapped to instances of concepts, attributes or relations, and this is not always a straightforward process. Also most IE systems based on Machine Learning methods, e.g Amilcare [Ciravegna and Wilks, 2003] do not deal well with relations. although they are very good at finding entities (which can be mapped to instances in the ontology). On the other hand, there have been few knowledge engineering approaches (which use rule-based systems, e.g. GATE [Cunningham *et al.*, 2002] that deal successfully with relations except in very restricted domains, and these require a substantial amount of time and effort to develop. GATE has been modified to use a combination of learning and rule-based methods, and allows a combination of IE and IR, so that for example, an IR engine can be used to find the relevant texts, before annotation is performed.

2.4.2 Requirements

Interoperability

Interoperability is concerned with how well the tool interacts with other tools and systems. Annotation is a task that is often combined with other applications, such as browsing, search and retrieval, indexing, etc., so it is important that annotation tools can easily interact with other systems. This is best achieved by conformance to existing standards.

Metadata is created through semantic tagging, and can be represented as inline or standoff annotation. Inline annotation means that the original document is augmented with metadata information, i.e. the text is actually modified. Standoff annotation, on the other hand, means that the metadata is stored separately from the original document, with some kind of pointers to the location of the corresponding text in the document. This can be either in the form of a database or as e.g. an XML file. For ontology creation or enhancement, standoff annotation method is generally much better, because the text itself is unimportant, rather it is the information gleaned from the text that is interesting.

Both methods are acceptable from the point of view of interoperability; however, standoff annotation is generally preferable, for the reasons mentioned above, as long as a standard form is used, such as TIPSTER format, or provided that a means of export to a recognised format is provided. This is the problem with inline annotation, because it is difficult to manipulate the annotations once created.

Interoperability evaluation not only covers annotation format, but also issues such as:

- data format: what kinds of text format can be processed, e.g. xml, html, sgml, txt, etc.;
- annotation schemes: whether annotation schemes can be imported/exported from other tools;
- plugins: if it is possible to plug in extensions;
- converters: if converters to/from other formats are provided if non-standard formats are used

Scalability

Semantic metadata creation can be manual, semi-automatic, or fully automatic. Manual creation is slow and time-consuming, and is therefore unsuitable for large-scale annotation. Semi-automatic methods save time and money, and, like manual methods, are very reliable, but they suffer from the drawback that they represent only one view (that of the user, with respect to a single ontology). They also still require human intervention at some stage in the process (either to train the system by providing initial manual annotation before the system takes over, and/or to verify and correct the results produced by the system). There is almost always a tradeoff between the level of automation, the size of the corpus, and the quality of the final output. Systems which perform well and on large documents are unlikely to be fully automatic; systems which are fully automatic may be able to handle large documents but with lower performance.

Reusability

Ideally. systems should be reusable in a wide variety of contexts, i.e. they should work on different kinds of domains and genres. Semi-automatic systems which rely on some degree of manual annotation and/or training can usually be adapted to new domains and ontologies, but will need retraining by the user. This means that they are generally best suited to annotating large volumes of data within a single domain, and in situations where the user has an interest in investing some initial time and effort in the application. They are less suitable for the casual user who wants a ready-made tool to provide instant annotations for his data. Automatic methods, on the other hand, can represent many diferent views, and they change according to the ontology in qqustion. The IE engine can be retrained for each ontology, and, furthermore, if the ontology changes, they remain up-to-date because the metadata can be regenerated dynamically. However, the tradeoff is that their performance is generally much lower.

2.4.3 Existing systems

KIM

KIM [Popov *et al.*, 2004] offers an end-to-end, extendable system which addresses the complete cycle of metadata creation, storage, and semantic-based search and includes a set of front-ends for online use, that offer semantically enhanced browsing. KIM contains an instance base (KIMO) which has been pre-populated with 200,000 entities.KIM has a special ontology enrichment stage where new instances found in the text are added to the ontology. This often involves a disambiguation step, because many instances could be added in more than one place. For example, ”Paris” could be an instance of the country France or the state of Texas. The disambiguation process uses an Entity Ranking algorithm, which involves priority ordering of entities with the same label based on corpus

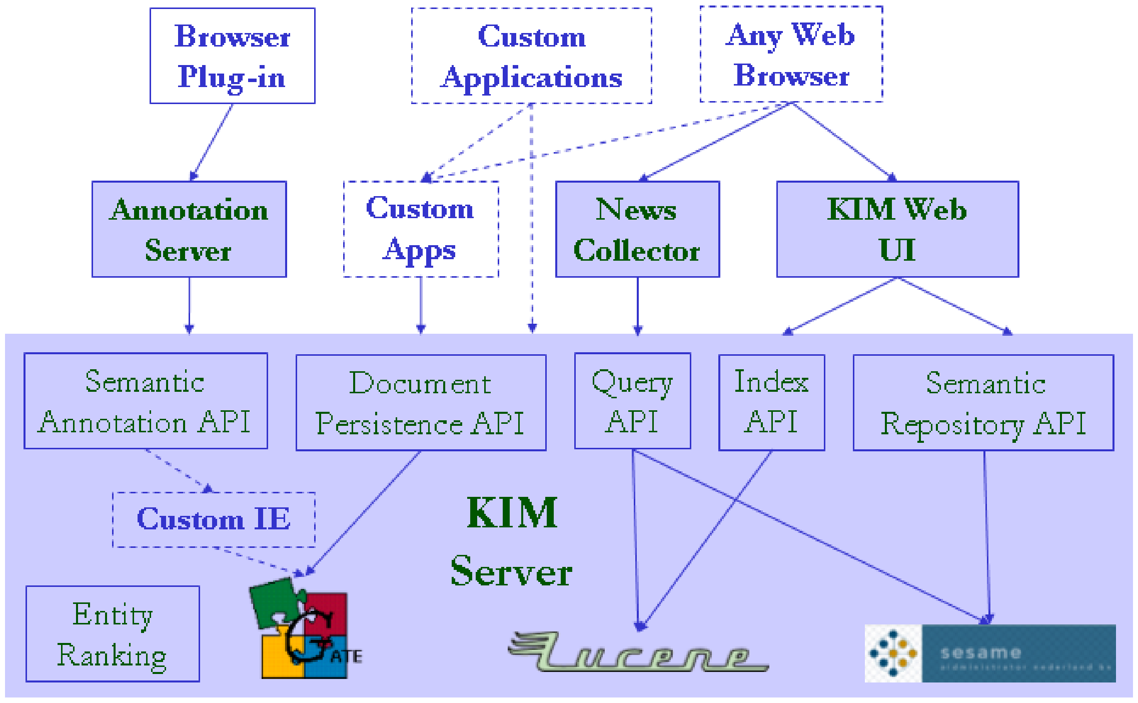


Figure 2.5: Architecture of KIM

statistics.

The essence of the KIM IE engine is the recognition of named entities with respect to the KIM ontology, which is achieved using ANNIE [Maynard *et al.*, 2002]. The entity instances all bear unique identifiers that allow annotations to be linked both to the entity type and to the exact individual in the instance base. For new (previously unknown) entities, new identifiers are allocated and assigned; then minimal descriptions are added to the semantic repository. The annotations are kept separately from the content, and an API for their management is provided.

The architecture of KIM (shown in Figure 2.5 consists of the KIM ontology, a knowledge base, the KIM server (with an API for remote access, embedding and integration), and front-ends (a browser plugin for Internet Explorer, the KIM web user interface with various access methods, and the Knowledge Explorer for navigation of the knowledge base). KIM relies on GATE, SESAME and Lucene, but all these, as well as the software platform itself, are domain and task independent.

h-TechSight

The h-TechSight Knowledge Management Platform (KMP) [Maynard *et al.*, 2004] aims to extract new domain data from free text on the web. It uses GATE [Cunningham *et al.*, 2002] to power the concrete data-driven analysis of concepts and in-

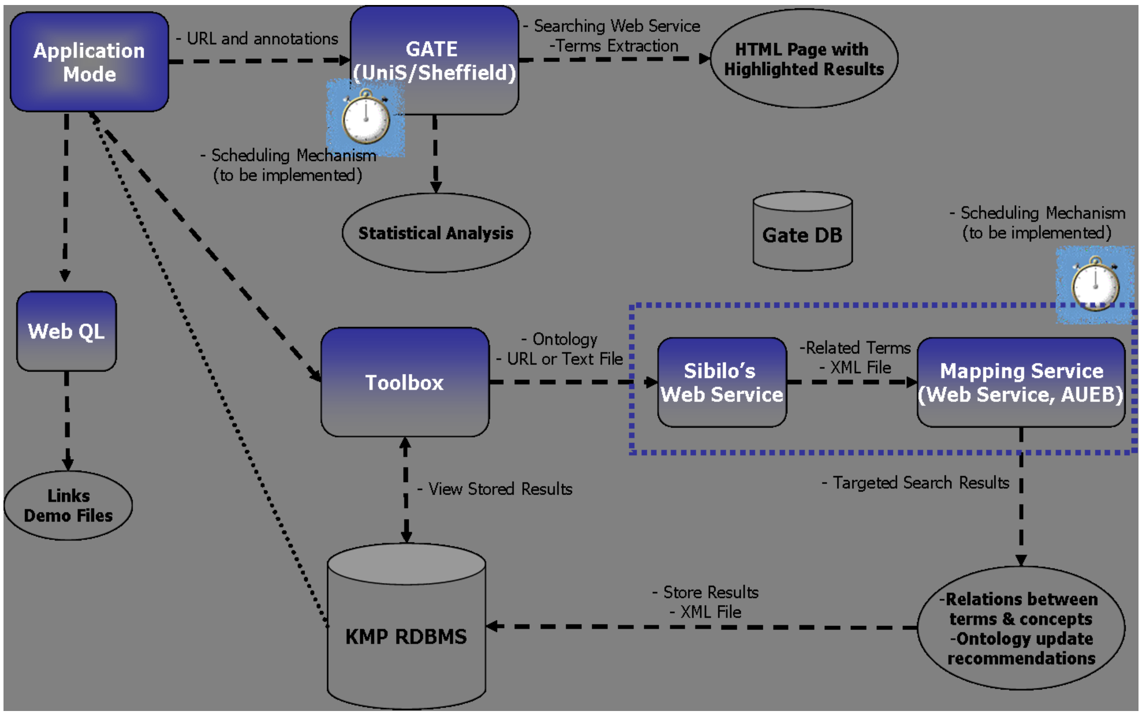


Figure 2.6: Architecture of h-TechSight Application Mode

stances in the knowledge management platform, with respect to an ontology and domain. The GATE IE application enables statistical information to be gathered about the data collected, and inferences drawn, which in turn leads to the monitoring of trends of new and existing concepts and instances.

The application uses two main inputs: a web mining application which feeds relevant URLs to GATE, based on the user's query, and a domain ontology. The texts are automatically annotated with semantic information based on the concepts in the ontology. When an instance of a concept is found, it is annotated with semantic metadata. Instances in the text can not only be visualised (through colour-coding) but are also output in two forms: into a database for further processing, and in ontological form. On the one hand, this annotation of semantic metadata enriches the text; on the other hand, the ontology may be enriched through its population with instances from the text.

h-TechSight performs metadata generation and ontology population (by adding new instances to the ontology), but also by enabling the process of ontology **evolution**. By this we mean that the IE application serves not only to populate the ontology with instances, but also to modify and improve the ontology itself on the conceptual level. Statistical analysis of the data generated can be used to determine how and where this should take place. For example, a set of instances will be linked to a concept in the ontology, but this concept may be too general. A clustering algorithm can be used to group such instances into more fine-grained sets, and thereby lead to the addition of new subconcepts in the hierarchy. hTechSight is also unique in performing monitoring of the data over time, which can also lead to suggested changes in the ontology.

The architecture of the KMP consists of two modes: the generic search mode and the application (targeted search) mode. Here we are only concerned with the application mode. This is comprised of 3 independent but related modules: GATE, Toolbox and the Mapping Service. The GATE module is a web service that takes as input a URL and a set of annotation types specified by the user, and outputs an HTML page with highlighted results. It also stores the results in a database which are then used by another submodule for statistical analysis. The Toolbox and Mapping Service are independent of the GATE module, but interoperate with each other. Sibilo extracts terms from the ontology and URL, and passes these to the Mapping Service which associates these with the concepts in the ontology and provides recommendations for updating the ontology. Essentially, these 2 modules perform a similar function to the GATE module, but in a different way. The GATE module, in which we are most interested here, is therefore fully interoperable with other systems and architectures, since it is not dependent on the other modules in the platform.

MnM

MnM [Motta *et al.*, 2002] is an annotation tool which provides both automated and semi-automated support for annotating web pages with semantic contents. MnM integrates a web browser with an ontology editor and provides open APIs to link to ontology servers and for integrating information extraction tools.

The architecture of MnM consists of a set of plugins, all of which implement a generic interface. Two abstract classes implementing the IEPlugin interface are provided: the IEEnginePlugin defines the methods required to deal with the IE mechanism, while the IEOntologyPlugin provides the methods needed to interact with ontologies. Thanks to these classes, the plugin developer only has to extend them and override the main methods. Plugins are contained in jar files.

Chapter 3

Data Layer Components

3.1 Semantic Query Processor (by VUM)

3.1.1 Expected Functionality

The data access module should primarily provide functionality that allows

- access to ontologies and knowledge bases, independent of the particular data representation, such as RDF;
- access to RDF and OWL data sets;
- support both distinguished and undistinguished variables;
- query answering over RDFS/OWL DL ontologies;
- multiple users to access data simultaneously in a safe and consistent manner.

3.1.2 Requirements

The achieve those functionalities the data access and persistence modules must support

- *data modularity*, by allowing to compose distributed data;
- *transactional changes*, by ensuring the commonly-known ACID properties;
- *separation of concerns* by realizing separate interfaces for the manipulated entities;
- *adhering to formal semantics* by being able to ensure that a given data set is valid;

- *adhering to the intention of change* by ensuring that changes have the intended effect;
- *change notification* by providing mechanisms that notify other users of changes;
- *human-readable presentation* by providing a lexical layer on top on abstract URIs.

3.1.3 Existing Systems

Jena

The Jena project [McBride, 2001] provides both an ontology API and an RDF API, which are separate software entities. Jena allows access to RDF and OWL data sets, it passes our test on query answering over RDFS ontologies. Its API can also rely on an external OWL DL reasoner (such as Racer) by means of a DIG interface: it sends the whole information to the reasoner, which will send back to the Jena API the results of the inference.

Sesame

Sesame [Broekstra *et al.*, 2002] allows access to RDF data sets, it also passes our test on query answering over RDFS ontologies. However, it only has a limited support for OWL, in particular it currently implements a set of rules that cover the DLP subset of OWL DL. Support for a more expressive fragment of OWL is already planned, but yet an open issue.

Triple

Triple [Sintek and Decker, 2002] supports both RDFS and DAML+OIL data sets, but not OWL DL. In terms of query answering, Triple allows only for inference over RDF(S) taxonomies, but it does not pass our test involving domain/range constraints of properties.

Racer

Racer [Haarslev and Möller, 2003] is a reasoning agent that supports inference over different ontologies, which can be expressed in different languages, including DAML+OIL and OWL DL. It passes most our tests on query answering over RDFS and OWL DL, except queries involving datatypes. It does not support the use of undistinguished variables.

Manchester OWL-QL Server

Manchester OWL-QL Server [Glimm and Horrocks, 2004] supports DAML+OIL and OWL DL. It passes most our tests on query answering over RDFS and OWL DL, except queries involving datatypes (because it does not support datatypes yet). It supports the use of undistinguished variables. It reduces query answering to basic ABox reasoning services provided by Racer.

Apart from the above query engines, the very new KAON2 also supports query answering in OWL DL, although we haven't tested it yet.

A more detailed survey of existing Semantic Web querying systems can be found in D2.5.3 "Report on Implementation and Optimisation Techniques for Ontology Query Systems".

3.2 Reasoner (by VUM)

3.2.1 Expected Functionality

The reasoning module should primarily provide functionality that allows

- the support existing Semantic Web standards, such as OWL DL;
- the support of both object and (customised) datatype constraints of ontologies;
- the support of the DIG interface, the general interface for DL reasoners;
- the support of rule extended ontologies;
- clients to identify the working reasoning engine and its capability.

3.2.2 Requirements

Formal semantics

The formal semantics specified by an ontology must be unambiguous and clear.

Flexibility

A reasoning module should enable flexible use of different ontology languages (DLs) and different datatypes and datatype predicates.

Accessability

A reasoning module should enable loose coupling, allowing access through standard web protocols, as well as close coupling by embedding it into other applications. This should be done by offering sophisticated standard APIs.

Consistency

Consistency of information is a critical requirement of any enterprise reasoning system. Each update of a consistent ontology must result in an ontology that is also consistent. In order to achieve that goal, precise rules must be defined for ontology evolution and components updating ontologies must implement and adhere to these rules. Also, all updates to the ontology must be done within transactions assuring the common properties of atomicity, consistency, isolation and durability (ACID).

Concurrency

It must be possible to concurrently access and modify information. This may be achieved using transactional processing, where objects can be modified at most by one transaction at the time.

Security

Guaranteeing information security means protecting information against unauthorized disclosure, transfer, modification, or destruction, whether accidental or intentional. To realize it, any reasoning services should only be accessible by properly authorized agents. Proper identity of the agent must be reliably established, by employing authentication techniques. Finally, reasoning should be based on accessible parts of the target ontologies.

Mapping

Often, there is a need for handling multiple ontologies. Complete support for handling multiple ontologies shall be given. This means that the reasoning component should allow mapping between heterogeneous ontologies and provide sound and complete inference based on the mapping.

Distribution

We assume that data in the Semantic Web will be distributed. Therefore capabilities for accessing and aggregation of distributed information is required. Means for detecting and working with duplicate data sets are required. For example the same data accessible through with different reasoning services, which may provide distinct and unique services for that data. A Semantic Web data store should provide means to detect that the same data set is used. Also means for querying and composition of distributed data should be provided.

3.2.3 Existing Systems

FaCT and FaCT-DG

FaCT [FaCT, 1998] supports efficient TBox reasoning of OWL DL (without nominals). FaCT [Pan, 2004a] is a datatype group extension of FaCT; it supports customised datatypes and datatype predicates. New datatype checkers can be easily plugged into the system, in order to meet the requirements of different applications.

FaCT++

FaCT++ [FaCT++, 2003] is a re-engineering of the FaCT system; it is written in C++. FaCT++ support efficient TBox reasoning of OWL Lite. It has a very simple support for datatypes.

Racer

Racer [Racer, 1999] support both relatively efficient TBox and ABox reasoning of OWL DL (without nominals). It supports concrete domains, but doesn't support user-defined datatypes and predicates.

Pellet

Pellet (from University of Maryland) supports TBox and ABox reasoning wit OWL Lite. It has a simple support for datatypes.

Hoolet

Hoolet (from University of Manchester) support SWRL, the Horn rule extension of OWL DL.

Many of the exiting DL reasoners (such as FaCT, FaCT++ and Racer) support the DIG interface, which is the general interface for DL systems. Therefore, applications which are conforming to DIG can use any DL reasoners that support DIG.

DIG

DIG/1.1 is effectively an XML Schema for the DIG interface along with ask/tell functionality. Applications can take an OWL DL ontology, either in OWL/RDF syntax or in OWL abstract syntax, and use the OWL-API [Bechhofer *et al.*, 2003][1] to translate it into an ontology in DIG/1.1 syntax. Then the clients (applications) communicate with a DIG/1.1 server (a Description Logic reasoner) through the use of HTTP `POST` request, and the DIG server should return `200 OK`, unless there is a low-level error. As DIG/1.1 was designed for DAML+OUL, it does not completely cover OWL DL, let alone OWL-E [Pan, 2004b] which extends OWL DL with necessary and expressive datatype support. Accordingly, [Pan, 2004a] extends DIG/1.1 to DIG/OWL-E, so as to cover the full features of OWL DL and OWL-E.

Apart from the above reasoning engines, the very new KAON2 also supports DL reasoning, although we haven't tested it yet.

3.3 Wrapper to existing Information Sources - Instance Mining (by UKARL)

As the Semantic Web enhances the first generation of the WWW with formal semantics, it offers new chances and challenges for Web Mining. The intention of Semantic Web Mining [Berendt *et al.*, 2002] is to improve the results of Web Mining by exploiting the new semantic structures in the web. As in traditional Web Mining, one can distinguish between content, structure, and usage mining. However, in the Semantic Web, content and structure are strongly interwoven, hence a distinction between Semantic Web Content Mining and Semantic Web Structure Mining blurs.

In this section we discuss requirements for an instance mining component.

3.3.1 Expected Functionality

A component identifying instances (so called instance mining) on information sources, e.g. the Web, should provide as a result *instances* of a given ontology vs. a given set of concepts as result. Indeed, given an ontology O_1 the result is an ontlogy O_2 whereby $O_1 \subset O_2$ and the number of instances increased for one ore more concepts.

Such component might be able to be combined within a complete knowledge discovery process. Furthermore, a mining component should support a knowledge discovery (KDD) process, which mainly consists of following steps.

1. Data Collection
2. Data Cleaning
3. Data Integration

[1]OWL-API can also translate OWL/RDF syntax into OWL abstract syntax, cf. `http://owl.man.ac.uk/api.shtml`.

4. Data Transformation
5. Data Mining
6. Knowledge Representation

Data Collection

Data can be stored on many different sources such as on the Web, accessible by the hypertext transfer protocol HTTP, or on a local computer. Therefore specialised access methods are required or other components are taken into account.

Data Cleaning, Integration and Transformation

Unfortunately, a lot of documents (e.g. on the web) come in less standardized form, partly containing syntactic errors. Therefore training examples need to be cleaned and tidied up.

In order to identify data patterns and to apply data mining algorithms the collected data has to be transformed into the representation used by the algorithm.

Data Mining

Essentially, a mining component has to provide data mining algorithms to discover common patterns or properties in a given data set.

Data Representation

For deployment and use of the patterns, functions for *accessing* and *browsing* generated patterns and identified instances are expected.

3.3.2 Requirements

Following requirements must hold for a Mining Component.

Accessability

Such component must be accessible through standardized application programmer interfaces (API´s) providing methods for controlling the component and receiving the results.

Supervised/automatic mining

Instance Mining can be done semi-automatic or fully automatically. Both mechanisms must be reflected by such component.

An automatic mining process would take an ontology into account and then start finding new instances in one step. The results are presented in the end. In this case no user interaction is possible which may lead to unwanted results. However, if the user knows the ontology well it might be an effective way of adding instances to an ontology without being involved in the detailed mining process.

On the other hand, in cases where detailed information is missing or the user wants to control the mining process in a step-by-step manner a semi-automatic mining mechanism should be provided. Then the user is asked wether to add instance or not or by re-configuring settings influencing the mining process.

Scalability

Instance Mining as mining in general is often a resource intensive task. Therefore such component must take into account issues like scalability and ensure that large mining processes with thousands of instances can be run on it.

Interoperability

Such component should be able to access several ontology formats or providing at least one well-known standard from which other formats can be transformed.

Usability

Instance Mining should be applied by end-users which are not necessarily domain experts. Hence usability apsects become important for the usage of a mining component.

3.3.3 Existing solutions

An existing system for instance mining is the ontology-based web crawler METIS[2]. METIS takes an ontology as domain knowledge whereby the user selects desired concepts for which the crawler should identify new instances. Then the crawler automatically crawls the web based on a detailed configuration. The results can be exported for further analyses. Normally, an instance mining process is encapsulated within a knowledge discovery process, like it is provided by the workbench ARTEMIS[3].

[2]see http://ontoware.org/projects/metis for details

[3]see http://ontoware.org/projects/artemis

The ARTEMIS Workbench represents a tool for knowledge engineers and industrial practioneers required to handle large and heterogenous sets of documents whereby it provides functionalities of well-known knowledge discovery tools to generate semantic enabled document models to apply them on the Semantic Web.

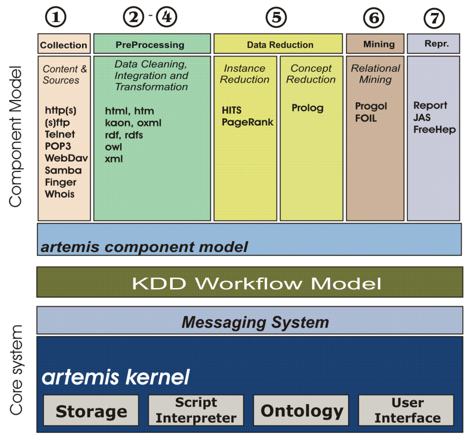


Figure 3.1: ARTEMIS Architecture

The workbench consists of three main blocks: (i) The **ARTEMIS Core System (ACS)** as surrounding technology, (ii) the **Workflow Model (WM)** providing a knowledge discovery workflow and (iii) the **Component Model (CM)** instantiates the workflow by extensible components as presented in figure 3.1.

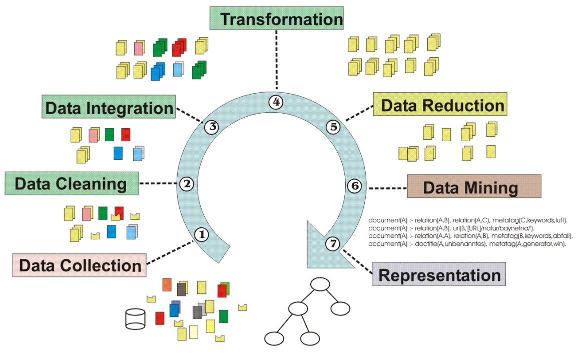
The ARTEMIS *Core System* contains the main system functionalities which are subdivided into the *kernel* and the *messaging system*. The *kernel* provides core functionalities for the workbench like realising **storage mechanisms**, running a **script interpreter** and providing the **ARTEMIS ontology** for the components.

ARTEMIS applies an expressive architecture in which communication within the workbench and its components is based on a script language interpreted by the *kernel script interpreter* and handled by the *messaging system*. Besides, the *scripting language* allows a *non-graphical usage* of the system and provides the ability to users to *record macros* of often occuring tasks in the workbench. ARTEMIS handles large amounts of data reflecting averaged sizes of current knowledge portals which can be stored on databases across the internet or on a local machine.

The accomplishment of a knowledge discovery process is handled by the **Workflow Model** which provides a workflow manager to monitor the flow of data and extracted information. Further, it assures the application of components depending on the current process step.

As indicated in Figure 3.2 ARTEMIS provides for each step of the process specialised components.

Generally, the approach can be used to enhance Web documents with semantic markup



```
document(A) :- relation(A,B), relation(A,C), metatag(C,keywords,luft).
document(A) :- relation(A,B), url(B,'[URL]/natur/baynetna/').
document(A) :- relation(A,A), relation(A,B), metatag(B,keywords,abfall).
document(A) :- doctitle(A,unbenanntes), metatag(A,generator,win).
```

Figure 3.2: ARTEMIS Workflow

in terms of an assignment to certain ontologies.

Chapter 4

Interface Layer Components

4.1 Annotation and Instance Editor (by L3S/USFD)

4.1.1 Expected Functionality

Nearly all applications need a user interface. For ontology-centric applications, such an interface needs to support the following areas:

- ontology instance creation and modification
- navigating the ontology

Ontology Instance Creation and Modification

A typical ontology consists of a large number of classes. Manually building a view/edit dialog for each of these classes would be a tedious process. The user interface component should be able to build such dialogs for a given ontology. These should then be further adaptable by the application developer. Such dialogs can also be used to provide query-by-example based query interfaces.

Ontology Navigation

To find the correct classes and/or instances, a user needs to get an overview of the ontology in use. This can be a simple tree view, but also much more elaborate interfaces, e.g. cluster maps [Aduna B.V., 2005].

4.1.2 Requirements

Internationalization

Internationalization is an essential requirement for many applications. Often ontologies already come with labels and terms in several languages. The user interface should be able to display these according to the user's language preferences.

Accessibility

The generated user interfaces should comply with accessibility regulations, e.g. US Section 508 in the United States, or results of the EuroAccessibility initiative [1].

Access Control

The user interface should adapt to the access rights of the current user and must only allow display and modification of the part of the ontology the user has the corresponding rights to see and/or change.

Usability

The generated dialogs should conform to the Look&Feel of the target platform. All generated user interfaces should have the same consistent handling.

Scalability

The user interface should not slow down significantly when handling a huge number of instances. It must provide means to navigate large ontologies without getting lost.

4.1.3 Existing Solutions

Magpie (by USFD)

Magpie [Domingue *et al.*, 2004] is a suite of tools which supports the interpretation of webpages and "collaborative sense-making", by annotating a text with instances from a known ontology. These instances can be used as a confidence measure for carrying out some services. The principle behind it is that it uses an ontology to provide a very specific and personalised viewpoint of the webpages the user wishes to browse. This is important

[1]http://www.euroaccessibility.org

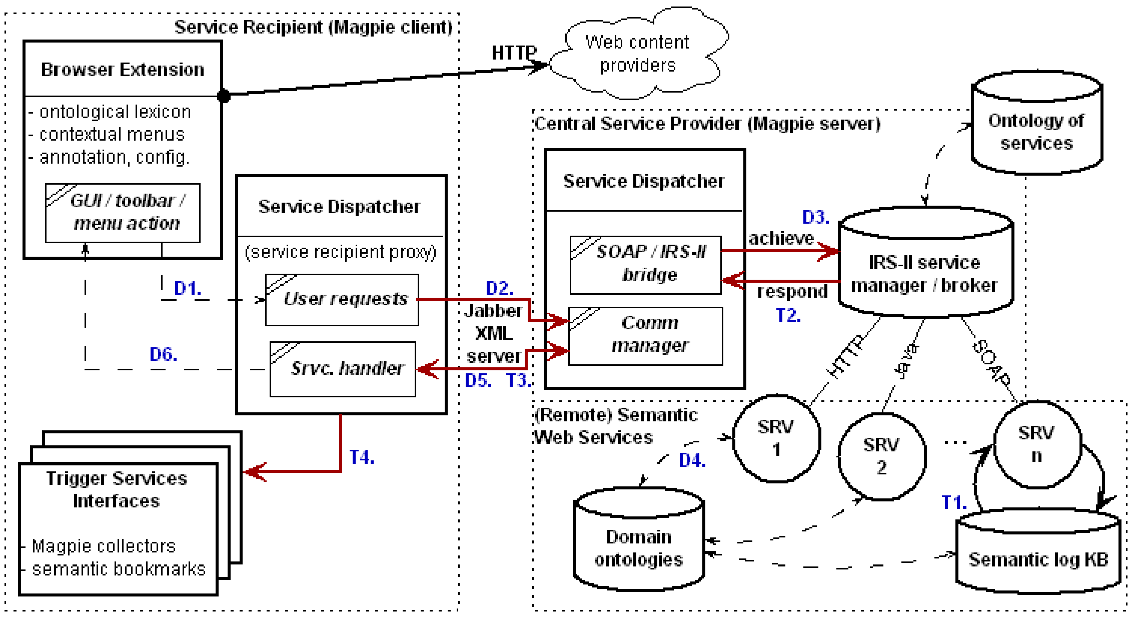


Figure 4.1: Architecture of MAGPIE

because different users often have different degrees of knowledge and/or familiarity with the information presented, and have different browsing needs and objectives.

The MAGPIE architecture is essentially that of a mediator, consisting of a Service Provider and a Service Recipient. Here we are interested mainly in the Service Provider component, which is built around a suite of tools providing access to a library of domain ontologies, populated knowledge bases, hand-crafted semantic services, and a semantic log knowledge base. Ontologies can be represented in RDF, DAML+OIl and OCML, and there is an OWL import/export mechanism. The Service Recipient (client side) components consist of a browser extension to Internet Explorer, a Service Dispatcher and Trigger Service Interfaces. The browser extension is embedded in the browser as a plugin.

OntoMat (by USFD)

OntoMat Annotizer [Handschuh *et al.*, 2002] is a user-friendly interactive annotation tool for web pages. It supports the user in the task of creating and maintaining ontology-based OWL markups, i.e. creating instances, attributes and relationships. It includes an ontology browser for the exploration of the ontology and instances, and an HTML browser that displays the annotated text. It is Java-based and provides a plugin Interface for extensions. The intended user is the individual annotator, i.e. somebody who wants to enrich their web pages with OWL metadata. Instead of manually annotating the page with a text editor, OntoMat allows the annotator to highlight relevant parts of the web page and create new instances via drag'n'drop interactions. It supports the metadata creation phase of the lifecycle, and is used in the OntoAgent project.

The OntoMat architecture comprises a plugin structure, which is flexible in terms of adding or replacing modules. The core OntoMat, which is downloadable, comprises an Ontology Guidance and Fact browser, a document viewer/editor, and an internal memory data structure for the ontology and metadata. The full semantic capabilities require a plugin connection to a corresponding annotation inference server.

SHAME - Standardized Hyper Adaptable Metadata Editor

The SHAME editor framework can be used to create RDF/XML metadata or to present the existing metadata of a given RDF model [Palmér *et al.*, 2004]. The SHAME Application Suite contains a standalone metadata editor, an application to create/edit formlets and a few examples. The standalone editor can be used to edit/create metadata using the following RDF metadata standards: LOM, Simplified Dublin Core, Dublin Core, PADLR LOM and some more. The examples show how to create an individual editor GUI using formlets, because of these formlets the SHAME RDF editor GUI is very flexible and can handle a lot of RDF metadata standards.

SHAME provides interfaces, so that the RDF editor GUI can be easily integrated into various applications, it also provides a Java Bean to integrate the RDF editor GUI into Java Server Pages. So e. g. the editor GUI can be integrated into an existing GUI application and all functions of the SHAME standalone editor can be used as well, so it is possible to program a very flexible RDF editor application, because the editor GUI can be changed easily from one metadata standard to another. Furthermore, because of the SHAME JB, the RDF editor GUI can be integrated into a system for storing RDF metadata into a database via a web interface. SHAME supports a lot of RDF metadata standards (e. g. LOM, DC) and also self defined attributes can be used. After editing the attributes SHAME parses the attributes, generates a RDF model and creates a RDF/XML stream. SHAME uses the Jena2 API to generate the RDF model of the given attributes. The Jena2 package provides methods for manipulating and creating RDF models, it also includes methods for querying RDF models and storing RDF metadata into a database or filesystem.

To create the RDF editor GUI, SHAME needs a *compound formlet*. This formlet refers to *atomic formlets* and, if any attribute uses a choice list, also to a RDF/XML ontology file containing the vocabularies for the choice list. An *atomic formlet* is a combination of two files for each attribute, a form file and a query file. In the query file the path to the attribute in the RDF model is specified. To specify the query path, the RDF Query Exchange Language (QEL) is used. The form file describes how the attribute is displayed in the editor GUI. This file includes the name of the attribute, a description, possibly both in different languages, the kind of the attribute (e. g. choice item), the cardinality of the attribute and possibly an ontology file which contains the values of the choice item. The ontology file is also in XML syntax, it contains the values for the choice to be shown in the editor GUI items and the corresponding values to be written to the RDF model.

Due to these formlets it is quite easy to create new attributes or to change a *compound formlet*. To create a new attribute only a form file which includes the specification of the attribute and the query file including the QEL query path to the attribute in the RDF model have to be created. To remove/add an attribute from an existing *compound formlet* only the reference in the *compound formlet* file has to be removed/added.

4.2 Semantic Web Service Infrastructure (by FT)

4.2.1 Introduction

Web Services (WSs) are interfaces that describe a collection of operations that are network-accessible through standardized Web protocols, and whose features are described using a standard XML-based language [Kreger, 2001, Curbera *et al.*, 2001]. Although there are other definitions of what a WS is, (refer to [Alonso *et al.*, 2003] for an enumeration of them), we believe that this definition captures better what a WS is (and from where its benefits come from). In few words, "'It's not the components, it's the interfaces"' [Kayne, 2003].

The main web services [Kreger, 2001, Curbera *et al.*, 2001] features are: (1) communication features that describe the protocols required to invoke the service execution; (2) descriptive features that detail the e-commerce properties; (3) functional features that specify the capabilities, enabling thus an external invoking agent to determine whether the service execution can obtain the requested results; and (4) structural features that describe the internal structure of a composite service, that is, which are its structural components and how these components are combined among them to execute the service. In this context, the Semantic Web has risen as an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation [Berners-Lee *et al.*, 2001]. Following this approach, Web Services in the Semantic Web, the so-called Semantic Web Services (SWS), will be the mark-up of WS to make them computer-interpretable, use-apparent and agent-ready [McIlraith *et al.*, 2001]. This mark-up is the semantic description of the WS and it will facilitate external agents to understand both the functionality and the internal structure of the services. The motivating tasks of SWS are to be able to discover, compose, and invoke automatically SWS [Hendler, 2001]. Several approaches have appeared with this aim, being the more relevant the OWL-S specification, IRS-II, WSMO and the SWSF.

The OWL-S [OWL Services Coalition, 2004] specification (formerly DAML-S [Ankolekar *et al.*, 2002]) has been proposed to describe services in a semantic manner, using OWL [Dean and Schreiber, 2004] in combination with the WSDL language [Christensen *et al.*, 2001] and SOAP [Gudgin *et al.*, 2003], achieving thus the desirable combination of WS standard languages and semantic annotation in order to use the current infra-structure of the WS [Sollazzo *et al.*, 2002].

The Internet Reasoning Service IRS-II [Motta *et al.*, 2003] is a SWS framework, which allows applications to describe semantically and execute SWS. It uses problem-solving methods (PSM) to represent SWS, attaching to each WS a PSM that describes it. A PSM is an abstract implementation of a domain-independent description of reasoning processes which can be applied to solve tasks in a specific domain. It is based on the UPML (Unified Problem Solving Method description Language) framework [Fensel *et al.*, 2003].

The Web Service Modelling Framework (WSMF) [Fensel and Bussler, 2002] tries to provide a model for describing the various components in an e-commerce environment. WSMF is the result of research carried out on modelling of reusable knowledge components and its core are two complementary principles: a strong de-coupling of the aforementioned components that carry out an e-commerce application; and a strong mediation between these elements. Mediation is applied at several levels: mediation of data structures; mediation of business logics; mediation of message exchange protocols; and mediation of dynamic service invocation. WSMF consists of four main elements: ontologies, goals, web services and mediators. All these elements are described using the Web Service Modelling Ontology (WSMO) [Lausen *et al.*, 2005]. The underlying representation language for WSMO is F-logic [Kifer *et al.*, 1995], a full first order logic language that provides second order syntax.

Web Service Description Language [Christensen *et al.*, 2001] standard does not contain the semantic expressivity needed to represent the requirements and capabilities of Web Services - a requirement for addressing the vexing heterogeneity challenges that need to be addressed for achieving (semi) automated discovery, improved reuse and faster composition. While efforts by the Semantic Web community to address this issue have resulted in proposals such as WSMF [Fensel and Bussler, 2002], OWL-S [OWL Services Coalition, 2004] and WSMO [Lausen *et al.*, 2005], they are seen as revolutionary and do not build on current WSDL standards, thereby making industry adoption less likely, since industry prefers evolutionary approaches whenever available. WSDL-S approach is right on that direction to demonstrate that any approach to annotating Web Services should and could be built by enriching the current Web Services standards with semantic descriptions.

SWSF (Semantic Web Services Framework) is a specification produced by the Semantic Web Services Language (SWSL) Committee of the Semantic Web Services Initiative (SWSI) (http://www.swsi.org/). In the same line of existing framework such as OWL-S, ODESWS, WSMO, the aim of SWSF is to provide richer semantic specifications of Web services in order to enable greater automation of service design, development end use. As in the other frameworks, the main objective of SWSF is to provide the needed technologies to realize the semantic web service vision [McIlraith *et al.*, 2001].

4.2.2 Service Directory

UDDI (Universal Description, Discovery and Integration) is a protocol and registry specification that enables organizations to use a standardized mechanism for organizing, discovering, reusing and managing Web services within an organization and across its partners.

UDDI specifies protocols for:

- Publishing and searching services registry
- Controlling access to registry
- Distributing and delegatating requests to other registries

Expected Functionality

UDDI provides a mechanism for managing a largely ad-hoc approach to building extensible Web service interactions. UDDI has a role defined both for development teams as well as within a Service-Oriented Infrastructure.

Typically, UDDI is used for:

- Publishing web services and finding them based on specific criteria
- Determining security and transport protocols a Web service supports
- Failover across Web services

in both a private registry scenario (enterprise) as well as interactions between trading partners (B2B).

In terms of semantics, UDDI's discovery is, as of now, based on:

– Simple keyword searching, which allows to specify the name (or part of the name) for a business, a service, a binding information or a "'tModel"' and return the entry from the repository.

  The use of Taxonomies and Identifiers to categorize UDDI elements at publishing time (businesses, services, "'tModel"' and also bindings with the UDDI V3.0) as well as specifying categories and/or identifiers to refine the search. Along with this, UDDI users can create and publish their own appropriate taxonomies, validate them and share them with other users.

UDDI V3 adds a policy guide to allow for additional flexibility within UDDI deployments given their context. However, the UDDI specification only provides a high level

Policy Schema and a set of Policy Abstractions both at the registry level and at the node level.

Like search engines, UDDI could provide automated searching capability that automatically and transparently search into and across taxonomies, based on keywords entered by the user, in order to provide a list of accurate services. This functionality would make UDDI searching capabilities much more accurate and could serve for other UDDI functionalities such as service versioning, life-cycle and quality of service, assuming that this kind of information can be provided using taxonomies.

Requirements

Ontology support Given the role for UDDI in Web services searching and discovery, the need to add more support for service semantics specification to the UDDI standard is obvious.

The following areas are currently being discussed within the UDDI Technical Committee and potential solutions as well as timelines are currently under discussion:

1. Standardize the description of taxonomies within UDDI by defining a common description language (using OWL). Indeed, the current use of taxonomy is limited by proprietary solutions, hence restricted to particular implementations which are not interoperable. A common way of describing external taxonomies would provide an option of using interoperable UDDI client applications that are not dependent on the implementation of the UDDI server they connect to.

2. Define a common API to manage the ontologies. The ideal scenario for taxonomy management would be achieved when users can create and publish their own taxonomies, share them, update or delete them, upload to and download from the server, check/validate value sets, manage security access, etc. Note that there is actually a lot of debate being done within the UDDI TC on whether UDDI should provide an API to manage ontologies (loading and deleting) or just recommend using existing 3rd party tools. A possible solution would be to have UDDI defining only the format in which users receive the ontology, the vendors would then have to implement themselves the management API.

3. Use Semantic Search protocols to enhance the UDDI discovery and search mechanism. The main problem is that service providers and service requesters have a very different knowledge on the services that are registered with UDDI. The use of Semantic expands the search to services that are similar to the services requested. A Technical Note was created by the UDDI TC to specify to exploit the semantics of OWL language and the relation between concepts as that are expressed in OWL ontologies to facilitate the service discovery in UDDI. Specifically, three types of searches are defined:

   - Exact search that retrieves entities whose classification matches the inquiry exactly;

     Specialization search to locate services whose specification is more specific than the inquiry (search within ontologies children);
   - Generalization search to locate services whose specification is less specific than the inquiry (search within ontology node parents - reverse of the specialization search). The proposal also extends the search by defining equivalency and similarity among the ontologies entries.

4. Formal policy language. While policies are defined in UDDI V3, the definition is abstract and an exact policy language is not specified. This is largely because no standard exists today around Web services policy definition. However, following a workshop held at the W3C in October 2004, the need for standardizing around Web services Constraints and Capabilities is well recognized.

Interoperability/Standard-Conformance The Service Directory should comply to the UDDI Specification. Unfortunately the current UDDI Specification (UDDI V1.0 to V3.0) does not provide any Semantic solution other than keyword searching and taxonomy classification. Hence, there is no vendor product available that offers any such standardized Semantic UDDI implementations. However, the next UDDI version (V4.0) will provide Semantics as follows:

- Using OWL as the language to describe taxonomies in UDDI. As explained above, standardizing a process to describe taxonomies is very valuable since all businesses would respect a same method to use taxonomies. Moreover, it would resolve interoperability issues when using different UDDI platforms from various vendors.
- Using OWL to make the searching mechanism more accurate and efficient.

As the UDDI and Semantic Web communities work together over time, it is possible that the standard will provide a dedicated ontology management API within the registry.

Existing Solutions

Commercial solutions Among registry or Semantic vendors, we can quote:

- Systinet WASP UDDI includes the complete set of pre-built taxonomies as defined by the UDDI specifications
- Network Inference provides tools to produce ontologies as well as services to help companies implementing ontologies
- Infravio: has their X-Registry

- HP provides a parser for RDF extended for OWL

University solutions

- Carnegie-Mellon University (CMU) created the Semantic Matchmaker, an entity that will allow web services to locate other services, provides a solution to the problem of matching, and allow for full implementation of interoperable service providers on the Web.
- University or Georgia (UGA) provides an environment for Web service discovery among multiple registries. This work uses an ontology-based approach to organize registries and enabling semantic classification and discovery of all Web services based on domains, using two algorithms created by UGA.
- Mind Swap provided an OWL-DL reasoner as well as an OWL-S Java API.

Open-source solutions

- Juddi is an open-source UDDI V2.0 server compliant solution
- Uddi4j is a UDDI V2.0 client solution

4.2.3 Service Discovery

The Semantic Web should enable greater access not only to content but also to services on the Web. Users and software entities should be able to discover, invoke, compose, and monitor Web resources offering particular services and having particular properties, and should be able to do so with a high degree of automation if desired. Powerful tools should be enabled by service descriptions, across the Web service lifecycle.

Expected Functionality

Service discovery is usually based on the rationale that services are selected, at run-time, based on their properties and capabilities. Providing automated support for service discovery constitutes, nowadays, one of the most challenging problems that web service technology faces. Indeed, because of the large number of available services and the heterogeneity and the versatility of the environment it is important to provide appropriate support for selecting services that are relevant to a specific user needs.

The Semantic Service Discovery Component aims at enhancing the potential of web services by providing support that enables effective automation of services discovery. The key idea is to ground the service discovery process on a semantic matchmaking between requester queries and available web service descriptions. To achieve this goal, the focus will be on fundamental issues related to: *(i)* service description, which consists in the definition of a formal semantics that enables understanding the structure and semantics of service description ontologies, and *(ii)* service discovery process, which consists in the design and implementation of effective reasoning mechanisms to support web service discovery.

Requirements

Semantic service description and user queries specification The service discovery component should provide rich and machine-processable abstractions that enable to describe service properties and capabilities as well as the specifications of user needs. This is essential to enable the development of reasoning mechanisms to handle the discovery process.

Approximate matchmaking A service discovery mechanism must support approximate matchmaking since it is unrealistic to expect service requests and service advertisments to exactly match. Such a approximate matchmaking process should be based on a measure of *semantic distance* between user queries and service descriptions. Given a user query, the matchmaking algorithm should be able to return the collection of services that best match the given query.

Consistency checking Semantic Service Discovery Component should be able to ensure that only services that are consistent with the user query are selected. Hence, the notion of consistency should be formally defined and the matchmaking algorithm should be designed in a way to prevent inconsistent answers.

Scalability This is one of the most important functionalities. The Service Discovery Component should provide support to enable efficient discovery of services in large, heterogeneous, and highly dynamic environments.

Service selection with respect to non-functional criteria The discovery process should also be able to select service with respect to non-functional criteria such as QoS measures, user preferences, etc.

Existing solutions

Current web services infrastructure have serious limitations with respect to meeting the automation challenges. For example, UDDI provides limited search facilities allowing only a keyword based search of businesses, services and the so-called tModels based on names and identifiers. To cope with this limitation, emerging approaches rely on semantic web technology to support service discovery [González-Castillo *et al.*, 2001, Paolucci *et al.*, 2002]. For example, [Bernstein and Klein, 2002] proposes to use process ontologies to describe the behaviour of services and then to query such ontologies using a Process Query Language (PQL). [Chakraborty *et al.*, 2001] defines an ontology based on DAML [DAML, 2005] to describe mobile devices and proposes a matching mechanism that locates devices based on their features (e.g., a type of a printer). The matching mechanism exploits rules that use the ontology, service profile information and the query to perform matching based on relationships between attributes and their values.

A Prolog based reasoning engine is used to support such a matching. There are other approaches based on a DAML-OIL [Horrocks, 2002] description of services that propose to exploit the description logic based reasoning mechanisms. [González-Castillo *et al.*, 2001] reports on an experience in building matchmaking prototype based on description logic reasoner which considers DAML+OIL based service descriptions. The proposed matchmaking algorithm is based on simple subsumption and consistency tests. [Paolucci *et al.*, 2002] proposes a more sophisticated matchmaking algorithm between services and requests described in DAML-S[2]. The algorithm considers various degrees of matching that are determined by the minimal distance between concepts in the concept taxonomy. Based on a similar approach, the ATLAS matchmaker [Payne *et al.*, 2001] considers DAML-S ontologies and utilizes two separate sets of filters: *1)* Matching functional attributes to determine the applicability of advertisements (i.e., do they deliver sufficient quality of service, etc). The matching is achieved by performing conjunctive pair-wise comparison for the functional attributes; *2)* Matching service functionality to determine if the advertised service matches the requested service. A DAML-based subsumption inference engine is used to compare input and output sets of requests and advertisements.

In [Benatallah *et al.*, 2005, Benatallah *et al.*, 2003], service discovery is formalized as a new instance of the problem of rewriting concepts using terminologies in the context of Description Logics, called the *best covering problem*. The proposed matchmaking algorithm provides the following building blocks for flexible and effective service discovery: *(i)* a global reasoning mechanism that allows to discover *combinations* of services that match (cover) a given request, *(ii)* a flexible matchmaking process that goes beyond subsumption tests, and *(ii)* effective computation of the missed information (i.e., the difference between the query and its rewriting) which can be used, for example, to improve service repository interactions.

Finally, it should be noted that the problem of capabilities based matching has also been addressed by several other research communities, e.g., information retrieval, software reuse systems and multi-agent communities. More details about these approaches and their applicability in the context of the semantic web services area can be found in [Bernstein and Klein, 2002, Paolucci *et al.*, 2002].

[2]http://www.daml.org/services/

4.2.4 Service Composition

Expected Functionality

Web services composition refers to the ability to synthetize a new service using existing services. Two kinds of service composition approaches have been proposed in the litterature: static composition, where services are composed at design time, and dynamic composition, where existing services are selected and composed at run-time. Dynamic service composition is emerging today as a promising technology for the effective automation of application-to-application integration.

The service composition component aims to advance the fundamental understanding to dynamically compose web services to perform complex tasks. This will lead to the development of a service composition infrastructure that enables users to locate and aggregate web services in a flexible and personalized manner. In particular, the proposed infrastructure will support the functionnalities described below.

Requirements

Supervised/automatic service composition The aim here is to provide assistance to a user in creating composite services. Two types of service composition should be supported: supervised and automatic composition. Supervised composition is based on a semi-automatic process that allows a user to progressively create a new service by interacting with the system. However, the aim of automatic composition is to fully automate the composition process. Given a specification of user needs, this implies the ability to locate services based on their capabilities and to aggregate them in order to achieve the required functionality.

Personalized service composition The service composition component should be able to take into account user preferences and/or profiles in the composition process in order to propose the most appropriate composite services to end users.

Correctness of service composition It is essential to ensure that synthetized composite services are correct in the sense that they can be effectively executed without generating errors. The notion of correctness of a composition must be formally defined and mechanisms to enforce it must be developped and integrated in the proposed composition infrastructure.

Scalability The service composition component should provide support to enable efficient composition of services in large, heterogeneous, and highly dynamic environments.

Service composition with respect to non-functional criteria The composition component should be flexible enough to enable to parameterize the service composition process using non-functional criteria such as QoS measures, etc.

Existing solutions There are some recent research work that are interested in providing formal foundations for service composition [Hull *et al.*, 2003, Bultan *et al.*, 2003, Berardi *et al.*, 2003]. [Hull *et al.*, 2003, Bultan *et al.*, 2003] propose a formal framework that enable to better understand the relations between the global properties of a composite service and the local properties of its components. The motivation behind this work is the development of techniques that enable to verify/synthetise global properties of composite services from the local properties of the components. [Berardi *et al.*, 2003] addresses he problem of automatic composition of services in a framework based on Deterministic Propositional Dynamic Logic (DPDL). The authors show that the problem of service composition existence can be reduced into the problem of satisfiability of a DPDL formula. [Narayanan and McIlraith, 2002, Hendler *et al.*, 2003] propose approaches that combine services annotation languages and planning techniques in order to enable automatic or semi-automatic composition of services. These kinds of approach constitute an alternative to static composition approaches based on BPEL4WS like languages. They enable the composition of new services from declarative specifications of their expected behaviors.

4.2.5 SemanticWeb Service Frameworks

WSMO Framework

The Web Service Modeling Ontology (WSMO) is an ontology for semantically describing Semantic Web Services. Taking the Web Service Modeling Framework (WSMF) [Fensel and Bussler, 2002] as a starting point, WSMO refines and extends this framework, and develops a formal ontology and language. WSMF consists of four different main elements for describing semantic Web Services:

- ontologies which provide the concepts and relationships used by other elements,
- goals that define the users' objectives, i.e. the (potential) problems that should be solved by Web Services,
- Web Services descriptions that define various aspects of a Web Service, and
- mediators which bypass interoperability problems.

The Web Service Modeling Ontology (WSMO) is developed in the context of WSMO Working Group, as part of the SDK cluster, with the aim of, through alignment between key European research projects in the Semantic Web Service area, the further the development of Semantic Web Services and works toward further standardization in the area of Semantic Web Service languages and to work toward a common architecture and platform for Semantic Web Services. The WSMO Working Group includes the WSML Working Group, which aims at developing a language called Web Service Modeling Language (WSML) that formalizes the Web Service Modeling Ontology (WSMO), and the WSMX Working Group, which aims at providing an execution environment and a reference implementation for WSMO.

WSMO Design Principles

WSMO provides ontological specifications for the core elements of Semantic Web Services. In fact, Semantic Web Services aim at an integrated technology for the next generation of the Web by combining Semantic Web technologies and Web Services, thereby turning the Internet from a information repository for human consumption into a worldwide system for distributed web computing. Therefore, appropriate frameworks for Semantic Web Services need to integrate the basic Web design principles, those defined for the Semantic Web, as well as design principles for distributed, service-orientated computing of the Web. WSMO is therefore based on the following design principles:

- Web Compliance - WSMO inherits the concept of URI (Universal Resource Identifier) [Berners-Lee *et al.*, 2005] for unique identification of resources as the essential design principle of the Word Wide Web. Moreover, WSMO adopts the concept of Namespaces for denoting consistent information spaces, supports XML and other W3C Web technology recommendations, as well as the decentralization of resources.
- Ontology-Based - Ontologies are used as the data model throughout WSMO, meaning that all resource descriptions as well as all data interchanged during service usage are based on ontologies. Ontologies are a widely accepted state-of-the-art knowledge representation, and have thus been identified as the central enabling technology for the Semantic Web. The extensive usage of ontologies allows semantically enhanced information processing as well as support for interoperability; WSMO also supports the ontology languages defined for the Semantic Web.
- Strict Decoupling - Decoupling denotes that WSMO resources are defined in isolation, meaning that each resource is specified independently without regard to possible usage or interactions with other resources. This complies with the open and distributed nature of the Web.
- Centrality of Mediation - As a complementary design principle to strict decoupling, mediation addresses the handling of heterogeneities that naturally arise in open environments. Heterogeneity can occur in terms of data, underlying ontology, protocol or process. WSMO recognizes the importance of mediation for the successful

deployment of Web Services by making mediation a first class component of the framework.

- Ontological Role Separation - Users, or more generally clients, exist in specific contexts which will not be the same as for available Web Services. For example, a user may wish to book a holiday according to preferences for weather, culture and childcare, whereas Web Services will typically cover airline travel and hotel availability. The underlying epistemology of WSMO differentiates between the desires of users or clients and available services.

- Description versus Implementation - WSMO differentiates between the descriptions of Semantic Web Services elements (description) and executable technologies (implementation). While the former requires a concise and sound description framework based on appropriate formalisms in order to provide a concise for semantic descriptions, the latter is concerned with the support of existing and emerging execution technologies for the Semantic Web and Web Services. WSMO aims at providing an appropriate ontological description model, and to be complaint with existing and emerging technologies.

- Execution Semantics - In order to verify the WSMO specification, the formal execution semantics of reference implementations like WSMX as well as other WSMO-enabled systems provide the technical realization of WSMO.

WSMO Top-level Elements

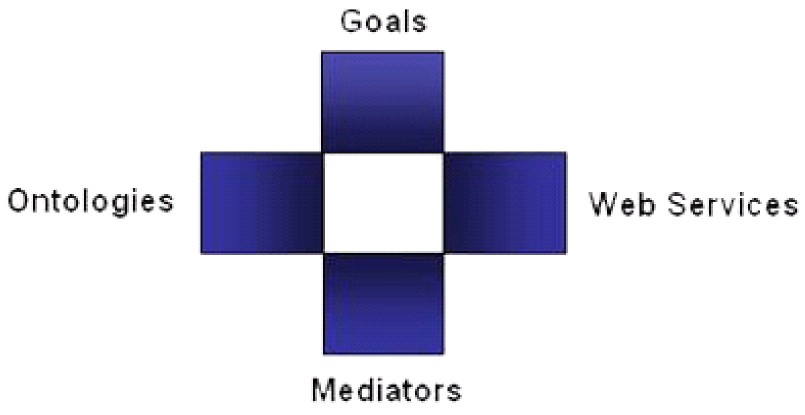


Figure 4.2: WSMO Top-level Elements

The following briefly outlines the conceptual model of WSMO - the complete specification can be found in [Lausen *et al.*, 2005]. The elements of the WSMO ontology are defined in a meta-meta- model language based on the Meta Object Facility (MOF) [MOF, 2002]. In order to allow complete item descriptions, every WSMO element is described by non-functional properties. These are based on the Dublin Core (DC) Metadata Set [DCMI Usage Board, 2005] for generic information item descriptions, and other service-specific properties related to the quality of service.

Ontologies Ontologies provide the formal semantics for the terminology used within all other WSMO components. Using MOF, we define an ontology as described in the Listing 1 below:

```
hasNonFunctionalProperties type nonFunctionalProperties
     importsOntology type ontology
     usesMediator type ooMediator
     hasConcept type concept
     hasRelation type relation
     hasFunction type function
     hasInstance type instance
     hasAxiom type axiom
```

Listing 1: Ontology Definition Class ontology

A set of non-functional properties are available for characterizing ontologies; they usually include the DC Metadata elements. Imported ontologies allow a modular approach for ontology design and can be used as long as no conflicts need to be resolved between the ontologies. When importing ontologies in realistic scenarios, some steps for aligning, merging and transforming imported ontologies in order to resolve ontology mismatches are needed. For this reason ontology mediators are used (ooMediators). Concepts constitute the basic elements of the agreed terminology for some problem domain. Relations are used in order to model interdependencies between several concepts (respectively instances of these concepts); functions are special relations, with a unary range and a n-ary domain (parameters inherited from relation), where the range value is functionally dependent on the domain values, and instances are either defined explicitly or by a link to an instance store, i.e., an external storage of instances and their values.

Web Services WSMO provides service descriptions for describing services that are requested by service requesters, provided by service providers, and agreed between service providers and requesters. In the Listing 2, the common elements of these descriptions are presented.

```
hasNonFunctionalProperties type nonFunctionalProperties
 importsOntology type ontology
 usesMediator type {ooMediator, wwMediator}
 hasCapability type capability multiplicity = single-valued
 hasInterface type interface
```

Listing 2: Service Description Definition Class service

Within the service class the non-functional properties and imported ontologies attributes play a role that is similar to that found in the ontology class with the minor addition of a quality of service non-functional property. An extra type of mediator to deal with protocol and process related mismatches between web services is also included.

The final two attributes define the two core WSMO notions for semantically describing Web Services: a capability which is a functional description of a Web Service, describing constraints on the input and output of a service through the notions of preconditions, assumptions, postconditions, and effects; and service interfaces which specify how the service behaves in order to achieve its functionality. A service interface consists of a choreography which describes the interface for the client-service interaction required for service consumption, and an orchestration which describes how the functionality of a Web Service is achieved by aggregating other Web Services.

Goals A goal specifies the objectives that a client may have when consulting a Web Service, describing aspects related to user desires with respect to the requested functionality and behavior. Ontologies are used as the semantically defined terminology for goal specification. Goals model the user view in the Web Service usage process and therefore are a separate top level entity in WSMO.

```
hasNonFunctionalProperties type nonFunctionalProperties
importsOntology type ontology
usesMediator type {ooMediator, ggMediator}
requestsCapability type capability multiplicity = single-valued
requestsInterface type interface
```

Listing 3: Goal Definition Class goal

As presented in Listing 3 above, the requested capability in the definition of a goal represents the functionality of the services the user would like to have, and the requested interface represents the interface of the service the user would like to have and interact with.

Mediators The concept of Mediation in WSMO addresses the handling of heterogeneities occurring between elements that shall interoperate by resolving mismatches between different used terminologies (data level), on communicative behavior between services (protocol level), and on the business process level. A WSMO Mediator connects elements and provides mediation facilities for resolving mismatches. The description elements of a WSMO Mediator are its source and target elements, and the mediation service for resolving mismatches, as shown in the Listing 4 below.

WSMO defines different types of mediators for connecting the distinct WSMO elements: OO Mediators connect and mediate heterogeneous ontologies, GG Mediators connect Goals, WG Mediators link Web Services to Goals, and WW Mediators connects interoperating Web Services resolving mismatches between them.

```
hasNonFunctionalProperties type nonFunctionalProperties
importsOntology type ontology
hasSource type {ontology, goal, service, mediator}
hasTarget type {ontology, goal, service, mediator}
hasMediationService type {goal, service, wwMediator}
```

Listing 4: Mediators Definition Class mediator

4.2.6 ODESWS Framework

With the aim of designing and composing SWS, the ODESWS framework (Figure 4.3) has been proposed for SWS [Gómez-Pérez, 2004a]. The following elements have been identified:

- **ODESWS Ontology.** To describe the features of a web service, we have used a stack of ontolgies since they present the features in a formal and explicit way. This stack of ontologies is composed of 4 layers: data-type ontology, knowledge representation ontology, PSM Description Ontology and SWS ontology.

- **Instance model.** Designing SWS means to instantiate each of the ontologies of the stack that describes what a service is: the domain ontology used by the service is built on top of the data type and knowledge representation ontologies; the service features are instances of both PSM and SWS ontologies. The whole instances constitute a model that specifies the SWS at the knowledge level. The essential idea here is that the user does not handle the ontologies directly but a graphical PSM alike structure.

- **Checking model.** Once the instance model has been created, it is necessary to guarantee that the ontology instances do not present inconsistencies among them. Design rules will be needed to check this, particularly when ontology instances have been created automatically (as in the case of (semi) automatic composition).

- **Translate model.** Although a service is modelled at the knowledge level, it must be specified in a SWS-oriented language to enable programs and external agents to access its capabilities. Therefore, once the instance that describes the WS is created and checked, it is automatically translated into any of the existent SWS representational language.

This framework will enable the (semi) automatic composition of SWS using (1) PSM refiners and bridges to adapt the PSM ontology instances to the required capabilities of the new service; and (2) design rules to reject both PSM and SWS ontology instances that present errors or inconsistencies among them. Design rules are used to reduce the service candidates that are to be combined to obtain a new service.

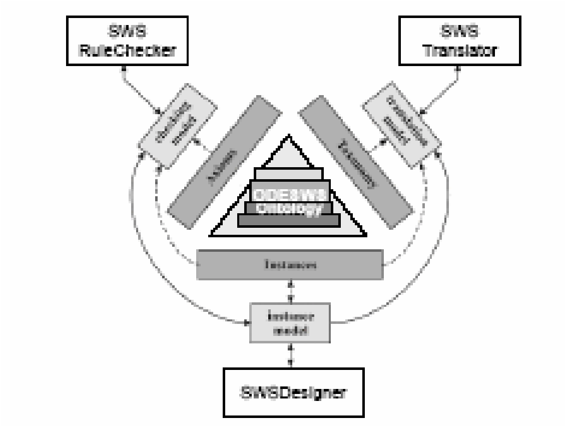


Figure 4.3: ODESWS Framework

ODESWS Ontology

The aim of designing SWS is to describe explicitly and semantically the features of a WS. To achieve this purpose, the use of ontologies seems to be the most appropriate solution, and, in fact, this approach has been followed by other authors. In the OWL-S specification [OWL Services Coalition, 2004] the ontology is directly constructed in a semantic-enriched mark-up language (OWL), and the internal structure of the services is described with workflows. The WSMO ontology [Lara *et al.*, 2004] uses the F-Logic language to specify the service and it introduces the concept of mediator to decouple the different elements of the service. Each of these solutions considers a particular semantic language to describe the ontology of SWS description.

The ODESWS framework, however, proposes an ontology (Figure 4.4), called ODESWS ontology [Gómez-Pérez, 2004a], that describes the SWS at the knowledge and independent-language level. Thus, the ODESWS ontology is composed of a stack of ontologies developed following well-known specifications or de facto standards that cover all the features of the SWS. These ontologies are:

- **Data Type (DT) Ontology.** It describes the types of concept attributes of the domain ontology used to define the input/output parameters. The DT ontology is based on the XML Schema Datatypes [Biron and Malhotra, 2001], a W3C recommendation formally included into the semantic Web languages, as OWL.

- **Knowledge Representation (KR) Ontology.** This ontology describes the knowledge primitives (concept, instance attribute, etc.) used to represent the domain ontology, which contains descriptions of the knowledge and data managed by the SWS. This ontology has been constructed on the basis of the WebODE knowledge model [Arpírez *et al.*, 2003], which is frame-based and incorporates formulas to represent axioms.

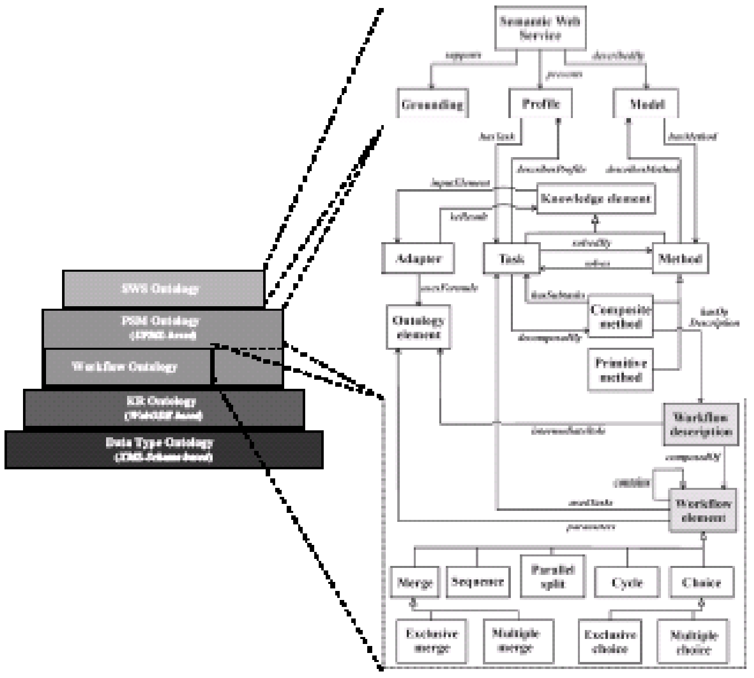


Figure 4.4: ODESWS Ontology

- **Problem-Solving Method Ontology.** It is based on the Problem-Solving Method (PSM) paradigm, where a PSM is a domain-independent and knowledge-level specification of a problem-solving behaviour which can be used to solve a class of problems [Motta, 1999]. The PSM ontology is based on the Unified Problem-solving Method Language (UPML) [Fensel *et al.*, 2003] that is a de facto standard for describing the PSM components: (1) tasks describe the operation to be solved in the execution of its solving method by specifying the input/output parameters and the pre/post-conditions required to be applicable (this description is independent of the method used for solving the task); (2) method details the control of the reasoning process to achieve a task. A method can be composite, which means that is composed of a number of subtasks whose coordinated execution indicates how the method will be carried out; and (3) adapters [Fensel, 1997] specify mappings among the knowledge components of a PSM. The adapters are used to achieve the reusability, since they bridge the gap between the general description of a PSM and the particular domain where it is applied. To specify the coordination of the subtasks of the composite methods, that is, the operational description of these methods, we have developed the workflow ontology. This ontology describes explicitly and semantically the workflow primitives [van der Aalst and van Hee, 2002, van der Aalst *et al.*, 2003] that will be used to determine the coordination of the execution of the subtasks.

- **Semantic Web Service Ontology.** The SWS ontology is on top of the stack and it describes all the features of the services. This ontology replicates the upper-level concepts of the OWL-S ontology, but these concepts do not have the same attributes and relationships of the OWL-S specification. Thus, these upper-level concepts are connected with the concepts of the PSM ontology in the following way: (1) profile is a concept whose attributes specify the SWS non-functional features, and it establishes a relationship (hasTask) (see Figure 2) with the task concept of the PSM ontology to describe the SWS functional features; (2) model is a concept that establishes a relationship (hasMethod) with the method concept of the PSM ontology; it describes the components of the internal structure of the service and the control flow that indicates how those components are coordinated to solve the task related to the functional features of the service; and (3) grounding, which specifies the access protocol and the necessary message exchanges to invoke the service.

4.2.7 OWL-S Framework

OWL-S (formerly DAML-S) is both a language and an ontology for describing semantic Web services. It builds on Semantic Web technology developed at W3C to describe the properties and capabilities of Web services. Indeed, OWL-S employs the Ontology Web Language (OWL) [Dean and Schreiber, 2004], a recommendation produced by the Web-Ontology Working Group at the W3C, to supply service providers with a core set of markup language constructs for describing Web services in a computer-interpretable form, thereby facilitating the automation of Web service discovery, invocation, composition and execution.

OWL-S has been developed in the context of the DAML program (http://www.daml.org/) and has been submitted to W3C member in November 2004. A number of OWL-S based tools has been proposed recently such as OWL-S Protégé-based Editor, which provides a set of capabilities for creating and maintaining OWL-S service descriptions, or OWLSM (OWL-S Matcher), which implements an algorithm that outputs different degrees of matching for individual elements of OWL-S descriptions. More details about existing tools can be found at http://www.daml.org/services/owl-s/tools.html.

OWL-S upper ontology

OWL-S can be viewed as a particular OWL ontology that enables to describe different facets of web services. As depicted at Figure 4.5, an OWL-S ontology of services is structured in three main parts :

- **ServiceProfile** describes the capabilities and parameters of the service. It is used for advertising and discovering services.
- **ServiceModel** gives a detailed description of a service's operation. Service operation is described in terms of a process model, which details both the control

structures and dataflow structures of the service required to execute a service.

- **ServiceGrounding** specifies the details of how to access the service, via messages (e.g., communication protocol, message formats, addressing, etc).

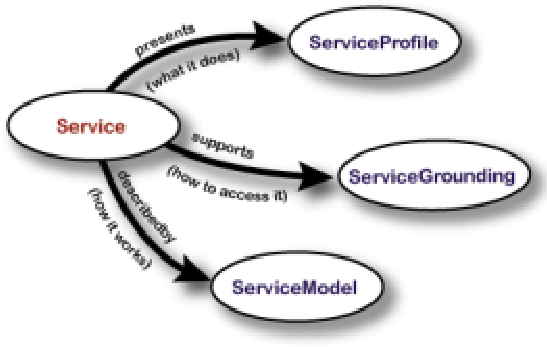


Figure 4.5: Top level of the OWL-S ontology

We describe below these components in more details.

OWL-S service profiles

The service profile provides information about a service that can be used by an agent to determine if the service meets its needs. A profile consists of three types of information:

- A (human readable) description of the service (e.g., serviceName, textDescription, contactInformation, etc).
- the functional behavior of the service which is represented as a transformation from the inputs required by the service to the outputs produced. A functional description of a service is expressed using the IOPE paradigm, i.e., the Inputs required by a service, the Outputs generated, the Preconditions required to be satisfied and the expected Effects that result from the execution of the service.
- Several other properties that are used to describe features of the service. Two main types of properties can be used here: (i) properties that enable to specify the category of a given service (e.g., the category of the service within the UNSPSC taxonomy), and (ii) non-functional attributes which specify QoS criteria (e.g., the cost of the service).

Figure 4.6 describes some classes and properties of a service profile.

It should be noted that in the OWL-S approach, a service profile is intended to be used by providers to advertise their services as well as by service requesters to specify their needs.

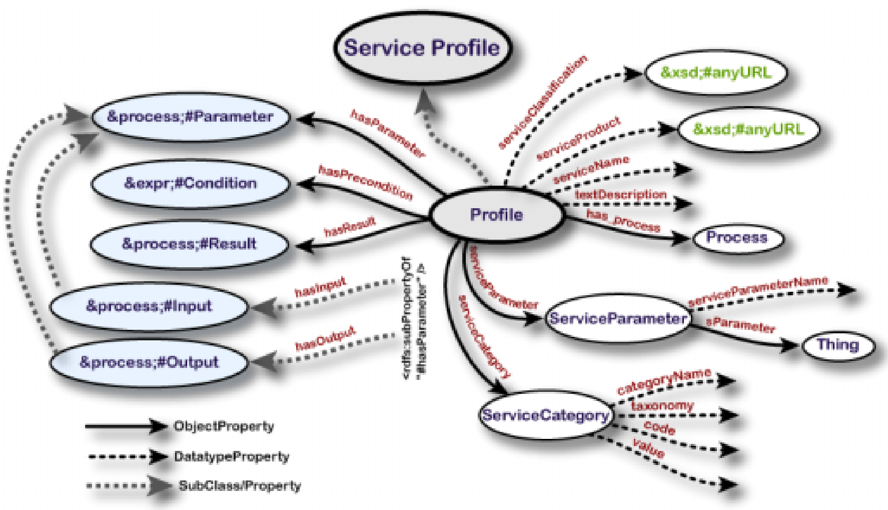


Figure 4.6: Classes and properties of a profile

OWL-S process model

In addition to service Profiles, OWL-S allows a process-based description of services using the class *Process*, a subclass of *ServiceModel*. A process specifies how a given client may interact with a service. OWL-S distinguishes between three kinds of processes:

- **AtomicProcess** corresponds to simple actions made of one-step interactions. Therefore, an atomic process do not support complex conversations between a service and its clients. It only expects one message as input and returns one message in response.
- **SimpleProcess** gives an abstraction mechanism to provide multiple views of the same process. They are made of a single execution step like atomic processes. However, simple processes are not invocable but can, instead, be used as elements of abstraction (e.g., to provide a simplified representation of a composite process).
- **CompositeProcess** correspond to actions that require multi-steps interactions (i.e., complex conversations). Therefore, a composite process needs to maintain information about execution states during conversations that involve multiple messages. OWL-S provides several flow control constructs that enable to specify composite processes (e.g., split, sequence, choice, if-then-else, etc).

It is worth noting that describing services as composite processes may be useful in several tasks such as, for example, (i) to perform a compatibility analysis in order to check whether two services (or a client and a service) can interact correctly, (ii) to create new services by composing exiting ones, (iii) to coordinate the activities of different participants during the course of service enactment, and (iv) to monitor service execution.

OWL-S service grounding

A service grounding specifies technical details of how a service can be accessed by a client. For example, a service grounding gives information about the address, the communication protocol and the message formats to be used by an agent to be able to communicate with a given service. A service grounding plays the role of a mapping from an abstract description of a service (e.g., ServiceProfile or ServiceModel) into a concrete one. Of particular interest is the mapping of OWL-S descriptions into WSDL which enables a developer to take benefits from: (i) rich description and reasoning mechanisms provided by OWL-S when designing a service, and (ii) the opportunity to reuse the extensive work achieved in the area of web services around WSDL and SOAP during the implementation of a service. Figure 4.7 given below show a mapping between OWL-S and WSDL.

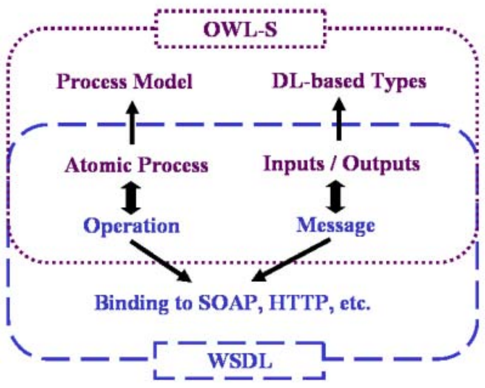


Figure 4.7: Relationships between OWL-S and WSDL

Briefly speaking, a mapping from OWL-S into WSDL is based on the following three correspondences:

- An atomic process is mapped into a WSDL operation,
- Inputs and outputs of an atomic process are mapped into WSDL messages.
- Types (OWL classes) of the inputs and the outputs are mapped into WSDL's extensible notion of abstract type.

4.2.8 WSDL-S Framework[3]

Requirements for Web Services Semantics

The WSDL-S effort recommends that certain principles guide any work to define a framework for Web services semantics. Those principles are the following.

Build on existing Web Services standards The Web services standards are fast becoming a preferred technology for application integration because of the promise of their interoperability. Companies are making investments in integration projects based on Web Services. Therefore, we believe that any approach to adding semantics to Web Services should be specified in an upwardly compatible manner so as to not disrupt the existing install-base of Web Services and associated investments in human training and technical solutions [Sivashanmugam *et al.*, 2003].

The mechanism for annotating Web services with semantics should be independent of the semantic representation language There are a number of potential languages for representing semantics such as OWL [Dean and Schreiber, 2004], WSMO [Lausen *et al.*, 2005], and UML [UML, 2003]. Each language offers different levels of semantic expressivity and developer support. Our position is that it is not

[3] This section contains material from "'Web Service Semantics - WSDL-S"'([Akkiraju *et al.*, 2005], Copyright 2005 International Business Machines Corporation and University of Georgia. All rights reserved. Copying is granted as long as the following copyright notice is included:
IBM and the University of Georgia (collectively, the "Authors") hereby grant you permission to copy and display the Web Service Semantics ? WSDL-S Technical Note, in any medium without fee or royalty, provided that you include the following on ALL copies of the Web Services Semantic Annotations ? WSDL-S Technical Note, or portions thereof, that you make:
1. A link or URL to the Specification at this location
2. The copyright notice as shown in the Web Service Semantics — WSDL-S Technical Note
EXCEPT FOR THE COPYRIGHT LICENSE GRANTED ABOVE, THE AUTHORS DO NOT GRANT, EITHER EXPRESSLY OR IMPLIEDLY, A LICENSE TO ANY OTHER INTELLECTUAL PROPERTY THEY OWN OR CONTROL.
WEB SERVICE SEMANTICS ? WSDL-S TECHNICAL NOTE IS PROVIDED "AS IS," AND THE AUTHORS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF WEB SERVICE SEMANTICS ? WSDL-S TECHNICAL NOTE ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
THE AUTHORS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE WEB SERVICE SEMANTICS ? WSDL-S TECHNICAL NOTE.
The name and trademarks of the Authors may NOT be used in any manner, including advertising or publicity pertaining to the Specification or its contents without specific, written prior permission. Title to copyright in Web Service Semantics ? WSDL-S Technical Note will at all times remain with the Authors.
No other rights are granted by implication, estoppel or otherwise.

necessary to tie the Web services standards to a particular semantic representation language. This is consistent with the approach prescribed by Sivashanmugam et al in their work [Sivashanmugam *et al.*, 2005]. By keeping the semantic annotation mechanism separate from the representation of the semantic descriptions, the approach offers flexibility to developer community to select their favorite semantic representation language. In the next section, we will show a way such independence can be achieved.

The mechanism for annotating Web services with semantics should allow the association of multiple annotations written in different semantic representation languages As mentioned earlier, there are many potential semantic representation languages. Service providers may choose to annotate their services in multiple semantic representation languages to be discovered by multiple discovery engines. Therefore, we believe that the mechanism for annotating Web Services with semantics should allow multiple annotations to be associated with Web Services.

Support semantic annotation of Web Services whose data types are described in XML schema A common practice in Web services-based integration is to reuse interfaces that are described in XML. The definition of business documents using XML schema is a wide-spread and successful practice. XML schemas will be an important data definition format for the foreseeable future. We believe that the semantic annotation of service inputs and outputs should support the annotation of XML schemas. WSDL 2.0 supports the use of other type systems in addition to XML Schema, so constructs in semantic models, such as classes in OWL [OWL] ontologies, could be used to define the Web service input and output data types. But an approach that does not address XML schema-based types will not be able exploit exiting assets or allow the gradual upgrade of deployed WSDL documents to include semantics.

Provide support for rich mapping mechanisms between Web Service schema types and ontologies Given our position on the importance of annotating XML schemas in Web service descriptions, attention should be given to the problem of how to map XML schema complex types to ontological concepts. Again, an agnostic approach to the selection of schema mapping languages is called for. For example, if the domain model is represented in OWL, the mapping between WSDL XSD elements and OWL concepts can be represented in any language of user's choice such as: XSLT, XQuery, RDF/S, OWL or any other arbitrary language as long as the chosen language is fully qualified with its own namespace.

Web Services Semantics: WSDL-S Approach

Following the principles described above IBM and University of Georgia have collaboratively developed Web Services Semantics - WSDL-S [Akkiraju *et al.*, 2005] - a technical note document. This technical note prescribes a mechanism to annotate Web Services with semantics. It is conceptually based on, but a significant refinement in details of, the original WSDL-S proposal [Miller *et al.*, 2004] from the LSDIS laboratory at the University of Georgia. In WSDL-S, we augment the expressivity of WSDL with semantics by employing concepts similar to those in OWL-S while being agnostic to the semantic representation language (we only refer to profile model in this document. OWL-S process model compares with BPEL4WS and it is not discussed here). The advantage of this evolutionary approach to adding semantics to WSDL is multi-fold. First, users can, in an upwardly compatible way, describe both the semantics and operation level details in WSDL- a language that the developer community is familiar with. Second, by externalizing the semantic domain models, we take a language-agnostic approach to ontology representation. This allows Web service developers to annotate their Web services with their choice of modeling language (such as OWL, or legacy models developed in UML or other knowledge representation languages discussed above). This is significant because the ability to reuse existing domain models expressed in modeling languages like UML can greatly alleviate the need to separately model semantics. Finally, it is relatively easy to update the existing tooling around WSDL specification to accommodate our incremental approach. While it is noted that the theoretical underpinnings of OWL-S in description logic or WSMO in F-Logic make them rich languages for representing semantics, we believe that extending the industry standards such as WSDL to include semantics is a more practical approach for adoption. Moreover, by externalizing the semantic domain models in our proposal, we still allow for richer representations of domain concepts and relationships in languages such as OWL, thereby bringing together the best of both worlds. Use of expressive mapping representation and techniques can further enable this approach to deal with significant types of syntactic, structural, representational and semantic heterogeneity.

Below, we illustrate the key elements of the WSDL-S approach to adding semantics to the WSDL specification.

Using the Extensibility Elements of WSDL

In this section we briefly describe how semantic annotations are added to WSDL document elements in the Web Services Semantics - WSDL-S technical note (for details, see the technical note [Akkiraju *et al.*, 2005]). In this approach, we focus on semantically annotating the abstract definition model of a service in WSDL specification to enable dynamic discovery services. Annotation of service implementation model is not discussed for brevity. In essence, we provide URI reference mechanisms via extensibility elements to the interface, operation and message constructs to point to the semantic annotations defined in the externalized domain models for services. A quick summary of the extensibility elements provided in this technical note are:

- an extension element, namely wssem:modelReference, to allow for one-to-one associations of WSDL input and output type schema elements to the concepts in a semantic model

- an extension attribute, namely wssem:schemaMapping, to allow for many-to-many associations of WSDL input and output type schema elements to the concepts in a semantic model - typically associated with XML schema complex types
- two new elements, namely wssem:precondition and wssem:effect, which are specified as child elements of the operation element and describe the semantics of the operation along the lines of the OWL-S approach. Preconditions and effects are pprimarily used in service discovery, and are not required to invoke a given service, and
- an extension attribute on interface element, namely wssem:serviceCategorization. It consists of service categorization information that could be used when publishing a service in a Web Services registry such as UDDI. It corresponds to the categorization concept proposed in OWL-S.

An example WSDL document that is semantically annotated using this approach is given below (again see. the technical note for details). In this sample, we present a simple purchase order service. The inputs and outputs of ProcessPurchaseOrder service are annotated with semantic references and preconditions and effects are introduced as extensibility elements to the ProcessPurchaseOder operation. The semantic concepts and their relationships are modeled in an OWL ontology - PurchaseOrder.owl (presented in Appendix C of [Akkiraju *et al.*, 2005]). Annotating XSD elements with OWL concepts represents a schema mapping problem with representation model heterogeneity, as OWL is far more expressive than XSD. A survey of schema matching and mapping approaches, as well as description of some approaches which have tried to automate the XSD to OWL approach is present in Appendix D of [Akkiraju *et al.*, 2005]. A revised version of the LSDIS semantic annotation tool that supports this proposal is expected in May 2005.

SWSF Framework

SWSF[4] (Semantic Web Services Framework) is a specification produced by the Semantic Web Services Language (SWSL) Committee of the Semantic Web Services Initiative (SWSI) (http://www.swsi.org/). In the same line of existing framework such as OWL-S, ODESWS, WSMO, the aim of SWSF is to provide richer semantic specifications of Web services in order to enable greater automation of service design, development end use. As in the other frameworks, the main objective of SWSF is to provide the needed technologies to realize the semantic web service vision [McIlraith *et al.*, 2001].

SWSF is made of two main components:

- **SWSL**, the Semantic Web Services Language, is logical language that is used to formally describe web services.

[4]http://www.daml.org/services/swsf/1.0/overview/

- **SWSO**, the Semantic Web Services Ontology, which presents a conceptual model, and its formal characterization, that enables to describe services.

These two components are described below.

SWSL (Semantic Web Service Language)

SWSL is a general-purpose logical language that includes special features (e.g., URIs, XML namespaces, etc) to make it usable with the Web technologies. More precisely, SWSL includes the following two sublanguages which are intended to be used in different tasks:

- **SWSL-FOL**, is a full first-order logic language, extended with features from HiLog [Chen et al. 1993] and F-logic [Kifer et al., 1995]. SWSL-FOL is used to specify the service ontology (i.e., SWSO). Its semantics is based on the standard first-order model theory and is monotonic. SWSL-FOL is well suited for specifying process ontologies and for providing associated reasoning mechanisms (e.g., planning techniques using process models which has been used to automate service composition).
- **SWSL-Rules**, a rule-based language with non-monotonic semantics. It includes a combination of features from Courteous logic programs [Grosof 1999], HiLog, and F-logic. SWSL-Rules is intended to be used as a specification and an implementation language for web services. It is well suited for tasks such as service discovery, profile specification, policy specification, etc.

Both sublanguages of SWSL are presented as layered languages where each new layer includes new concepts that enhance the modeling power of the language. SWSL specification describes how the two sublanguages can be combined together (i.e., how specifications in one sublanguage can use specifications written in the other sublanguage). More details about such a bridge between the two sublanguages can be found at http://www.daml.org/services/swsf/1.0/swsl/.

SWSO (Semantic Web Services Ontology)

SWSO presents a conceptual model that enable description of the web service ontology. It includes a complete first-order logic axiomatization, called FLOWS (First-Order Logic Ontology for Web Services), which defines the model-theoretic semantics of the ontology. FLOWS is given using the sublanguage SWSL-FOL. A translation of FLOWS into the sublanguage SWSL-Rules has been proposed and the resulting ontology is called ROWS (Rules Ontology for Web Services).

FLOWS captures the main concepts of various existing models of Web services as, for example, OWL-S (e.g., atomic processes, inputs, outputs, preconditions and effects) and WSMO (e.g., ontology for describing the intended goals of Web services). Following the upper ontology of OWL-S, FLOWS displays three major components:

- **Service Descriptors**, provides basic functional and non-functional information about a service (e.g., service name, service contact information,service reliability, service cost, etc). Service descriptiors are primarily intended to be used for service discovery.
- **Process Model**, based on a subset of PSL (Process Specification Language) [Gruninger 2003], an international standard (ISO 18629), that was originally developed to enable specification of manufacturing processes. FLOWS extends PSL with number of concepts in order to cope with web service specifics. FLOWS process model is layered and hence one can incorporate only the ontological constructs that are needed for a given application. More precisely, FLOWS ontology consists of a core set of axioms, called FLOWS-Core, and a set of extension ontologies. FLOWS-Core provides the basic notions of services described as activities composed of atomic activities. There are currently five ontology modules that extends the FLOWS-Core: **(i) Control Constraint**, includes workflow-style constructs (e.g., split, sequence, if then else, etc) **(ii) Ordering Constraint**, allows specification of sequencing properties of atomic processes **(iii) Occurrence Constraints**, allows specification of nondeterministic activities within services **(iv) State Constraints**, allows specification of activities which are triggered by states that satisfy a given condition **(v) Exception Constraints**, supplies basic constructs for modelling exceptions.
- **Grounding**, as in OWL-S, the role of grounding is to provide concrete details (e.g., message formats, transport protocols, and network addresses) of how to access a given service.

Chapter 5

Portals as Example for a Modularized Application (by UPM)

A good reference of using several of these Semantic Web Components is the Knowledge Web (KW) Semantic Portal[1]. It is a software infrastructure underpinning the integration of the activities of the KW partners. It serves as portal for information access and a dissemination point for ontology researchers, engineers, application and content developers in both academic and industrial institutions. It provides a common medium of presentation where the partners' development work is deployed, publicized and promoted, along with work on technology promotion, research and e-learning.

The KW Semantic Portal is the latest instantiation of ODESeW [Corcho *et al.*, 2003]. This is an ontology-based application built inside the WebODE ontology engineering workbench, that allows managing knowledge-intensive ontology-based Intranets and Extranets.

In order to achieve these objectives, the KW Semantic Portal uses some of the Semantic Web Components mentioned before.

5.1 Ontology Development Components

Ontology Editor . The KW Semantic Portal uses this component to provide the *Semantic Editing* functionality. This consists of providing content to the KW Semantic Portal by allowing internal users to edit concept instances and the values of their attributes, and to connect such instances by means of relations, even if they belong to different ontologies. With this component, the users are also able to create and remove an instance of a certain concept, or even to move an instance from one concept to another.

[1] http://knowledgeweb.semanticweb.org/

Ontology Visualization . This component is used by the KW Semantic Portal because it must provide the appropriate view for each user and for each situation, that is, depending on the user's permissions and its situation in the current navigation model. It is represented in the portal by the *Semantic Visualization* functionality.

5.2 Data Layer Components

Storing Ontology Content and Instances . The KW Semantic Portal uses this component because its ontologies are safely stored in an Ontology Server, namely WebODE.

In fact, when a user edits any of the ontologies published with ODESeW using the WebODE ontology editor, (s)he can observe at run-time the modifications in the KW portal, which means that there is auto-synching of the portal with respect to the ontology server. This is all represented by the *Ontology Repository* system.

Ontology Querying . This component is used by the KW Semantic Portal, and it is represented by the *Semantic Searching* functionality. It implements the search engine that allows querying for information in one or in all the ontologies of the portal. There are two types of searching: *Search In Term Names* and *Search In Instance Values*.

With the first one, the search engine looks for instances or concept names that contain the keywords specified in the query.

The last one is quite more complete. The KW Semantic Portal provides an advanced search function by means of a query form. The fields to be filled in the query form are attributes and relations taken from the ontology we are querying. Once the user introduces the values (s)he is looking for, the search engine returns the instances that satisfy the conditions imposed in the attributes values specified in the form.

5.3 Interface Layer Components

Ontology-Driven User Interfaces . This component is used by the KW Semantic Portal in order to perform a dynamic user interface. All the forms shown to the portal user are provided by the *Semantic Navigation Model Management* subsystem. For example, when editing a particular instance, the KW Semantic Portal obtains the instance information from the Ontology Server,and then shows the form to the user. The fields of this form represent the instance attributes and relations. This way, (s)he now can make the convenient changes in their values.

The semantic navigation is managed by another subsystem mentioned above, the *Semantic Navigation Model Management*.

A navigation model is implemented as an ontology inside the *Ontology Server*. It may be seen as an state diagram, in which the states are represented by concepts of the ontology (which actually are views), and the transitions between states are defined by relations between concepts (which actually are actions between different views). This way, each view has a name, description, precondition to be accomplished to retrieve the view and its location (URL). All these attributes are represented as concept attributes (class attributes) of the views.

When the user clicks on a hyperlink, (s)he is actually giving to the KW Semantic Portal the current view and the requested action. Afterwards, the portal will look up the navigation model and return to the user the destination view.

Chapter 6

Conclusion

As a step towards a Semantic Web framework, necessary components have been identified and their requirements determined. The decomposition of such an environment into components, and assignment of responsibilities and requirements will serve as an important foundation for the further framework development. As long as standardized components do not exist, such a decomposition can already serve as design guideline for large Semantic Web applications. Of course, the long-term goal is to specify and standardize component interfaces and protocols for their interaction.

Based on the requirements analysis we can identify cross-cutting functional requirements which are important for most components:

Ontology languages Of course, the fundamental condition for interoperability is the availability of shared languages for Ontology description. Standardized solutions already have emerged here. RDFS is a basic Ontology definition language which is used by several solutions. The usage of OWL also becomes more widespread. A third basis are rule-based definition languages which are needed especially for Ontology integration. The standardization is already advanced here, but unfortunately we don't have a strict layering where one language forms a subset of the next more powerful one. This makes interoperability of components based on different languages very difficult.

Storage access All components need access to the data layer (ontology and instance store, reasoner). This access can be split into two aspects, (i) directly storing and fetching items, and (ii) querying. For none of these, standardized solutions exist, and for the former there are not even standard processes started. For querying, an SQL-based standard language is currently in the process of being defined (SPARQL, [Prud'hommeaux and (eds.), 2005]). Also, a default reasoning API (DIG) has been defined which is supported by the major DL reasoners. In the world of databases, access standars as ODBC/JDBC together with a standardized query language (SQL) helped immensely to foster database technology usage. It seems that the Semantic Web area could benefit too from applying such a strategy

Version/Configuration management Ontologies and corresponding applications are continuously evolving. To make sure that all components and ontologies stay consistent with each other, we need a shared approach to configuration management. The listed solutions show that the consistency requirement can be tackled by checking adherence to formal specifications. Existing Semantic Web standards (e.g. OWL) alread allow to formulate such specifications and can be applied in this context, too. The only thing we need to make sure is that such a validation process has access to all configurations via a common API.

Another result of the requirements collection is that some non-functional requirements are of special importance across all components:

Scalability The most recurring topic in component requirements is scalability. This affects components in different ways (e.g. compare the issue of storing huge instance collections with the problem of visualizing ontologies consisting of thousands of concepts), but still should have an important impact when ensuring interoperability. Some of the existing solutions and even standards (e.g. OWL-Full) have not sufficiently considered the scalability question. When designing interfaces and formats for interoperability, we have to make sure that components can provide efficient implementions for these even when faced with huge ontologies and instance collections.

Distribution To support the vision of the semantic *web*, components need to work in a distributed fashion. Their interfaces have to provide remote access, and they in turn need to be able to exploit capabilities of remote components. Semantic Web Services seem to be the most important element in realizing the distribution requirement.

As with all software frameworks, it is important to build and analyse applications to find a good framework architecture and interface design. Therefore, one of the next steps should be the analysis of current applications built from existing components. The goal of this activity would be to evaluate how good current solutions do interoperate and to identify their interoperability weaknesses. The results of this work will provide the necessary experience to carry out the design and specification of interfaces and protocols to ensure that standard interfaces will meet applications needs and thus foster Semantic Web application development based on interoperable components.

Bibliography

[Aduna B.V., 2005] Aduna B.V. Aduna autofocus brochure, 2005. http://aduna.biz/products/autofocus/docs/autofocus_brochure.pdf.

[Akkiraju *et al.*, 2005] R. Akkiraju, J. Farrell, J. A. Miller, M. Nagarajan, M.-T. Schmidt, A. Sheth, and K. Verma. Web service semantics—WSDL-S, technical note, version 1.0. Technical report, International Business Machines Corporation and University of Georgia, 2005. http://lsdis.cs.uga.edu/library/download/WSDL-S-V1.html.

[Alonso *et al.*, 2003] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services*. Springer, Berlin, Germany, 2003.

[Ankolekar *et al.*, 2002] Anupriya Ankolekar, Mark H. Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew V. McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne, and Katia P. Sycara. Daml-s: Web service description for the semantic web. In *Proceedings of the First International Semantic Web Conference*, pages 348–363. Springer, 2002.

[Arpírez *et al.*, 2003] J. C. Arpírez, O. Corcho, M. Fernández-López, and A. Gómez-Pérez. Webode in a nutshell. *AI Magazine*, 24(4):37–48, 2003.

[Bechhofer *et al.*, 2003] Sean Bechhofer, Phillip Lord, and Raphael Volz. Cooking the Semantic Web with the OWL API. In *2nd International Semantic Web Conference (ISWC2003), Sanibel Island, Florida*, Oct 2003.

[Benatallah *et al.*, 2003] B. Benatallah, M-S. Hacid, C. Rey, and F. Toumani. Request Rewriting-Based Web Service Discovery. In *Proceedings of the 2nd International Semantic Web Conference (ISWC2003)*, volume 2870 of *LNCS*, pages 242–257, Sanibel Island, FL, USA, October 2003. Springer.

[Benatallah *et al.*, 2005] Boualem Benatallah, Mohand-Said Hacid, Alain Leger, Christophe Rey, and Farouk Toumani. On automating web services discovery. *The VLDB Journal*, 14(1):84–96, 2005.

[Berardi *et al.*, 2003] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic Composition of E-services That Export Their Behavior. In

Maria E. Orlowska, Sanjiva Weerawarana, Mike P. Papazoglou, and Jian Yang, editors, *ICSOC'03, Trento, Italy*, volume 2910 of *LNCS*, pages 43–58. Springer, December 2003.

[Berendt *et al.*, 2002] Bettina Berendt, Andreas Hotho, and Gerd Stumme. Towards semantic web mining. In *Proceedings of International Semantic Web Conference 2002*, pages 264–278, 2002.

[Berners-Lee *et al.*, 2001] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, pages 34–43, may 2001.

[Berners-Lee *et al.*, 2005] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifier (URI): Generic syntax. Technical report, The Internet Society, 2005. http://www.gbiv.com/protocols/uri/rfc/rfc3986.html.

[Bernstein and Klein, 2002] A. Bernstein and M. Klein. Discovering Services: Towards High Precision Service Retrieval. In *CaiSE workshop on Web Services, e-Business, and the Semantic Web: Foundations, Models, Architecture, Engineering and Applications. Toronto, Canada*, May 2002.

[Biron and Malhotra, 2001] P.V. Biron and A. Malhotra. Xml schema part 2: Datatypes. Technical report, World Wide Web Consortium, 2001. http://www.w3.org/TR/xmlschema-2/.

[Broekstra *et al.*, 2002] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *International Semantic Web Conference (ISWC-02)*, 2002.

[Bultan *et al.*, 2003] Tevfik Bultan, Xiang Fu, Richard Hull, and Jianwen Su. Conversation specification: a new approach to design and analysis of e-service composition. In *WWW 2003, Budapest, Hungary*, pages 403–410. ACM, May 2003.

[Chakraborty *et al.*, 2001] D. Chakraborty, F. Perich, S. Avancha, and A. Joshi. DReggie: Semantic Service Discovery for M-Commerce Applications. In *Workshop on Reliable and Secure Applications in Mobile Environment, 20th Symposium on Reliable Distributed Systems*, pages 28–31, October 2001.

[Chalupsky, 2000] Hans Chalupsky. OntoMorph: a translation system for symbolic knowledge. In *Proceedings of 7th international conference on knowledge representation and reasoning (KR), Breckenridge, (CO US)*, pages 471–482, 2000.

[Christensen *et al.*, 2001] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web service description language (WSDL) 1.1. Technical report, World Wide Web Consortium, 2001. http://www.w3c.org/TR/2001/NOTE-wsdl-20010315/.

[Ciravegna and Wilks, 2003] F. Ciravegna and Y. Wilks. Designing Adaptive Information Extraction for the Semantic Web in Amilcare. In S. Handschuh and S. Staab, editors, *Annotation for the Semantic Web*. IOS Press, Amsterdam, 2003.

[Corcho *et al.*, 2003] O. Corcho, A. Gómez-Pérez, A. Lopez-Cima, V. López-García, and M.C. Suárez-Figueroa. ODESeW. automatic generation of knowledge portals for intranets and extranets. In *Proceedings of the 2^{nd} International Semantic Web Conference (ISWC2003) Industrial Track*, pages 802–817, Sanibel Island, FL, USA, October 2003.

[Corcho, 2004] Oscar Corcho. *A declarative approach to ontology translation with knowledge preservation.* PhD thesis, Universidad Politécnica de Madrid, 2004.

[Cunningham *et al.*, 2002] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*, 2002.

[Curbera *et al.*, 2001] F. Curbera, W. A. Nagy, and S. Weerawana. Web service: Why and how? In *Proceedings of the OOPSLA-2001 Workshop on Object-Oriented Services*, Tampa, FL, USA, 2001.

[DAML, 2005] DAML web site, 2005. http://www.daml.org/.

[DCMI Usage Board, 2005] DCMI Usage Board. DCMI metadata terms, 2005. http://dublincore.org/documents/dcmi-terms/.

[Dean and Schreiber, 2004] Mike Dean and Guus Schreiber. OWL web ontology language. Technical report, World Wide Web Consortium, 2004. http://www.w3.org/TR/owl-ref/.

[Domingue *et al.*, 2004] J. Domingue, M. Dzbor, and E. Motta. Magpie: Supporting Browsing and Navigation on the Semantic Web. In N. Nunes and C. Rich, editors, *Proceedings ACM Conference on Intelligent User Interfaces (IUI)*, pages 191–197, 2004.

[Dou *et al.*, 2003] Dejing Dou, Drew V. McDermott, and Peishen Qi. Ontology translation on the semantic web. In *Proceedings of the Eleventh International Conference on Cooperative Information Systems*, 2003.

[Euzenat and Stuckenschmidt, 2003] Jérôme Euzenat and Heiner Stuckenschmidt. The 'family of languages' approach to semantic interoperability. In Borys Omelayenko and Michel Klein, editors, *Knowledge transformation for the semantic web*, pages 49–63. IOS press, Amsterdam (NL), 2003.

[Euzenat and Tardif, 2002] Jérôme Euzenat and Laurent Tardif. Xml transformation flow processing. *Markup languages: theory and practice*, 3(3):285–311, 2002.

[Euzenat, 2004] Jérôme Euzenat. An api for ontology alignment. In *Proc. 3rd international semantic web conference, Hiroshima (JP)*, pages 698–712, 2004.

[FaCT, 1998] FaCT. `http://www.cs.man.ac.uk/~horrocks/FaCT/`, 1998.

[FaCT++, 2003] FaCT++. `http://owl.man.ac.uk/factplusplus/`, 2003.

[Fensel and Bussler, 2002] Dieter Fensel and Christoph Bussler. The web service modeling framework wsmf. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

[Fensel *et al.*, 2003] Dieter Fensel, Enrico Motta, Frank van Harmelen, V. Richard Benjamins, Monica Crubézy, Stefan Decker, Mauro Gaspari, Rix Groenboom, William E. Grosso, Mark A. Musen, Enric Plaza, Guus Schreiber, Rudi Studer, and Bob J. Wielinga. The unified problem-solving method development language upml. *Knowl. Inf. Syst.*, 5(1):83–131, 2003.

[Fensel, 1997] Dieter Fensel. The tower-of-adapter method for developing and reusing problem-solving methods. In *Proceedings of the 10th European Workshop on Knowledge Acquisition, Modeling and Management*, pages 97–112, Sant Feliu de Guixols, Spain, 1997. Springer.

[Glimm and Horrocks, 2004] Birte Glimm and Ian Horrocks. Query answering systems in the semantic web. In *CEUR workshop proceedings of KI-2004 Workshop on Applications of Description Logics (ADL 2004)*, September 24 2004.

[Gómez-Pérez, 2004a] Asunción Gómez-Pérez. *Ontological engineering*. Springer-Verlag, 2004.

[Gómez-Pérez, 2004b] Asunción Gómez-Pérez. A survey on ontology tools. ontoweb deliverable 1.3. Technical report, OntoWeb Project, 2004. http://ontoweb.org/About/Deliverables/D13_v1-0.zip.

[González-Castillo *et al.*, 2001] J. González-Castillo, D. Trastour, and C. Bartolini. Description Logics for Matchmaking of Services. In *KI-2001 Workshop on Applications of Description Logics Vienna, Austria*, September 2001. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-44/.

[Gudgin *et al.*, 2003] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen. SOAP version 1.2 part 1: Messaging framework. Technical report, World Wide Web Consortium, 2003. http://www.w3.org/TR/soap12.

[Haarslev and Möller, 2003] V. Haarslev and R. Möller. Racer: A core inference engine for the semantic web. In *EON*, 2003.

[Handschuh *et al.*, 2002] S. Handschuh, S. Staab, and F. Ciravegna. S-CREAM — Semi-automatic CREAtion of Metadata. In *13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02)*, pages 358–372, Siguenza, Spain, 2002.

[Hendler *et al.*, 2003] J. Hendler, D. Nau, B. Parsia, E. Sirin, and D. Wu. Automating DAML-S Web Services Composition Using SHOP2. In *In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, International Semantic Web Conference (ISWC 2003), Sanibel Island, FL, USA*, volume 2870 of *LNCS*, pages 195–210. Springer, October 2003.

[Hendler, 2001] James A. Hendler. Agents and the semantic web. *IEEE Intelligent Systems*, 16(2):30–37, 2001.

[Horrocks, 2002] Ian Horrocks. DAML+OIL: A Reasonable Web Ontology Language. In *Proceedings of EDBT'2002 Prague, Czech Republic*, pages 2–13, March 2002.

[Hull *et al.*, 2003] Richard Hull, Michael Benedikt, Vassilis Christophides, and Jianwen Su. E-services: a look behind the curtain. In *Proc. 22th Principles of Database Systems (PODS'03), San Diego, CA, USA*, pages 1–14. ACM, June 2003.

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

[Kayne, 2003] D. Kayne. *Loosely Coupled, The Missing Pieces of Web Services*. Rds Associates Inc., 2003.

[Kifer *et al.*, 1995] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *J. ACM*, 42(4):741–843, 1995.

[Kreger, 2001] H. Kreger. Web services conceptual architecture (WSCA 1.0). Technical report, IBM Software Group, 2001. http://www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf.

[Lara *et al.*, 2004] Rubén Lara, Dumitru Roman, Axel Polleres, and Dieter Fensel. A conceptual comparison of wsmo and owl-s. In *Proceedings of the European Conference on Web Services*, pages 254–269, Erfurt, Germany, 2004.

[Lausen *et al.*, 2005] Holger Lausen, Axel Polleres, and Dumitru Roman. Web service modeling ontology (WSMO). Technical report, World Wide Web Consortium, 2005. http://www.w3.org/Submission/WSMO/.

[Mädche *et al.*, 2002] Alexander Mädche, Boris Motik, Nuno Silva, and Raphael Volz. MAFRA – a mapping framework for distributed ontologies. In *Proc. ECAI workshop on Knowledge Transformation for the Semantic web, Lyon (FR)*, pages 60–68, 2002.

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

[Maynard *et al.*, 2002] D. Maynard, V. Tablan, H. Cunningham, C. Ursu, H. Saggion, K. Bontcheva, and Y. Wilks. Architectural Elements of Language Engineering Robustness. *Journal of Natural Language Engineering – Special Issue on Robust Methods in Analysis of Natural Language Data*, 8(2/3):257–274, 2002.

[Maynard *et al.*, 2004] D. Maynard, M. Yankova, N. Aswani, and H. Cunningham. Automatic Creation and Monitoring of Semantic Metadata in a Dynamic Knowledge Portal. In *Proceedings of the 11th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA 2004)*, Varna, Bulgaria, 2004.

[McBride, 2001] Brian McBride. Jena: Implementing the rdf model and syntax specification. In *Proceedings of SemWeb 2001*, 2001.

[McIlraith *et al.*, 2001] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[Miller *et al.*, 2004] J. A. Miller, K. Verma, P. Rajasekaran, A. Sheth, R. Aggarwal, and K. Sivashanmugam. WSDL-S: A proposal to the W3C WSDL committee. Technical report, University of Georgia, 2004. http://lsdis.cs.uga.edu/projects/WSDL-S/wsdl-s.pdf.

[MOF, 2002] MOF. Metaobjectfacility(MOF) specification. Technical report, Object Management Group, Inc., 2002. http://www.omg.org/docs/formal/02-04-03.pdf.

[Motta *et al.*, 2002] E. Motta, M. Vargas-Vera, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna. MnM: Ontology Driven Semi-Automatic and Automatic Support for Semantic Markup. In *13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02)*, pages 379–391, Siguenza, Spain, 2002.

[Motta *et al.*, 2003] Enrico Motta, John Domingue, Liliana Cabral, and Mauro Gaspari. Irs-ii: A framework and infrastructure for semantic web services. In *Proceedings of the Second International Semantic Web Conference*, pages 306–318, Sanibel Island, FL, USA, 2003.

[Motta, 1999] Enrico Motta. *Reusable Components for Knowledge Modelling*. IOS Press, Amsterdam, Netherlands, 1999.

[Narayanan and McIlraith, 2002] S. Narayanan and S.A. McIlraith. Simulation, verification and automated composition of web services. In *International World Wide Web Conference*, pages 77–88, May 2002.

[Omelayenko, 2002] Borys Omelayenko. Integrating vocabularies: discovering and representing vocabulary maps. In *Proc. 1st International Semantic Web Conference (ISWC-2002), Chia Laguna (IT)*, pages 206–220, 2002.

[OWL Services Coalition, 2004] OWL Services Coalition. OWL-S 1.0 release: Semantic markup for web services. Technical report, DAML Program, 2004. http://www.daml.org/services/owl-s/1.0/owl-s.pdf.

[Palmér *et al.*, 2004] M. Palmér, A. Naeve, and F. Paulsson. The SCAM framework: Helping semantic web applications to store and access metadata. In *Proceedings of the European Semantic Web Symposium 2004*, 2004.

[Pan, 2004a] Jeff Z. Pan. *Description Logics: Reasoning Support for the Semantic Web*. PhD thesis, School of Computer Science, The University of Manchester, Oxford Rd, Manchester M13 9PL, UK, 2004.

[Pan, 2004b] Jeff Z. Pan. Reasoning Support for OWL-E (Extended Abstract). In *Proc. of Doctoral Programme in the 2004 International Joint Conference of Automated Reasoning (IJCAR2004)*, July 2004.

[Paolucci *et al.*, 2002] M. Paolucci, T. Kawamura, T.R. Payne, and K.P. Sycara. Semantic Matching of Web Services Capabilities. In *Int. Semantic Web Conference, Sardinia, Italy*, pages 333–347, June 2002.

[Payne *et al.*, 2001] T.R. Payne, M. Paolucci, and K. Sycara. Advertising and Matching DAML-S Service Descriptions (position paper). In *International Semantic Web Working Symposium, Stanford University, California, USA*, July 2001.

[Popov *et al.*, 2004] B. Popov, A. Kiryakov, A. Kirilov, D. Manov, D. Ognyanoff, and M. Goranov. KIM – Semantic Annotation Platform. *Natural Language Engineering*, 2004.

[Prud'hommeaux and (eds.), 2005] Eric Prud'hommeaux and Andy Seaborne (eds.). SPARQL query language for RDF. Technical report, World Wide Web Consortium, 2005. http://www.w3.org/TR/rdf-sparql-query/.

[Racer, 1999] Racer. `http://www.sts.tu-harburg.de/~r.f.moeller/racer/`, 1999.

[Sintek and Decker, 2002] M. Sintek and S. Decker. Triple–a query, inference, and transformation language for the semantic web. In *International Semantic Web Conference (ISWC-02)*, 2002.

[Sivashanmugam *et al.*, 2003] Kaarthik Sivashanmugam, Kunal Verma, Amit P. Sheth, and John A. Miller. Adding semantics to web services standards. In *Proceedings of the International Conference on Web Services*, pages 395–401, Las Vegas, Nevada, USA, 2003.

[Sivashanmugam *et al.*, 2005] Kaarthik Sivashanmugam, John A. Miller, Amit P. Sheth, and Kunal Verma. Framework for semantic web process composition. *International Journal of Electronic Commerce (IJEC)*, 9(2):71–106, 2005.

[Sollazzo *et al.*, 2002] Tanja Sollazzo, Siegfried Handschuh, Steffen Staab, Martin R. Frank, and Nenad Stojanovic. Semantic web service architecture – evolving web service standards toward the semantic web. In *Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference*, pages 425–429, Pensacola Beach, FL, USA, 2002.

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

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

[Stuckenschmidt and Wache, 2000] Heiner Stuckenschmidt and Holger Wache. Context modelling and transformation for semantic interoperability. In M. Bouzeghoub, M. Klusch, W. Nutt, and U. Sattler, editors, *Knowledge Representation Meets Databases (KRDB 2000) - CEUR Workshop Proceedings*, 2000.

[UML, 2003] UML. Unified modeling language (UML), version 1.5. Technical report, Object Management Group, Inc., 2003. http://www.omg.org/technology/documents/formal/uml.htm.

[Valente *et al.*, 1999] Andre Valente, Thomas Russ, Robert MacGregor, and William Swartout. Building and (re)using an ontology of air campaign planning. *IEEE Intelligent Systems*, 14(1):27–36, 1999.

[van der Aalst and van Hee, 2002] Wil M. P. van der Aalst and Kees M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, MA, USA, 2002.

[van der Aalst *et al.*, 2003] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[Züllighoven, 2005] Heinz Züllighoven. *The Object-Oriented Construction Handbook*. Elsevier, 2005.