D 1.2.5 Architecture of the Semantic Web Framework v2

Raúl García-Castro (UPM)

Óscar Muñoz-García (UPM), M. Carmen Suárez-Figueroa (UPM), Asunción Gómez-Pérez (UPM), Stefania Costache (L3S), Diana Maynard (USFD), Stamatia Dasiopoulou (CERTH), Raúl Palma (UPM), Vit Novacek (NUIG), Freddy Lécué (FT), Ying Ding (UIBK), Monika Kaczmarek (PUE), Ruzica Piskac (UIBK), Dominik Zyskowski (PUE), Jérôme Euzenat (INRIA), Martin Dzbor (OU), Lyndon Nixon (FU Berlin), Alain Léger (FT), Tomas Vitvar (NUIG), Michal Zaremba (NUIG), Jens Hartmann (UKARL)

Abstract.
EU-IST Network of Excellence (NoE) FP6-507482 KWEB
Deliverable D1.2.5 (WP1.2)
This deliverable continues with the definition of the SWF and its architecture. This framework contains the definition of the semantic-related components, the dependencies that exist between these components, and the implementations of components that could be used.

Document Identifier:	KWEB/2006/D1.2.5/v1.3
Class Deliverable:	KWEB FP6-507482
Version:	V1.3
Date:	February 11th, 2008
State:	Final
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 Polythechnique 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

University of Aberdeen (UNIABDN)
King's College
AB24 3FX Aberdeen
United Kingdom
Contact person: Jeff Pan
E-mail address: jpan@csd.abdn.ac.uk

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 the writing of the document:

France Telecom
Freie Universität Berlin
Institut National de Recherche en Informatique et en Automatique
National University of Ireland Galway
Learning Lab Lower Saxony
The Open University
Universidad Politécnica de Madrid
University of Innsbruck
University of Sheffield
Poznan University of Economics
Centre for Research and Technology-Hellas

Changes

Version	Date	Author	Changes
0.1	22-06-2007	Raúl García-Castro	First skeleton
0.2	24-07-2007	Raúl García-Castro	Inserted the component implementations
0.3	25-07-2007	Raúl García-Castro, Óscar Muñoz-García, Asunción Gómez-Pérez	Introduction
0.4	1-08-2007	Óscar Muñoz-García	Related Work section
0.5	5-08-2007	Raúl García-Castro, M. Carmen Suárez-Figueroa, Óscar Muñoz-García	Semantic Web Framework section
0.6	1-9-2007	Raúl García-Castro, Jérôme Euzenat, Óscar Muñoz-García	Modified *Ontology Engineering* dimension
0.7	28-11-2007	Martin Dzbor, Óscar Muñoz-García	Modified *Ontology Customization* dimension
0.8	1-12-2007	Óscar Muñoz-García, Raúl García-Castro	Updated component implementations
0.9	8-12-2007	Óscar Muñoz-García, Lyndon Nixon	Updated use cases
1.0	12-12-2007	Óscar Muñoz-García	Inclusion of comments by Rosario Plaza
1.1	20-12-2007	Óscar Muñoz-García, Raúl García-Castro	Inclusion of comments by the Quality Assessor (Asunción Gómez-Pérez)
1.2	7-2-2008	Óscar Muñoz-García, Raúl García-Castro	Inclusion of comments by the Quality Controller (Lyndon Nixon)
1.3	11-2-2008	Óscar Muñoz-García, Raúl García-Castro	Final version

Executive Summary

Developing Semantic Web applications is a difficult task for people outside the research community. To ease the construction of Semantic Web Applications, in this deliverable we provide a definition of the Semantic Web Framework, a structure in which Semantic Web applications can be organised and developed.

The architecture of the Semantic Web Framework classifies the different types of Semantic Web technologies according to their functionalities and represents these technologies as independent components. This component-based framework contains the definition of the semantic-related software components that can be used in the development of Semantic Web applications, the dependencies that exist between these components, and it also lists the existing software that implement the components.

This deliverable also represents the architecture of the Knowledge Web use cases (described in D1.1.4 v1) using the Semantic Web Framework components.

The document, extends and replaces the deliverable D1.2.4 [1] improving the previous version by:

- Identifying existing implementations of the components.
- Contrasting the definitions of the use cases with the systems developed using the Semantic Web Framework dimensions and components.
- Refining the definition of the Semantic Web Framework dimensions and their components.

Contents

1 Introduction

Semantic Web technology is being used beyond the borders of the research world and is reaching all kinds of users ranging from individuals to companies. When developing Semantic Web applications, IT developers have to face several obstacles:

- IT developers do not know precisely the types of Semantic Web technologies and the functionalities that Semantic Web technologies provide, nor do they know the dependencies between them.
- For naive Semantic Web users, it is not easy to know how to use existing Semantic Web technologies and how to reuse or include them when building Semantic Web applications.
- It is not always known how Semantic Web technologies interoperate with other Semantic Web technologies or with existing IT systems.
- They cannot accurately make decisions when developing Semantic Web applications such as estimating the cost and resources needed when semantic capabilities are included into legacy applications or when new Semantic Web applications are built from scratch.

The Semantic Web Framework is intended to help Semantic Web application developers to build Semantic Web applications quicker and better and to solve the obstacles explained before. It is a reference framework that

- Describes the existing types of Semantic Web technologies, their functionalities, and the dependencies between these technologies.
- Identifies existing implementations for each type of Semantic Web technologies.
- Facilitates technology reuse by providing specifications and guidelines.
- Shows how to achieve interoperability between Semantic Web technologies.
- Helps application developers to make decisions when developing Semantic Web applications from scratch or when they introduce semantic into an application.

The first version of the Semantic Web Framework was presented in D1.2.4 [1], which was elaborated from the requirements stated in D1.2.2 [2]. This first version included initial definitions of the components that compose the Semantic Web Framework and the description of the use cases according to the Semantic Web Framework made by the deliverable authors. This deliverable extends and replaces the previous deliverable (D.1.2.4 [1]) by:

- **Identifying existing implementations of the components**. In order to speed up the construction of Semantic Web Applications, this deliverable identifies implementations of the different components, taking into account existing

implementations in the Web and those in the *Semantic Web Tools and Applications Information Repository*[1] developed inside workpackage 1.4.

- **Contrasting the definitions of the use cases with the systems developed**. In order to give recommendations for the Knowledge Web business use cases and to perform a first validation of the Semantic Web Framework v1, the use cases described in D1.1.4 v1 [3] have been redefined using the components included in this version of the deliverable.
- **Refining the definition of the Semantic Web Framework components**. The two tasks above produced feedback that allowed us to update and complete the definition and the dependencies of the Semantic Web Framework components.

This deliverable is structured as follows:

- Chapter 2 describes the related work about Component-based Software Development, Software Architectures and Frameworks, Semantic Web Applications and Semantic Web Application Architectures.
- Chapter 3 presents the Semantic Web Framework itself.
- Chapter 4 updates the definitions of the components of the Semantic Web Framework v1 and their dependencies.
- Chapter 5 examines the semantic functionalities of the use cases according to the Semantic Web Framework v2.
- Chapter 6 draws some conclusions and outlines future lines of this work.
- Appendix I show in tables the dependencies between all the components of the Semantic Web Framework.
- Appendix II show in tables the dependencies between the use cases and the components of the Semantic Web Framework.
- Appendix III includes the implementations found for the components described in Chapter 3.

[1] http://cgi.csc.liv.ac.uk/KWebToolsSurvey/

2 Related Work

2.1 Component-based Software Development

In Component-based Software Development, application developers reuse software components already developed and tested in order to build their applications in a robust and quick way. In this scenario, it is normally known the components interfaces or contracts given by the components developers, but it is unknown the details about the components implementation or the way the components were conceived to be used.

A software component is a software composition unit that specifies a set of interfaces and a set of requirements. A software component can be composed of other components independently in time and space [4].

Component-based systems have the following characteristics [5]:

- **Interoperability**. Components cooperate despite differences in language, interface, and execution platform.
- **Distribution**. Components can be hosted in different machines in a network.
- **Heterogeneity**. Components can be executed in different platforms or operating systems and written in different languages by different developers.
- **Extensibility independence**. The applications are modifiable and extensible allowing the addition of new components.
- **Dynamism**. Applications can evolve by extending, extinguish, or substituting components, or by reconfiguring the relationships between components.

2.2 Software Architectures and Frameworks

Software architecture is defined as the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution [6].

The objectives of software architectures are 1) to understand and improve complex application structures, 2) to reuse the application structure to solve similar problems, 3) to plan the application evolution, 4) to analyse the application correction and the compliance degree with respect to the initial requirements, and 5) to allow the study of some domain specific parts.

A **framework** is a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact [7]. Frameworks define a set of components and their interfaces in an abstract way, establishing the interaction rules and mechanisms between them. Frameworks are a kind of domain-specific software architecture [8] that define the architectural style relating the components inside a system.

2.3 Semantic Web Applications

The Semantic Web is an extension of the current web, in which information is given well-defined meaning, and which better enables computers and people to work in

cooperation [9]. Semantic Web Applications have been characterized by different authors [10] and by events such as the Semantic Web Challenge[2] with the following features:

- Data is represented using formal descriptions.
- Semantic data is reused, manipulated and processed.
- Data sources are heterogeneous and are owned or controlled by different organisations.
- Applications assume an open world (i.e. the information is never complete).
- Multiple natural languages are supported.
- RDF(S) and OWL, the open standards recommended by the W3C, are used.

In addition, Motta et al. define the features for a next generation of Semantic Web Applications [10] as follows:

- Semantic data can be defined in terms of many different ontologies.
- Semantic Web Applications must scale in terms of the amount of data used and of distributed components working together.
- Semantic Web Applications ought to embed Web 2.0 features.

In the Semantic Web, reuse appears not only at the data level, as shown above, but also at the application level. Nowadays we can find much open software from a wide range of sources that can be reused when building Semantic Web Applications. In the application level, semantic software reuse follows three different approaches: a distributed services approach, by integrating web service technology in their architectures; a shared memory approach, by composing components that use a shared space of common memory to communicate, as in the case of reusing libraries inside an application; and a mixed approach, by combining the two other approaches.

2.4 Semantic Web Application Architectures

Mika et al. sketch a generic architecture of ontology-based applications (see Figure 1) based in a call-and-return style and structured in three hierarchical layers [11]. The layers involved are from bottom to top: ontology, middleware and application. The ontology layer contains the components concerned with the creation and maintenance of the model of the application; the middleware layer supplies common ontology-related services; and the application layer builds on the ontology and related services to provide some kind of ontology functionality to an end user. They follow the architectural styles classification presented in [12].

[2] http://iswc2007.semanticweb.org/callfor/SemanticWebChallenge.asp

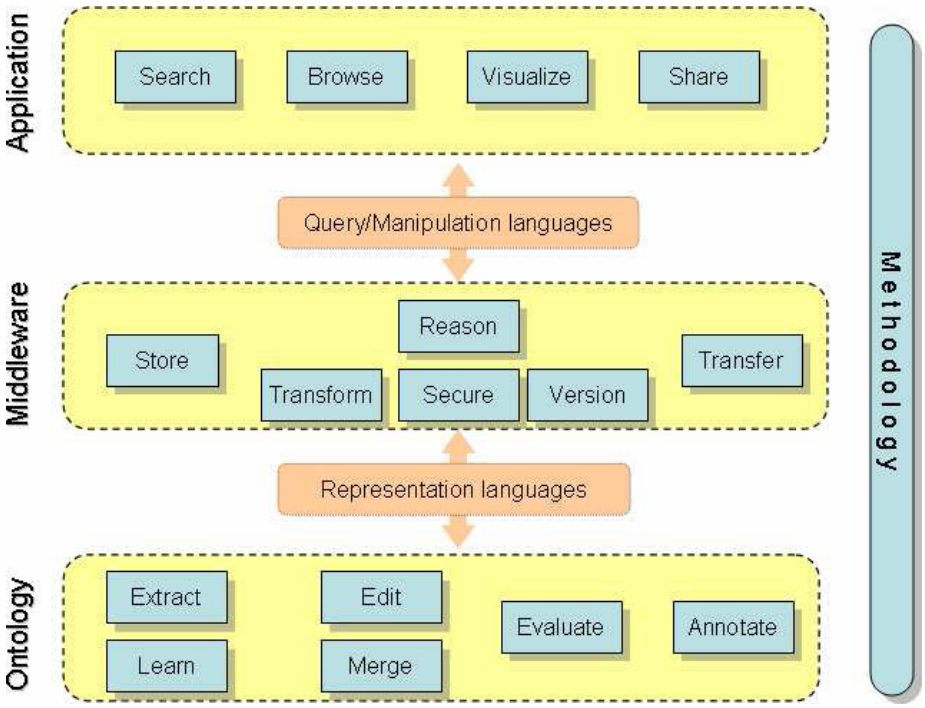

Figure 1. A generic architecture of ontology-based applications taken from [11].

Thanh et al. [13] present a service oriented architecture (see Figure 2) also structured in hierarchical layers: the data layer hosts any kind of data sources, including others different from ontological sources; the logic layer includes application-specific services that are implemented for a particular use case and operate on specific object models; the presentation layer hosts presentation components that the user interacts with. They also classify the components inside the logic layer into ontology services, ontology engineering services and ontology usage services.

The Semantic Web Technology is composed of heterogeneous systems. Therefore, the framework described in this document is an open system and is not divided in layers. Some of the disadvantages of layered approaches are the difficulty in structuring some systems in a layered fashion; performance considerations when high level functions require close coupling to low level implementations; and the difficulty to find the right level of abstraction, especially if existing systems cross several layers [12].

The main differences between the two architectures presented before and the Semantic Web Framework is that the first identify some example components for illustrating their approaches while in the Semantic Web Framework we have tried to exhaustively identify the existing semantic components of Semantic Web applications. The 33 components identified in the Semantic Web Framework cover 16 and 21 components identified in the previous approaches. On the other hand, [11] and [13] do not identify the exhaustive list of relationships and dependences between the components, nor do they list a set of software implementations of the components.

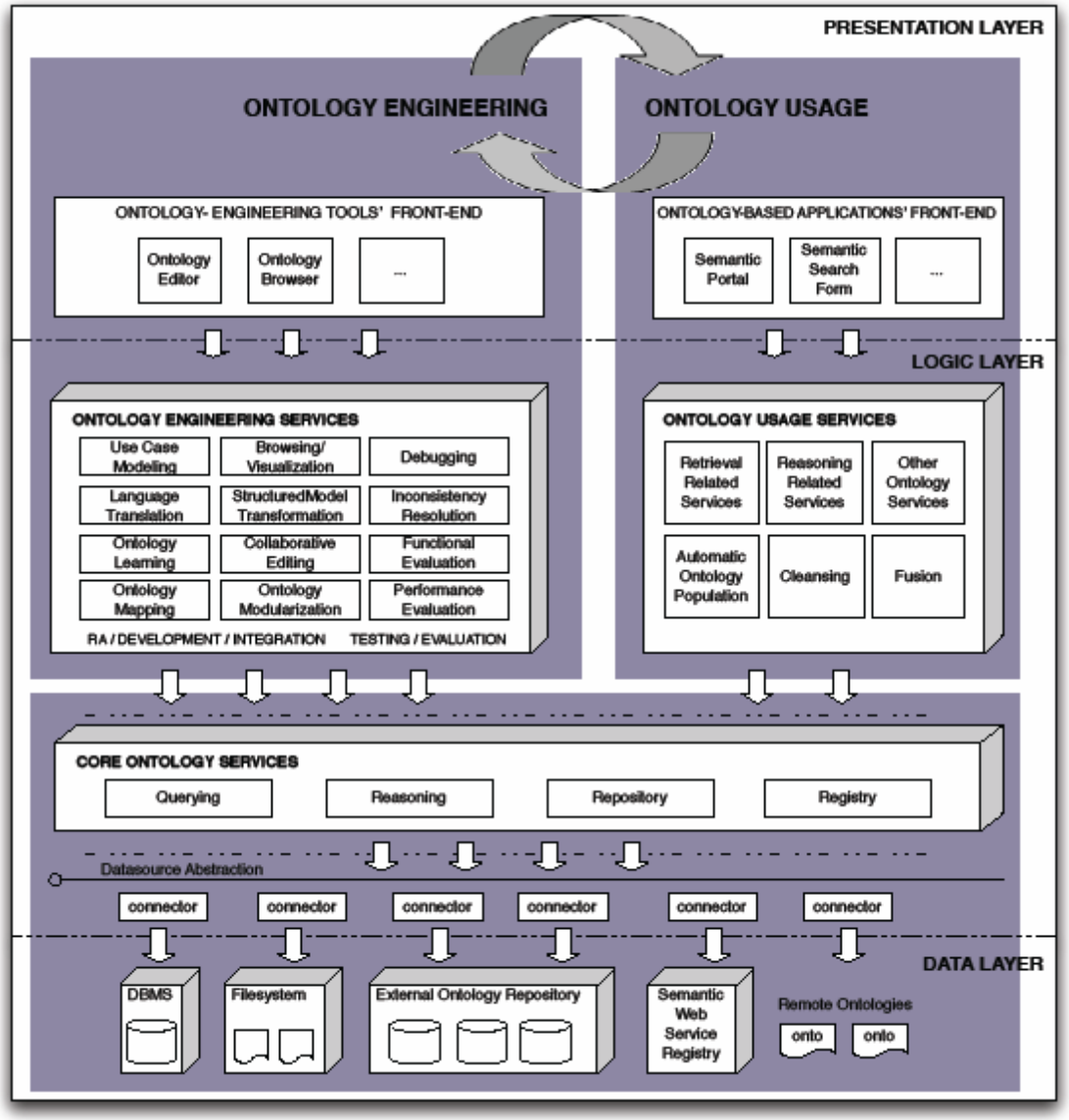

Figure 2. A generic architecture taken from [13].

3 Semantic Web Framework

In this deliverable, the Semantic Web Framework (SWF) is defined as a structure in which Semantic Web applications can be organized and developed.

3.1 Design principles of the Semantic Web Framework

The Semantic Web Framework is guided by general design principles. These principles state that the Semantic Web Framework should be as follows:

- **Developer-oriented.** To consider different audiences such as developers with low expertise with Semantic Web technologies or ontology practitioners.
- **Easy to understand.** To facilitate the understanding and use of the Semantic Web Framework, its components have been organised in dimensions according to the major properties of the problem space that have significant variation over Semantic Web technology.
- **Inexpensive to adopt**. To develop a Semantic Web application or to upgrade an existing application with semantic capabilities should be easy and thus, the impact on legacy systems to be enriched with semantics is minimized.
- **Semantics focused**. To describe only the components that provide semantic functionalities and functionalities to manage semantics. Other components that deal with communication, distribution, etc. are not taken into account to ease the integration of the components of the Semantic Web Framework in other software architectures.
- **Component based.** To define some specifications of these components that allow different implementations of them, providing each of these components a basic functionality.
- **Evolving**. To extend easily the Semantic Web Framework by inserting new components or modifying the existing ones because the Semantic Web, and also its technology, is continuously evolving.

The Semantic Web Framework has been defined as a component-based framework because Semantic Web applications possess similar characteristics to those of component-based systems described above: interoperability, distribution, heterogeneity, extensibility independence, and dynamism.

Furthermore, component-based frameworks provide the following features that facilitate software reuse [14]:

- **Abstraction**, to reduce and factor out details;
- **Selection**, to help developers locate, compare and select reusable software artefacts;
- **Specialisation**, to allow specialising generic artefacts; and
- **Integration**, to combine a collection of artefacts.

According to the definition of software architecture presented in Section 2.2, to define the architecture of the Semantic Web Framework we need to identify its components and their interaction.

In this deliverable, we focus on the identification of the components of the Semantic Web Framework; on their classification, as stated below; and on the main interfaces of these components with other components and with the environment. In a future work, we will define a concrete specification of the interfaces, the patterns that describe the composition of the components, and the restrictions when applying those patterns.

3.2 Definition and classification of the components

We follow the definition of component given by Szyperski [4] since a Semantic Web Framework component is an autonomous and modular unit with well defined interfaces that describes a service and performs a specific functionality. Such components can be used either independently or together to develop applications for the Semantic Web. Components in this sense can be divided into four types: services, program libraries, applications, and protocols.

Components are usually defined by specifying some *general information* about them, such as a natural language description, their *interfaces*, including the functionalities that the component implements and those that it uses, and their *contracts*, which are specifications added to the interface, and which establish the use and implementation conditions [4].

In this version of the Semantic Web Framework, we do not describe the component contracts, which will be defined in future works; we explicitly divide the interfaces into the functionalities that a component implements and those that it uses. Therefore, each component is defined by the following fields:

- **Name**. The name of the component.
- **Description**. A high-level description of the component.
- **Functionalities provided**. An enumeration of the functionalities that the component provides, specifying for each functionality the type or types of interface that it provides (user interface, programming interface, service interface, hardware interface, etc.).
- **Component dependencies**. An enumeration of the functionalities required by the component for working correctly and provided by other components.

To classify the components of the Semantic Web Framework, we have considered the dimensions of an architecture as the major properties of the problem space that have significant variation over the systems of concern to the architecture, in other words, the groups of components that provide some specific support to the architecture. These dimensions are subjective. In this deliverable we have classified the different components according to the main functionalities that they provide. Furthermore, these dimensions are not exhaustive.

The dimensions considered are the following:

- ***Data and metadata management***. This dimension includes those components that manage knowledge and data sources.

- ***Querying and reasoning***. This dimension includes those components that generate and process queries.
- ***Ontology engineering***. This dimension includes those components that provide functionalities to develop and manage ontologies.
- ***Ontology customization***. This dimension includes the components that customize ontologies.
- ***Ontology evolution***. This dimension includes those components that manage the ontology evolution.
- ***Ontology instance generation***. This dimension includes those components that generate ontology instances.
- ***Semantic web services***. This dimension includes those components that discover, adapt/select, mediate, compose, choreograph, ground, and profile semantic web services.

4 Components of the Semantic Web Framework

Figure 3 presents the components of the Semantic Web Framework that have been identified from those currently available or under construction. The enumeration of components is neither exhaustive nor complete, and is open to improvements and extensions in future works.

In Figure 3, each dimension of the architecture is represented as a column and reflects those components that provide a particular functionality to the architecture. The order of the components or of the dimensions in the figure does not imply any precedence or relation between them.

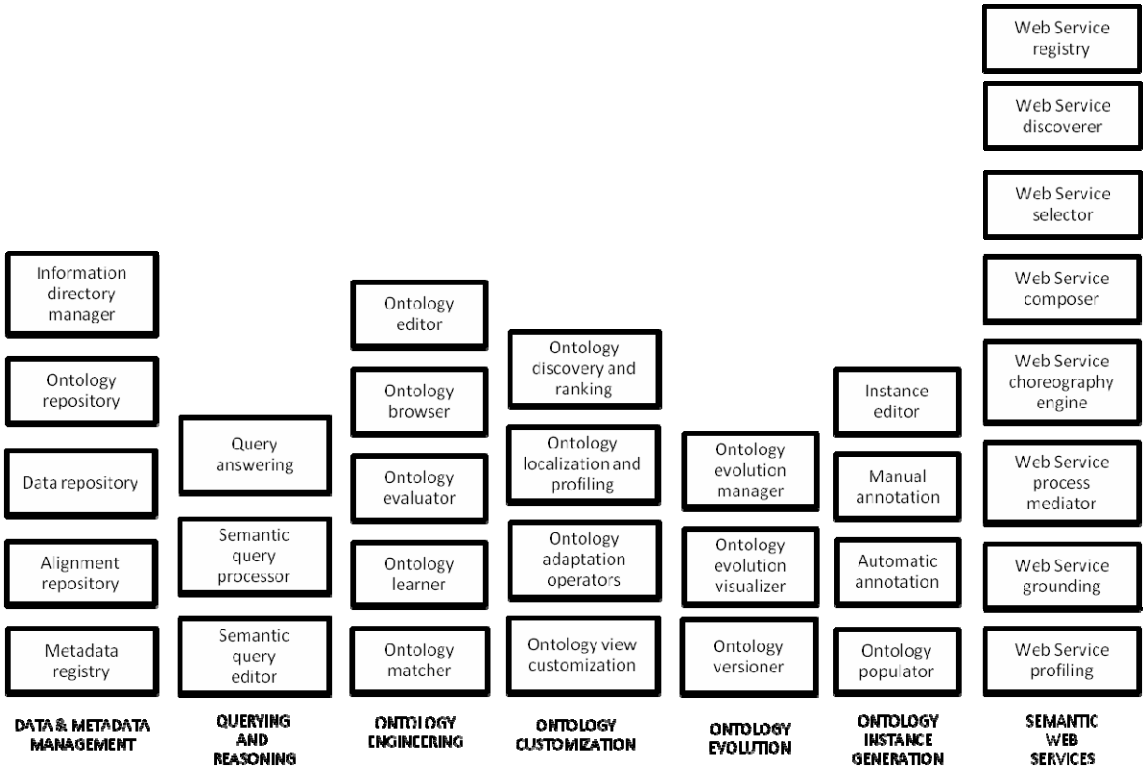

Figure 3. Components of the Semantic Web Framework v2.

Next a brief description of the components of the Semantic Web Framework v2 is shown. This description will be extended in the subsections that deal with each dimension in this section.

- *Data and metadata management*:
  - ***Information directory manager* component**. This component provides functionalities to handle query distribution, to manage a content provider directory, to identify information providers from a query, and to handle the storage and access to distributed ontologies and data.
  - ***Ontology repository* component**. This component provides functionalities to locally store and access ontologies and ontology instances.

  - ***Data repository*** **component**. This component provides functionalities to locally store and access data and ontology annotated data.
  - ***Alignment repository*** **component**. This component provides functionalities to handle the storage and access to distributed alignments.
  - ***Metadata registry*** **component**. This component provides functionalities to locally store and access metadata information.
- *Querying and reasoning*:
  - ***Query answering*** **component**. This component takes care of all the issues related to the logical processing of a query by providing reasoning functionalities to search results from a knowledge base.
  - ***Semantic query processor*** **component**. This component takes care of all issues related to the physical processing of a query, by providing functionalities to manage query answering over ontologies in distributed sources.
  - ***Semantic query editor*** **component**. This component takes care of all the issues related to the user interface.
- *Ontology engineering*:
  - ***Ontology editor*** **component**. This component provides functionalities to create and modify ontologies, ontology elements, and ontology documentation.
  - ***Ontology browser*** **component**. This component provides functionalities to browse an ontology visually.
  - ***Ontology evaluator*** **component**. This component provides functionalities to evaluate ontologies, either their formal model or their content, in the different phases of the ontology life cycle.
  - ***Ontology learner*** **component**. This component provides functionalities to acquire knowledge and generate ontologies of a given domain through some kind of (semi)-automatic process.
  - ***Ontology matcher*** **component**. This component provides functionalities to match two ontologies and output some alignments.
- *Ontology customisation*:
  - ***Ontology localization and profiling*** **component**. This component provides functionalities to adapt an ontology according to some context or some user profile.
  - ***Ontology discovery and ranking*** **component**. This component provides functionalities to find appropriate views, versions or sub-sets of ontologies, and then to rank them according to some criterion.
  - ***Ontology adaptation operators*** **component**. This component is in charge of applying appropriate operators to the ontology in question, the result of which is an ontology customized according to some criterion.
  - ***Ontology view customisation*** **component**. This component is responsible for enabling the user to change or amend a view on a particular ontology in order to fit a particular purpose.
- *Ontology evolution*:
  - ***Ontology versioner*** **component**. This component allows maintaining, storing and managing different versions of an ontology.

  - ***Ontology evolution visualizer* component**. This component allows visualising different versions of an ontology.
  - ***Ontology evolution manager* component**. This component allows maintaining, storing and managing different versions of an ontology, and possibly visualising the versions within a broader context of complex ontology evolution and development platform.
- *Ontology instance generation*:
  - ***Instance editor* component**. This component provides functionalities to manually create and modify instances of concepts and of relations between them in existing ontologies.
  - ***Manual annotation* component**. This component is in charge of the manual and semi-automatic annotation of digital content documents (e.g. web pages) with concepts in the ontology.
  - ***Automatic annotation* component**. This component is in charge of the automatic annotation of digital content (e.g. web pages) with concepts in the ontology.
  - ***Ontology populator* component**. This component provides functionalities to automatically generate new instances in a given ontology from a data source.
- *Semantic web services*:
  - ***Web service discoverer* component**. This component provides functionalities to publish and search service registries, to control access to registries, and to distribute and delegate requests to other registries.
  - ***Web service selector* component**. After discovering a set of potentially useful services, this component needs to check whether the services can actually fulfil the user's concrete goal and under what conditions.
  - ***Web service composer* component**. This component will be in charge of the automatic composition of the web services in order to provide new value-added web services.
  - ***Web service choreography engine* component**. This component provides functionalities to use the choreography descriptions of both the service requester and provider to conduct the conversation between them.
  - ***Web service process mediator* component**. This component provides functionalities to reconcile the public process heterogeneity that can appear during the invocation of web services.
  - ***Web service grounding* component**. This component is responsible for the communication between web services.
  - ***Web service profiling* component**. This component provides functionalities to create web service profiles based on their execution history.
  - ***Web service registry* component**. This component provides functionalities to register semantic web services.

In the following subsections, component dependencies are represented graphically in the following way: when one component depends on the functionalities of another, it is then represented with an arrow that goes from the first component to the component that provides the functionalities. Furthermore, different colours have been given to the different components in order to facilitate the reading of the figures.

For example, Figure 4 shows the dependence between the *Ontology editor* and the *Ontology repository* components, where the *Ontology editor* component uses functionalities from the *Ontology repository* component to access and store ontologies and ontology elements.

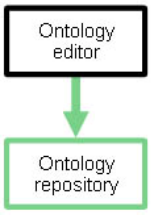

Figure 4. Dependence between the *Ontology editor* and the *Ontology repository* components.

The table below summarizes the dependencies found between the different dimensions, that is, if a component of a given dimension (A) depends on another component in other dimension (B), then, there is a potential dependency from A to B. The dependencies of a given dimension are shown in the rows, while the dimensions that are dependent on a given dimension are shown in the columns. For example, as shown in row (3) the *ontology engineering* dimension potentially requires components from the *data and metadata management*, *querying and reasoning* and *ontology customization* dimension; while the components in *data and metadata management* dimension are potentially required by the rest of the dimensions, as shown in column (1).

	(1)	**(2)**	**(3)**	**(4)**	**(5)**	**(6)**	**(7)**
***Data and metadata management* (1)**			X				
***Querying and reasoning* (2)**	X						
***Ontology engineering* (3)**	X	X		X			
***Ontology customization* (4)**	X						
***Ontology evolution* (5)**	X						
***Ontology instance generation* (6)**	X						
***Semantic web services* (7)**	X	X	X				

The next sections present the definition of the components for each of the dimensions; Appendix I includes tables that show the dependencies between all the components of the Semantic Web Framework.

4.1 Data and Metadata Management

This dimension includes those components that provide functionalities to manage knowledge and data sources.

The basic dependencies of the components in this dimension are shown in Figure 5:

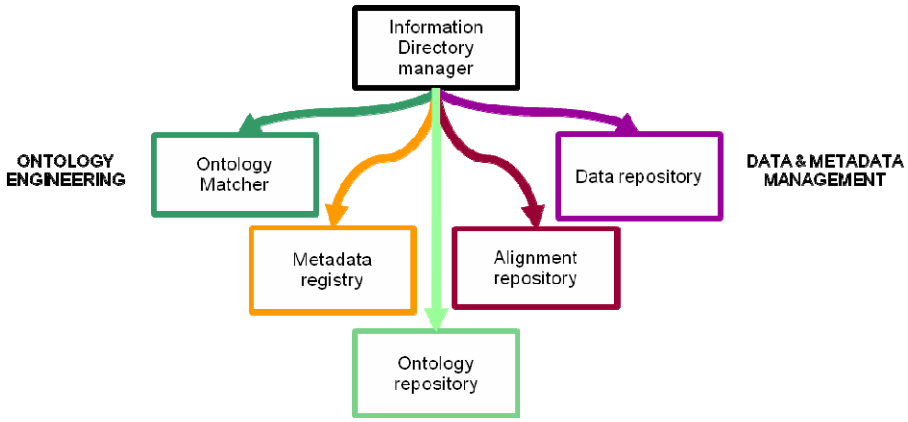

Figure 5. Dependencies of the components in the *Data and metadata management* dimension.

4.1.1 Information directory manager component

This component will be in charge of handling the distribution of a given query among the data providers, managing a content provider directory, identifying relevant information providers from a query and identifying provider self-descriptions, and handling the storage and access to distributed ontologies and data, independent of the particular representation formalism.

Functionalities provided

- It handles the distribution of a given query among the data providers.
- It supplies a unique mechanism for accessing different data and metadata (registry) sources.
- It supplies a SAIL, which is a storage and inference layer for accessing and synchronising different data and metadata (registry) repositories, and which also gets involved in the distributed query answering.
- It consists of various local data repositories and annotated data and metadata (registry) repositories that are synchronized with the help of a SAIL.

Component dependencies

This component uses

- The *Data repository* component to get access to the local data and annotated data sources, especially for answering a query but also for normal management functions, via an application interface.
- The *Metadata registry* component to get access to the local metadata repositories, especially for query answering but also for normal management functions, via an application interface.
- The *Ontology matcher* component (described below in the *Ontology engineering* dimension) to manage query answering, using its data translation functionalities, independent of the representational form of the data and metadata, via an application interface.

- The *Ontology repository* component to get access to the local ontologies descriptions, especially for query answering but also for normal management functions, via an application interface.
- The *Alignment repository* component to get access to the local alignments, especially for query answering but also for normal management functions, via an application interface.

4.1.2 Ontology repository component

The *Ontology repository* component provides functionalities to store and access ontologies and ontology instances locally. Optionally, the ontology repository can be distributed and, therefore, it will provide transparent access to ontologies and ontology instances logically and physically distributed.

Functionalities provided

- Storage capabilities that
  - Provide a defined protocol to access ontologies.
  - Support standard ontology query languages
  - Offer a virtual unique storage space
  - Provide fault tolerance mechanisms by ensuring access to the system in the case of a failure of the central server.
  - Implement caching mechanisms to improve performance of resource retrieval.
  - Manage change propagation automatically when an ontology is updated (i.e. to ensure consistency of dependent artefacts e.g. dependent ontologies, ontology instances, ontology metadata, etc.)
  - Scale its resources without any performance decrease
  - Allow access management to ontologies and ontology instances.
- Optionally, distributed storage capabilities that
  - Allow access to ontologies and ontology instances resources transparently from their location.
  - Manage internally the physical location where the ontologies and ontology instances are actually stored.
  - Provide high availability of ontologies and ontology instances by automatically distributing replicas, ensuring the consistency among them.
  - Provide fault tolerance mechanisms by ensuring access to the system regardless of whether any node of the distributed repository is temporarily unavailable

Component dependencies

For this component to communicate with the other components that it might interact with, it will send its requests and updates to the *Information Directory Manager*; any messages that the component needs to receive, it will get them through the *Information Directory Manager*.

4.1.3 Data repository component

The *Data repository* component provides functionalities to store and access any type of data (text, images, etc.) and ontology annotated data locally. Optionally, the data repository can be distributed and, therefore, it will provide transparent access to data and annotated data logically and physically distributed.

Functionalities provided

- Storage capabilities that
  - Provide a defined protocol to access data resources.
  - Offer a virtual unique storage space
  - Provide fault tolerance mechanisms by ensuring access to the system in the case of a failure of the central server.
  - Implement caching mechanisms to improve performance of resource retrieval.
  - Manage change propagation automatically when a data resource is updated (i.e. to ensure consistency of dependent artefacts e.g. data annotations, applications, related ontologies, etc.)
  - Scale its resources without any performance decrease
  - Allow access management to its resources
- Optionally, distributed storage capabilities that
  - Allow access to data and data annotations resources transparently from their location.
  - Manage internally the physical location where the data and data annotations are actually stored.
  - Provide high availability of data and data annotations by automatically distribute replicas, ensuring the consistency among them.
  - Provide fault tolerance mechanisms by ensuring access to the system regardless of whether any node of the distributed repository is temporarily unavailable

Component dependencies

For this component to communicate with the other components that it might interact with, it will send its requests and updates to the *Information Directory Manager*; any messages that the component needs to receive, it will get them through the *Information Directory Manager*.

4.1.4 Alignment repository component

This component will be in charge of providing functionalities for handling the storage and access to distributed alignments.

Functionalities provided

- Stores different defined alignments.
- Provides access, via a protocol, to different alignments.
- Allows the different alignments to be published into this distributed repository.

Component dependencies

For this component to communicate with the other components that it might interact with, it will send its requests and updates to the *Information Directory Manager*; any messages that the component needs to receive, it will get them through the *Information Directory Manager*.

4.1.5 Metadata registry component

The *Metadata registry* component provides functionalities to store and access metadata information (e.g., ontology metadata) locally. Optionally, the registry can be distributed and, therefore, it will provide transparent access to metadata information logically and physically distributed.

Functionalities provided

- Storage capabilities that
  - Provide a defined protocol to access metadata information.
  - Offer a virtual unique storage space.
  - Provide fault tolerance mechanisms by ensuring access to the system in the case of a failure of the central server.
  - Implement caching mechanisms to improve performance of metadata information retrieval.
  - Manage change propagation automatically when a metadata element is updated (i.e. to ensure consistency of dependent artefacts e.g. related ontologies.)
  - Scale its resources without any performance decrease.
  - Allow access management to metadata information.
- Optionally, distributed storage capabilities that
  - Allow access to metadata information transparently from their location.
  - Manage internally the physical location where the metadata information is actually stored.
  - Provide high availability of metadata information by automatically distributing replicas, ensuring the consistency among them.
  - Provide fault tolerance mechanisms by ensuring access to the system regardless of whether any node of the distributed repository is temporarily unavailable.

Component dependencies

For this component to communicate to the other components that it might interact with, it will send its requests and updates to the *Information Directory Manager*; any messages that the component needs to receive, it will get them through the *Information Directory Manager*.

4.1.6 Data and Metadata Management Implementations

The implementations of the *Ontology repository* component here considered are those specialized in semantic web resources (i.e., ontologies, RDF schemas, etc.). They can be classified in 2 different types: centralized or decentralized.

We considered implementations of the *Data repository* component as any collection of digital data that is available to one or more entities (e.g. users, systems) for a variety of purposes (e.g. learning, administrative processes, research, etc.) and that has the characteristics proposed by Heery and Anderson [26]. Note that over time, data repositories have been referred by other names (e.g. knowledge base, data library, digital library, data warehouse) depending on its contents, purpose or capabilities.

Not so many implementations were found for the *Alignment repository* component since this area is quite new in the research environment. Several other components of this kind are under development but they have not yet been made publicly available. Besides, we can easily see that the tools that were identified also incorporate an alignment tool, which shows again that this area is very young and further refinements are expected.

Finally, even though we could find many *Metadata registry* implementations of general purpose, we only considered those that are either specialized for ontologies or that can store ontology metadata in some way.

4.1.7 Existing Implementations

The table below shows the implementations found for these dimensions[3]:

Component	#	Implementations
Information directory manager	6	Aduna Metadata Server, Alvis, Beagle++, Gnowsis, Haystack, OWLIM
Ontology repository	15	KAON2, Jena, Sesame, Ontology Server, RDF Server, Knowledge zone, Onthology, OntoSelect, DAML Ontology Library, SchemaWeb, ONTOSEARCH2, Protégé Ontologies Library, OntStore, RDFPeer, RDF2GO
Data repository	3	DSpace, Lucene, Zebra
Alignment repository	2	COMA++, Alignment API and Alignment Server
Metadata registry	17	3store, AllegroGraph, Boca, Brahms, Hawk, The open metadata registry (prototypes 1-3), OASIS ebXML Registry, Oyster, Oyster2, Kowari, RDFGateway, RDF2GO, RDFStore, SemWeb, YARS

4.2 Querying and Reasoning

This dimension includes those components that provide functionalities to generate and process queries.

[3] Note that not all the implementations found for a certain component cover all the functionalities described for the component. Identifying the degree of coverage of an implementation for a component is out of the scope of this deliverable.

The basic dependencies of the components in this dimension are shown in Figure 6:

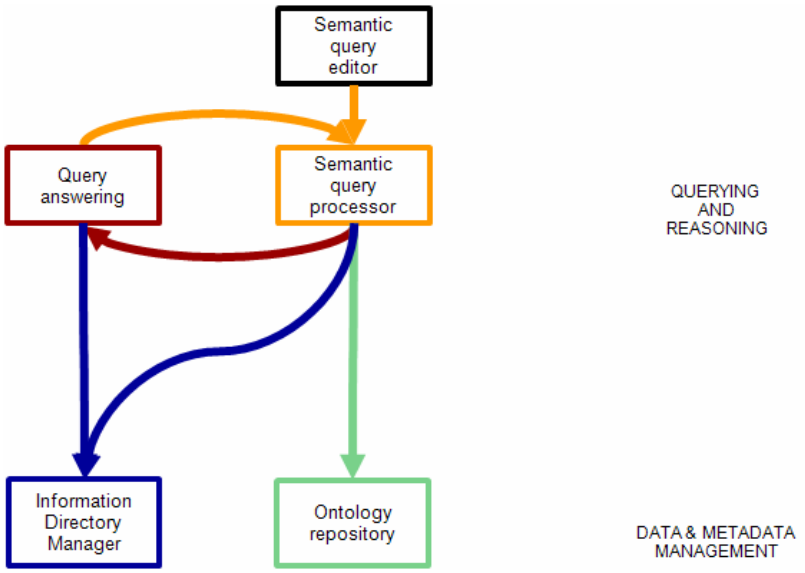

Figure 6. Dependencies of the components of the *Querying and reasoning* dimension.

4.2.1 Query answering component

The *Query answering* component takes care of all the issues related to the logical processing of a query by providing reasoning functionalities to search results from a knowledge base [15].

Functionalities provided

The *Query answering* component provides the following main functionalities:

- Query consistency functionalities, which are in charge of checking whether a query is satisfiable [15]. In particular, the ontology consistency functionalities will be able to check whether the whole knowledge base is satisfiable.
- Query containment functionalities, which check whether the answer to a query is a subset of the answer to another query by using the subsumption functionalities, which check whether a concept is more general than another [15].
- Instance checking functionalities, which check whether an individual is an instance of a particular concept (cf. [15], pg. 67).
- Query rewriting functionalities. The answer to a query can be a subset (*view*) of the knowledge base. The scope of a query can be not the whole knowledge base but just a view of it. Such a query can be seen as the cascade (*intersection*) of two queries, the first of which being the one that retrieves the suitable view of the knowledge base and the second, the proper query.

It can also provide the following functionalities:

- Checking if a query pattern matches an ontology. The *Query answering* component returns a boolean value, which is true if the query pattern matches an ontology schema.

- Selecting ontology concepts that satisfy query constraints. The *Query answering* component returns zero or more bindings of variables contained in the query. Each set of bindings satisfies the query.
- Extracting ontology parts relevant to the particular query. The *Query answering* component recreates and returns an ontology part (sub ontology) that matches the query.
- Describing ontology concepts. The *Query answering* component returns all the information about the given concept available within an ontology, including its connections to the other ontology concepts.
- Restricting result number. The *Query answering* component allows the specification of an upper bound on the number of query results returned.

Component dependencies

This component uses

- The *Semantic query processor* component through an application interface.
- The *Information Directory Manager* component as a mediator for accessing various components, the *Ontology repository* component to access to knowledge bases or the *Data repository* component to access instances.

4.2.2 Semantic query processor component

The *Semantic query processor* component takes care of all issues related to the physical processing of a query, by providing functionalities to manage query answering over ontologies in distributed sources. This involves (among other functions) translating queries and their results from one ontology to another. The *Semantic query processor* will have, as input, the results from the *Query answering* component. It will also merge results from different information sources into a consistent unified result which can be presented to the end user.

Functionalities provided

Other additional functionalities provided by this component are

- Identifying sources that contain information relevant to the query.
- Requesting information from the identified sources.

Component dependencies

This component uses

- The *Query answering* component in order to translate queries to the ontologies used by distributed sources through an application interface. It uses this component through an application interface in order to initiate query execution in remote sources and to obtain query results.
- The *Ontology repository* and the *Information Directory Manager* components to answer queries over ontologies in distributed sources and/or distributed information sources.

4.2.3 Semantic query editor component

The *Semantic query editor* component takes care of all issues related with the interface with the user, by supporting a user in formulating a query.

Functionalities provided

- Supports the user in formulating a query.
- Provides a user-friendly query language.
- Provides a user-friendly representation of results.

Component dependencies

This component uses

- The *Semantic query processor* component to translate queries and their results from the user-friendly format to the others and back again.

4.2.4 Querying and reasoning implementations

The most important conclusion that can be drawn for this dimension is that the components that we have identified here are not always easily outlined and delimited in real life implementations. Even so, we still feel that the identified components are correct, since the logical procedure in answering a query matches the identified components.

4.2.5 Existing Implementations

Next table shows the implementations found for these dimensions:

Component	**#**	**Implementations**
Query answering	6	AJAX Client for SPARQL, Bor, Corese, KAONP2P, KAONWeb, Oyster2
Semantic query processor	2	AeroText, Sesame
Semantic query editor	2	Ontogator, SemSearch

4.3 Ontology Engineering

This dimension includes those components that provide functionalities to develop and manage ontologies.
The basic dependencies of the components in this dimension are shown in Figure 7:

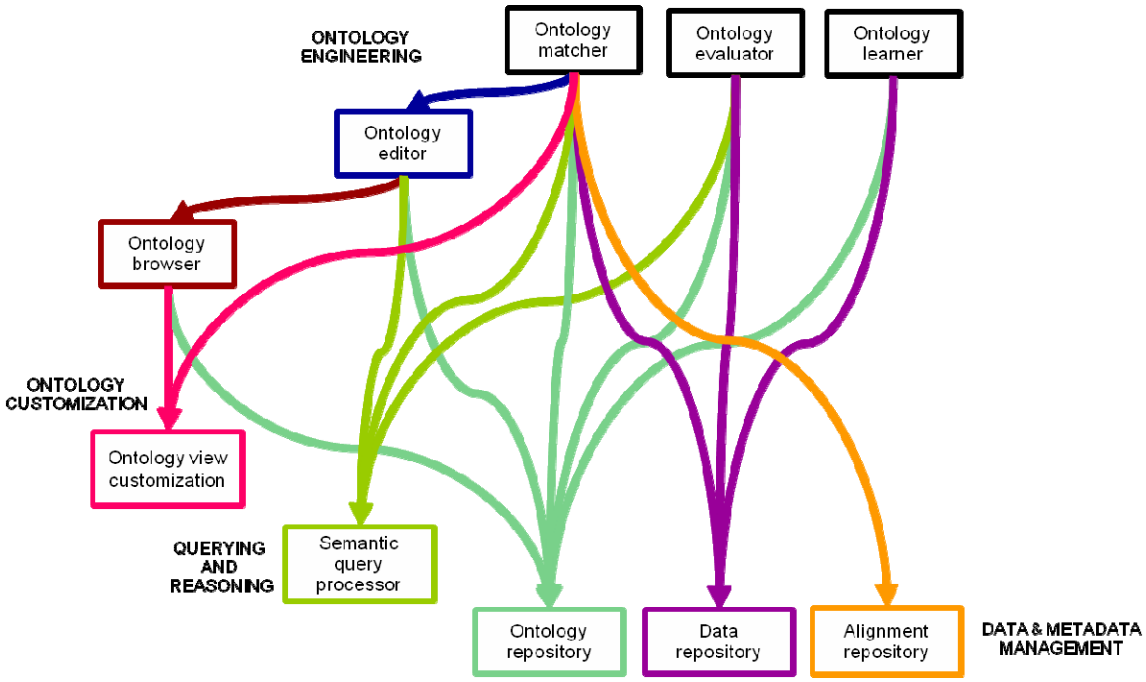

Figure 7. Dependencies of the components in the *Ontology engineering* dimension.

4.3.1 Ontology editor component

The *Ontology editor* component provides functionalities to create and modify ontologies, ontology elements, and ontology documentation. These functionalities include a single element edition or a more advanced edition such as ontology pruning, extension or specialization.

The *Ontology editor* component does not restrict the ontology edition to a specific knowledge representation formalism or format, the restriction depends on the implementation of the component.

Functionalities provided

- Ontology edition functionalities, to create and modify ontologies and ontology elements. These functionalities are provided through a user interface.
- Ontology pruning functionalities, to remove elements from an ontology which are no relevant to a given application domain [15]. These functionalities are provided through a user interface.
- Ontology extension functionalities, to broaden the covered domain of an ontology by extending its elements. These functionalities are provided through a user interface.
- Ontology specialization functionalities, to specialize ontology elements for a particular domain [16]. These functionalities are provided through a user interface.
- Ontology documentation functionalities, to document ontologies and ontology elements. These functionalities are provided through a user interface.
- Ontology browsing functionalities, to visually browse an ontology. While the *Ontology browser* component also provides these functionalities, they are an integral

part of the *Ontology editor* component. These functionalities are provided through a user interface.

Component dependencies

This component uses

- The *Ontology repository* component to access and store ontologies and ontology elements through a programming interface.
- Optionally, the *Semantic query processor* component to check if an ontology is satisfiable after performing changes through a programming interface.
- Optionally, the *Ontology browser* component to navigate through an ontology, to insert, modify or document its elements, either through a user interface or through a programming interface.

4.3.2 Ontology browser component

The *Ontology browser* component provides functionalities to visually browse an ontology.

Functionalities provided

- Visual browsing functionalities to visually browse an ontology. These functionalities are provided through a user interface.

Component dependencies

This component uses

- The *Ontology repository* component to access ontologies through a programming interface.
- Optionally, the *Ontology view customization* component to visualize the ontology to be browsed through a programming interface.

4.3.3 Ontology evaluator component

The *Ontology evaluator* component provides functionalities to evaluate ontologies, either their formal model or their content, in the different phases of the ontology life cycle.

Functionalities provided

- Ontology evaluation functionalities, to make a technical judgment of the ontologies, their associated software environments and documentation with regard to a frame of reference during each phase and between phases of their life cycle [17]. These functionalities can be provided through a user interface or a programming interface.
- Ontology verification functionalities, to ensure that the ontology implements correctly the ontology requirements and competency questions, or that functions correctly in the real world [18]. These functionalities can be provided through a user interface or a programming interface.
- Ontology validation functionalities, to prove that the world model (if it exists and is known) is compliant with the world modelled formally in the ontology [18]. These functionalities can be provided through a user interface or a programming interface.
- Ontology assessment functionalities, to judge the understanding, usability, usefulness, abstraction, quality and portability of the ontology from the user's point of view [18].

These functionalities can be provided through a user interface or a programming interface.
- Ontology diagnosis functionalities, to identify the causes of errors in an ontology. These functionalities can be provided through a user interface or a programming interface.

Component dependencies

This component uses
- The *Semantic query processor* component using subsumption functionalities to decide whether a concept is more general than another, classification functionalities to build concept hierarchies, and ontology consistency functionalities to check if an ontology is satisfiable, through a programming interface.
- The *Ontology repository* component to access ontologies through a programming interface.
- Optionally, the *Data repository* component to access other sources such as linguistic resources to help in the ontology evaluation through a programming interface.

4.3.4 Ontology learner component

The *Ontology learner* component provides functionalities to acquire knowledge and generate ontologies of a given domain through some kind of (semi)-automatic process.

Functionalities provided

- Ontology learning functionalities, to derive ontologies (semi)-automatically from natural language texts as well as semi-structured sources and databases by means of machine learning and natural language analysis techniques [15]. These functionalities can be provided through a user interface or a programming interface.

Component dependencies

This component uses
- The *Ontology repository* component to access ontologies through a programming interface.
- The *Data repository* component to access data sources through a programming interface.

4.3.5 Ontology matcher component

The *Ontology matcher* component provides functionalities to match two ontologies and output some alignments. We can distinguish two main types of such systems: those that provide only matching and those that directly use matching for processing another task (merging, mediating, etc.).

Among the other types of differentiation are the requirement that a user drive the system and the type of input required by the system.

Functionalities provided

The functions provided by theses systems, which here are called matchers, can be numerous:

- Matcher: The *ontology matcher* will provide an alignment (set of correspondences) from a list of two or more ontologies.
- Server: The server stores and loads alignments from persistent repositories such as a file system, a database or a distributed repository.
- API: An API provides alignment manipulation such as trimming or format change.
- Editor: An editor provides the graphic representation and manipulation of the alignments.
- Transformer: From one ontology written in a particular ontology language, the *Ontology transformer* component generates an ontology in another language
- Merger: From an alignment between two ontologies to generate a new ontology that contains the entities of both ontologies as well as the relationships between these entities.
- Data translator: A data translator performs the translation of data according to an alignment between a source ontology or data source with regard to which the data is expressed and a target ontology to which it is translated.
- Mediator generation: A mediator generator generates a program able to transform queries from one ontology or a data source to another ontology according to an alignment between them and to transform the answers to the query with regard to the same alignment.

So, we have recasted the modules of deliverable 1.2.4 into functions because they are often tied to matchers.

Component dependencies

The dependencies of this component are mostly those of the Ontology API they use. They are also sometimes related to some repository and reasoning mechanism (they relate to some Ontology API). We mention them when they are applied in the description of the systems.

The dependencies with components that are now identified as functions are presented under that heading in the following.

4.3.6 Ontology engineering implementations

Ontology Editor

The *Ontology editor* component implementations (ontology editors from now on) can be classified into 2 different types. The most common one is that of applications whose main goal is ontology edition and the least frequent are ontology edition plugins of larger applications.

Ontology editors dealing with one specific ontology have not been considered.

Some ontology editors provide other ontology engineering functionalities besides pure ontology edition.

All the ontology editors are standalone or web applications that are accessed through user interfaces. They access other component implementations using programming interfaces.

Most of the ontology editors manage RDF(S), OWL, or both for knowledge representation.

Although different ontology editors use different *Ontology repository* implementations, most of them are able to access ontologies stored in a local file system and, in some cases, remote ontologies stored in a web server.

Not all the implementations benefit from the reasoning capabilities of *Semantic query processor* implementations. And, on the other hand, *Ontology browser* implementations are only used by one implementation (Protégé). The rest of the implementations provide themselves ontology browsing functionalities.

Ontology Browser

The *Ontology brows*er component implementations (ontology browsers from now on) can be classified into 3 different types: applications whose main goal is ontology browsing, ontology browsing plugins of larger applications, and ontology development tools that provide ontology browsing functionalities. In this section we do not consider implementations of ontology development tools, as they are included in the implementations of the *Ontology editor* component. Furthermore, ontology browsers that only deal with one specific ontology have not been considered.

All the ontology browsers are standalone or web applications that are accessed through user interfaces; they have ontology browsing as their only functionality, and access other component implementations using programming interfaces.

Ontology browsers manage RDF(S), OWL, or both. Except in the case of the Protégé plugins, which use Protégé as *Ontology repository* implementation, the rest of the ontology browsers only access ontologies stored in files that are located on local file systems, web servers, or both.

Ontology browsers do not use *Ontology view customization* components.

Ontology Evaluator

The *Ontology evaluator* component implementations (ontology evaluators from now on) are either applications (standalone or web) or program libraries. In the case of program libraries, usually one small application has been developed using the program library to allow users to evaluate ontologies.

The implementations of the *Semantic query processor* component can be used to evaluate ontologies using their subsumption, classification and consistency checking functionalities. In this section we do not consider these implementations, as they are included in the implementations of the *Semantic query processor* component.

All the ontology evaluators access other component implementations using programming interfaces.

All the ontology evaluators manage RDF(S), OWL, or both for knowledge representation.

Although different ontology evaluators use different *Ontology repository* implementations most of them are able to access ontologies stored in a local file system and, in some cases, remote ontologies stored in a web server.

The implementations do not benefit from the reasoning capabilities of *Semantic query processo*r implementations, and *Data repository* implementations are only used by one implementation (CLEANONTO).

Ontology Learner

The *Ontology learner* component implementations (ontology learners from now on) are either standalone applications or program libraries. Some of these ontology learners are part of ontology engineering environments that provide other functionalities.

All the ontology learners access other component implementations using programming interfaces.

Most of the ontology learners manage RDF(S), OWL, or both for knowledge representation.

Although different ontology editors use different *Ontology repository* implementations most of them are able to access ontologies stored in a local file system and, in some cases, remote ontologies stored in a web server. All the implementations use local file systems as *Data repository* implementations.

Ontology Matcher

The *Ontology matcher* component implementations can be classified into 3 different types: applications whose main goal is ontology browsing, ontology browsing plugins of larger applications, and ontology development tools that provide ontology browsing functionalities.

There are many ontology matchers available. A regularly updated list of such systems can be found at http://www.ontologymatching.org/projects.html.

A technical description of most of the systems below has been provided in Deliverable 1.2.2.2.1. A more systematic analysis of such systems has been provided in [25].

According to this last reference, we distinguish Ontology matchers, which are algorithms for matching, from Alignment framework, which supports the whole alignment lifecycle.

It can be observed that there are many different tools that can provide ontology matching. As far interoperability of these tools is concerned, they are very often tied to a particular implementation of ontology API or editor, and they deliver heterogeneous formats.

Therefore, communication is difficult and often has to go through serialisation of the alignments, i.e., printing and parsing.

Using a framework ensures the availability of many different functions under the same hood, which avoids these printing and parsing cycles. These frameworks also often have many different ways of interacting with them providing easier integration.

Some of these frameworks have sheltered various tools made by third parties (Alignment API, Prompt). This is often a guarantee of openness and easier customisation. It is thus largely advisable to use such tools unless specific requirements are at play.

4.3.7 Existing Implementations

The table below shows the implementations found for these dimensions:

Component	#	Implementations
Ontology editor	23	Altova Semanticworks, DOODLE (Domain ontology rapid development environment), DOE, DOME, Fenfire, Graphl, GrOWL, IBM Integrated Ontology Development Toolkit, Infered, IsaViz, KAON OI Modeler, Linkfactory, Ontotrack, Powl, Protégé, Rhodonite, SemTalk, SWOOP, Topbraid composer, WebODE, DogmaModeler, ICOM
Ontology browser	15	Brownsauce, BrowseRDF, Drve RDF Browser, Disco, Horus, Longwell, OINK, RDF Gravity, Tabulator, Welkin, Jambalaya, Ontosphere 3D, OntoViz, OWLViz, TGVizTab
Ontology evaluator	7	ARP: Another RDF Parser, CLEANONTO, ConsVISor, Eyeball, ODEVal, OWL API, Semantic Web RDF Library for C#/.NET
Ontology learner	4	DOODLE (Domain ontology rapid development environment), KEA (Keyphrases Extraction Algorithm), Text2Onto, TERMINAE
Ontology matcher	29	AMV, AUTOMS, CMS, CtxMatch, eTuner/iMap/Glue/LSD, Falcon-AO, NOM, QOM, APFEL, H-Match, LOM, MapOnto, MetaQuerier, MoA, OLA, S-Match, SAMBO, Similarity Flooding, ToMAS/Clio, OntoBuilder, OntoMerge, Aligment API & Aligment server, COMA & COMA++, FOAM, PROMPT, Rondo, Chimaera, MAFRA, Mapping Discovery

4.4 Ontology Customization

This dimension includes the components that provide functionalities to customize and tailor ontologies. In general, the ontology customization and personalization tasks attempt to address the problems that arise as ontologies become larger and more complex. In principle, we distinguish two types of customization for the purposes of this deliverable:

- The *customization of the view on an ontology*, e.g. when exploring a network of ontologies. This customization is more or less ad-hoc and the results of the customization may be discarded once the user proceeds with exploring the ontology. This customization when exploring an ontology tries to reduce the complexity of an ontology and only shows parts which are relevant to the current user.
- The customization for the purposes of reusing ontologies and integrating them into a network with other ontologies according to specific needs (e.g. during the ontology deployment, reasoning or design phases). Here the results of the customization will often be integrated into the edited ontology; and so the nature of these results is more persistent.

The basic dependencies of the components in this dimension are shown in Figure 8:

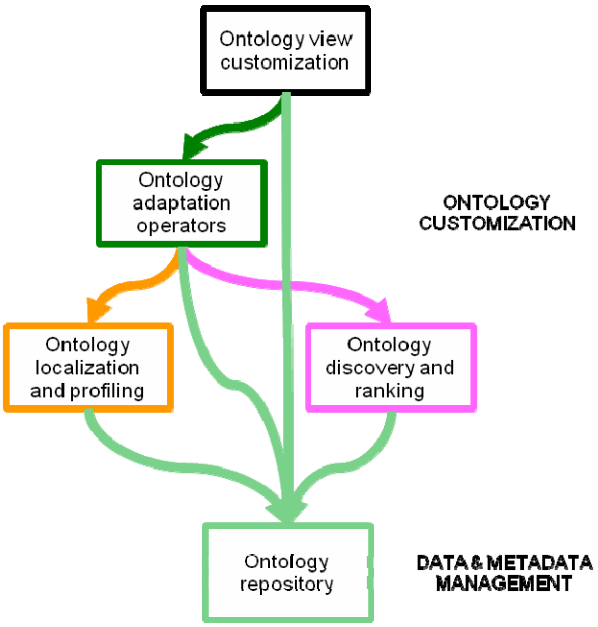

Figure 8. Dependencies of the components in the *Ontology customization* dimension.

4.4.1 Ontology localization and profiling component

This component is in charge of providing functionalities to adapt an ontology according to some context (e.g. communal or individual preferences, language, expertise, etc.) or some user profile.

Customizing/personalizing ontologies via adjusting the views on them based on the user profiles can be seen as bringing an ontology in the context of a particular user. Consequently, user profiles and preferences can be seen as contextual modifiers. There are several ways of constructing and using a user profile for this purpose. User profiles are seen here as a mode of describing some user properties or characteristics and thus, as a representation of the context of a user. Such a profile may, for example, provide information about the role of a user, the domain of interest or the current task.

When talking about the user, it is important to mention that we can decide to have an *abstract user* – this would, in principle, correspond to any member of a group of users in a particular situation. The same user can belong to different groups depending on the task at hand. A user profile can be constructed in different ways depending on the data it includes and the methods used for its construction: *manual*, *semi-automated* and *fully automated*. While manual methods have no problem with providing benefit to the new users, automatic methods can only make guesses. Hence, automated methods may sometimes provide very limited or no benefit to the new user, while they are waiting to collect enough data for automatically constructing a reasonable user profile. On the other hand, one of the important dimensions of the user profile is also its adaptation to the changes over time, which may be important to some applications (e.g., recommending clothes or movies to the user). The adaptation requires updating of the user profile, which is easier if we have automatic methods for profiling.

Functionalities provided

- Manual construction and activation of user profiles
    - Acquisition of preferences from the user
    - Acquisition of information about the user (interests, demographic, …)
- Semi-automated construction and adaptation of user profiles
    - Extension of profiling dimensions (for subsequent manual acquisition)
    - Addition of new profiling dimensions (for manual acquisition)
    - Monitoring of the user alongside defined profiling dimensions
    - Machine learning with subsequent user feedback
- Automated acquisition and management of user profiles
    - Case-based profile learning and adaptation
    - Statistical profile learning and adaptation
    - Explicit and/or implicit feedback facilities

User profiles can be used in the analysis of the users, providing some insights into the population that use the system, but more significantly, they may be used to change some action/interface of the system; i.e. they provide us the opportunity to influence user's further actions based on his or her current profile. An example of how user profiling may be used in the context of ontological engineering can be found in [19]. The user profile is then constructed in interaction with the user by grouping documents into a hierarchy based on their content similarity. The approach is based on the previously proposed idea of capturing the interest of the user in a topic hierarchy automatically constructed from the web documents visited by the user. The same idea was also used in an automatic user profiling to enable the interest-focused visualization of the ontologies [20].

Component dependencies

This component uses

- The *Ontology repository* component to access ontologies through an application interface.

4.4.2 Ontology discovery and ranking component

This component is in charge of providing functionalities to find appropriate views, versions or sub-sets of ontologies, and then to rank them according to some criterion.

It is a widely cited fact that the number of ontologies and semantically marked up data is growing at a rapid pace. However, the current knowledge of the quality of the content in the distributed Semantic Web resources is very sparse. A component is therefore needed to facilitate and enable advanced Semantic Web applications to *access* and *use* ontologies that may be distributed throughout the Web, and to *enrich* access to distributed ontologies by taking into account their quality, the relationships and the interdependencies.

Functionalities provided

Three **broad functionalities** can be identified for the following purposes:

- Crawling the Web (or another repository) for ontological content
  - Executing and managing generic data crawling through a network
  - Syntactic filtering of ontological and semantic content
  - Caching and storing crawled content
- Validating crawled ontologies and calculating a range of quality measures
  - Managing syntactic dependencies to ensure consistency
  - Charting topological relationships between and within ontologies
  - Inferring other semantic relationships within ontologies
  - Establishing semantic networked relationships among ontologies
  - Indexing discovered ontologies for future use
- Supporting queries for ontologies, ontological entities, relationships, etc.
  - User-level (human-centric) queries (e.g. web forms)
  - Machine-level (content-centric) queries (e.g. SPARQL)
  - Keyword/term level queries
  - Concept and/or ontology URI-s level query
  - Exploratory navigation through the discovered ontologies

Component dependencies

This component is not dependent on other prerequisites; however, the three functionalities mentioned above are dependent on each other. In other words, querying relies on some information being recorded during the validation stage, and validation, in turn, relies on ontologies or other semantic content being available.

Components that may depend on the ontology discovery include e.g. feeds to the ontology registry, to ontology metadata schemas, etc.

This component uses functionalities from the *Ontology repository* component to access ontologies, through an application interface.

4.4.3 Ontology adaptation operators component

This component is in charge of applying appropriate operators to the ontology in question, the result of which is an ontology customized according to some criterion (e.g. levels of trust or group preferences).

One way in which (usually) large ontologies could be customized for different purposes is that of splitting them into a network of smaller sub-ontologies or modules. Arranging and relating these modules to each other represents already a form of customization.

These techniques typically involve some selection and filtering; e.g. removing parts which are not relevant for the user.

A modularized approach to create ontologies such as this one would facilitate ontology reuse and would also help to breakdown ontologies into smaller, manageable pieces. This would benefit subsequent functional components of the architectural framework. For the purpose of this section, we would emphasize the SELECT operation. There may exist several selection operators that help to select those parts of an ontology or those modules that are of interest for the user.

Functionalities provided

- Selecting an ontology (sometimes referred to as module selection).
- Summarizing an ontology or its parts (or sometimes ontology collapsing).
- Making a glossary from (or flattening) an ontology into a list of terms.
- Filtering a sub-set of concepts satisfying a given condition/criterion.
- Extracting a sub-set of concepts and construction of a coherent and consistent module/ontology.

Note that the above functionalities are focused largely on working with one ontology and on making some amendments to it. There is obviously the possibility of taking a complementary view whereby one would work with a set or network of ontologies and attempt to carry out operations such as *merge*, *match*, *assemble*, *compute intersect* or *compute union*. However, these adaptation operations are more applicable in the context of networked ontologies and will not be elaborated here.

Component dependencies

In terms of dependencies, the operators in this component may rely on

- The *Ontology profiling and localization* component, and
- The Ontology *discovery and ranking* component, to a limited extent.
- The *Ontology repository* component to access ontologies through an application interface.

4.4.4 Ontology view customization component

This component is responsible for enabling the user to change or amend a view on a particular ontology to fit a particular purpose (e.g. preview, content-based view, topography, etc.).

One of the key characteristics of the Semantic Web is the emphasis on separating the content from its presentation. Semantic Web languages are in principle presentation-free. Like the Web, when using technologies as for example XHTML+CSS or even XML+XSLT, content is separated from its presentation. This has a major advantage - information and knowledge become comprehensible to computers and artificial agents. However, it also has a major disadvantage - any semantically annotated chunk of formal

knowledge needs to be 'transformed' into a form or shape comprehensible by a human user.

First, datasets typical of the Semantic Web are relatively large; the challenge of large open information spaces is to provide a simple, yet effective way of finding, sorting and viewing relevant information. Second, ontologies that conceptually underpin some Semantic Web applications could be complex structures representing different types of relationships. If each of such potential relations is treated as a dimension in which allowed values could be depicted, then even a moderately complex ontology leads to a multi-dimensional space, which poses challenges for navigation and interaction.

Functionalities provided

Functionally, we can distinguish between

- Handling large and/or complex conceptual spaces using *reduction strategies* – these are typically concerned with showing *less* information (in our case, fewer concepts, entities or relationships), and
- Handling complexity and/or size by an *appropriate projection* – which tackles the same issue by showing the same set of concepts, entities and relations *differently*.

Subsequently, we can identify the following sub-components contributing to the ontology view adaptation:

- Faceted browsing techniques (e.g. Longwell, \facet, etc.)
- Spatial navigation and representations (w.g. VIKI, mSpace, etc.)
- Focus-context projections based on semantic relationships/properties (such as FishEye or CropCircles)
- Ontology content depiction, e.g. in terms of layering an ontology over a statistically constructed landscape of a particular corpus/domain

Component dependencies

In terms of dependencies, the parts of this component rely to some extent on

- The *Ontology adaptation operators* component (in particular summarization, glossarization and filtering).
- The *Ontology localization and profiling* component.
- The *Ontology repository* component to access ontologies, through an application interface.

4.4.5 Ontology customization implementations

In principle we distinguished three types of customization tools for the purposes of reviewing implementations:

- Tools focusing on *gathering and mining information* that can later be used for ontology customization, adaptation, and view construction, but also for ontology extension, learning, etc.

- Tools accomplishing *customization of the view on an ontology*, mostly the view on ontological instances and the relationships among them. This customization is more or less ad-hoc and the results of the customization may be discarded once the user proceeds with exploring the ontology. This customization, when exploring an ontology, tries to reduce the complexity of an ontology and only shows parts which are relevant to the current user.
- The customization for the purposes of *reusing ontologies and integrating them* into a network with other ontologies according to specific needs (e.g. during the ontology deployment, reasoning or design phases). Here the results of the customization will often be integrated into the edited ontology and their nature is more persistent.

Of the three groups, the best progress is visible in the first one, followed by the second. Especially, in recent years, the second group has been extended with a number of high-profile applications working with real datasets. Yet, most tools in this second group focus on customizing view of the datasets, less so on ontologies.

4.4.6 Existing Implementations

The table below shows the implementations found for these dimensions.

Component	#	Implementations
Ontology localization and profiling	4	Ontogen, Calendar Apprentice, Personal WebWatcher, Document Atlas
Ontology discovery and ranking	3	Watson, Swoogle
Ontology adaptation operators	4	ONION, PROMPT, Chimaera, FONTE
Ontology view customisation	14	Longwell, TGVizTab, OntoViz, Jambalaya, OWLViz, /facet, mSpace, VIKI, CropCircles, CS Aktive Space, SpaceTree, TreeMap, Spotlight, IsaViz

4.5 Ontology Evolution

This dimension includes those components that provide functionalities to manage the ontology evolution.

The basic dependencies of the components in this dimension are shown in Figure 9:

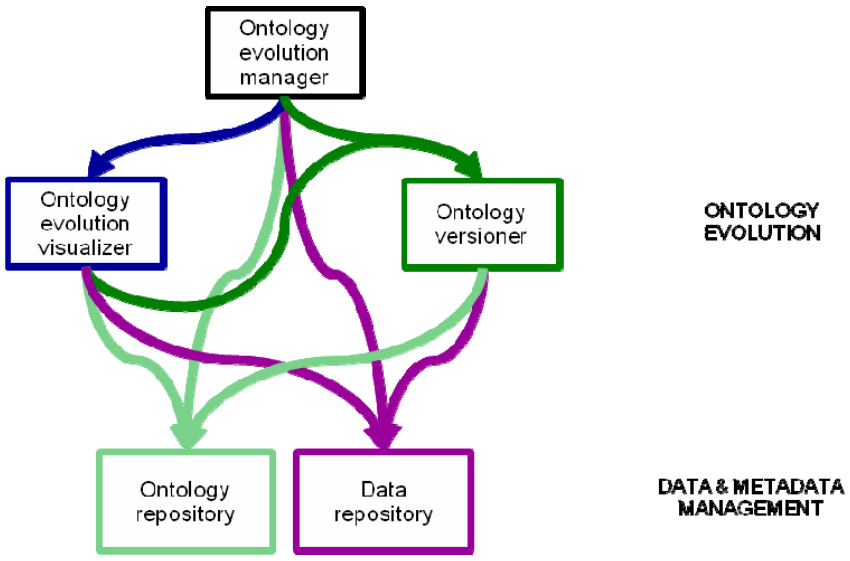

Figure 9. Dependencies of the components in the *Ontology evolution* dimension.

4.5.1 Ontology versioner component

The *Ontology versioner* component allows maintaining, storing and managing different versions of an ontology.

Functionalities provided

- The *Ontology versioner* component performs changes to ontologies.

Component dependencies

This component uses

- The *Ontology repository* component for manipulating the ontologies.
- The *Data repository* component for manipulating the data.

4.5.2 Ontology evolution visualizer component

The *Ontology evolution visualizer* component allows visualizing different versions of an ontology. The dependencies of the *ontology evolution visualizer* with other components of the SWF are shown in Figure 9.

Functionalities provided

- The component will visualize the evolution of ontologies.

Component dependencies

This component uses

- The *Ontology versioner* component for visualizing different versions of ontologies.
- The *Ontology repository* component for querying the ontologies.
- The *Data repository* component for querying the data.

4.5.3 Ontology evolution manager component

The *Ontology evolution manager* component allows maintaining, storing and managing different versions of an ontology, and visualizing as well as possible the versions within a broader context of complex ontology evolution and development platform. The dependencies of the *ontology evolution manager* with other components of the SWF are shown in Figure 9.

Functionalities provided

- The component will manage the adaptation of an ontology regarding changes that may arise.

Component dependencies

This component uses

- The *Ontology repository* component for manipulating the ontologies.
- The *Data repository* component for manipulating the data.
- Other evolution components such as the *Evolution visualizer* and the *Ontology versioner* wrapping the functionalities of this two lower-level components.

4.5.4 Ontology evolution implementations

Ontology versioner

The *Ontology versioner* component implementations usually present a library that implements the essential functions for storing ontology versions, difference computation (either syntactic or semantic), querying of multiple versions, change management (e.g. user commits, check-outs, branching, etc.). Then other components of the evolution dimension may build on these libraries.

Conclusions

- Implementation of the operations underlying the change management of ontologies.
- Ontology versioning functionalities provided via API.

Ontology evolution visualizer

The *Ontology evolution visualizer* component implementations usually present a user interface that allows browsing an ontology in the context of its multiple versions, compare visually different ontologies, and might also perform some versioning operations within the visual interface (e.g. merging of branches). Other components of the evolution dimension may incorporate this interface.

Conclusions

- Implementation of the visualization of different versions of ontologies.
- The visualization and possible version management functionalities provided via user interface.

Ontology evolution manager

The *Ontology evolution manager* component implementations are usually incorporated into complex ontology development and evolution framework, providing either APIs or user (web of standalone) interfaces.

Conclusions

- Implementation of the ontology evolution within a broader ontology development platform.
- Lower-level ontology versioning functionalities realised via specific implementations of other evolution components (e.g. *ontology versioner*) can, however, be often built-in into the particular platforms (based for instance on database transactional model or pure syntactic versioning similar to CVS principles).

4.5.5 Existing Implementations

The table below shows the implementations found for these dimensions:

Component	#	Implementations
Ontology versioner	2	SemVersion, DIP ontology versioning
Ontology evolution visualizer	3	SemVersion Protégé plug-in, PROMPT, PROMPTDiff
Ontology evolution manager	5	KAON, DOME, MarcOnt Portal, Linkfactory, Powl

4.6 Ontology Instance Generation

This dimension includes four components that provide functionalities to generate ontology instances.

The basic dependencies of the components in this dimension are shown in Figure 10:

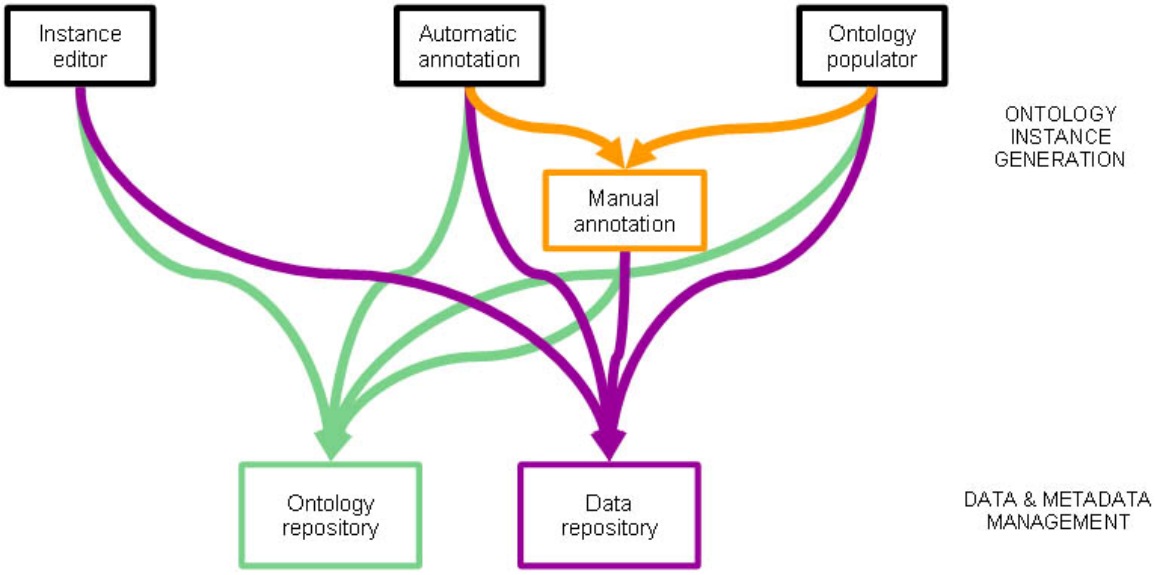

Figure 10. Dependencies of the components in the *Ontology instance generation* dimension.

4.6.1 Instance editor component

The *Instance editor* component is in charge of providing functionalities to manually create and modify instances of concepts and of relations between them in existing ontologies.

Functionalities provided

This component provides a GUI that enables the user to manually add, delete and modify instances of concepts in an existing ontology, and also properties of and relations between the instances. Adding a new instance is a fairly simple process. However, when an instance is modified or deleted, this may affect other concepts in the ontology. For example, an instance may belong to more than one concept in the ontology, so it may need to be removed or modified in several places. The same holds true for properties. The *instance editor* component provides some of the same functionalities as the *ontology editor* component described in Section 4.1, but the latter have added a functionality which may not be present in the simpler *instance editor* component. The *instance editor* component is a simple tool which may be easier to integrate with the other ontology instance generation tools described in this section.

Component dependencies

This component uses

- The *Ontology repository* component to insert ontology information.
- The *Data repository* component to obtain information from the content sources.

4.6.2 Manual annotation component

The *Manual annotation* component is in charge of manual and semi-automatic annotation of digital content documents (e.g. web pages) with concepts in the ontology. With respect to textual data, mentions of instances in the text which correspond to concepts in the ontology are annotated manually in the document. Similarly, for non-textual data (e.g., visual, audio and audiovisual sources), the concepts in the ontology, reflecting the meaning conveyed, are associated with the media content item. This annotation process may be assisted or guided by a machine (semi-automatic annotation).

Functionalities provided

The manual annotation of text must provide a user-friendly GUI which enables users to view the ontology and text and annotate mentions of instances in the text with classes from the ontology (for example, to annotate "John Smith" with the concept "Person"). Note that if there are several occurrences of "John Smith" in the text, which all refer to a person, then they are all annotated identically. Compare this with the *Ontology populator* component, which will only create one instance in the ontology for "John Smith" no matter how many times it is mentioned in the text (given that it is referring to the same person). The aforementioned also holds for the case of non-textual data annotation, while additionally appropriate functionalities need to be provided so that the user can select the specific media parts that he wants to annotate. For example, when annotating a video, a user may want to associate a concept in the ontology with a sequence of frames, a single frame, or some 2D spatially defined region within a frame.

The *Manual annotation* component requires an ontology to be selected for annotation, and a content item or set of content items to be loaded. Ideally, for the content to be annotated, a variety of formats should be supported such as plain text, XML, HTML, PDF, MPEG-2, JPG, TIFF, etc. The annotator should enable, at the least, annotation with class information from the ontology, and preferably annotation with instance information and properties.

There are a number of desirable (though not essential) features that an annotator component should contain:

Multiple ontologies: the component should enable the user to select multiple ontologies to work with simultaneously. This could be via a pull-down menu of ontologies available to the system. With respect to non-textual content annotation, another useful property refers to providing support for using not only domain-specific ontologies, but also ontologies for describing the structure and decomposition of content, so that the interoperability of annotations is further enhanced.

Ontology evolution: the GUI ideally needs to cater for changes in the ontology – provide a migration support, where content already annotated with concepts and instances that have changed can automatically be re-annotated or, if this is not possible, provide a way for the user to specify the mappings from old to new concepts/properties or even correct manually each one of the occurrences.

Collaborative annotation: in order to reduce the burden of annotation, the component can allow several users to annotate content from the same collection simultaneously. The collaboration can take place at two different levels: first, when the same ontology is shared among several users and needs to be edited to populate instances in it, and second, when a same document is annotated by several annotators at the same time. In order to address the first scenario, having a common ontology repository is important. This provides a way to set up a common ontology repository from which users can load an ontology and save it back. The second scenario requires the use of Web services such as an Annotator GUI. Each user who participates in the annotation process is allocated a separate annotation set where all the annotations they create are stored. The document can then be merged once all annotators have completed their task.

Searching for similar annotated content: the user might be unsure of how something needs to be annotated or has forgotten whether he/she has already annotated such content before. One interesting feature is to allow searching for similar content. This can be done using IR-based document similarity or by measuring how much overlap of annotation exists between the documents [15].

Machine-assisted annotation: in order to gain faster a large number of instances with minimal manual annotation effort, an interface that collects previous occurrences of annotations and suggests them to the user is required. Once a user has annotated, for example, some string or an image region as a class and instance, then a background process collects all occurrences of this string or visually similar regions in the corpus and suggests them in a KWIC-like (or other) manner to the user. The user may have the chance to change them if they are not correct. These then get passed to an algorithm that, based on the surrounding context, can learn some rules and return new annotations on

other documents. Those with the highest confidence get presented to the user, who has the chance to change them, and so on.

Component dependencies

This component uses

- The *Ontology repository* component to obtain ontology information and for collaborative annotation.
- The *Data repository* component to access and insert information into the content sources.

4.6.3 Automatic annotation component

The *Automatic annotation* component is in charge of the automatic annotation of digital content (e.g. web pages) with concepts in the ontology. Occurrences in the considered content of instances of concepts in the ontology are automatically detected and subsequently annotated.

Functionalities provided

Like the *Manual annotation* component, this component is responsible for annotating occurrences of instances in the content with classes from the ontology (for example, to annotate "John Smith" with the concept "Person" or to annotate region *r1* that holds some specific visual characteristics as "Sea"). It differs in that the process is run by an engine such as the ontology-based information extraction (OBIE) or by the so called, in terms of non-textual content, ontology-based semantic analysis.

For the purposes of automatic annotation, the OBIE process needs to perform the following two tasks:

1. Identify mentions in the text, using the classes from the ontology instead of the flat list of types in “traditional” information extraction systems.
2. Perform disambiguation (for example, if we find two occurrences of “John Smith" in the text, of which one refers to a person, and the other to the name of the beer, the first might be annotated with the concept "Person" and the second with the concept "Beer").

It may also perform a third task: that of identifying relations between instances.

In a similar fashion, in order to carry out ontology-based semantic analysis of multimedia content in an automatic way, the following tasks are required:

1. Identify (possible) instances of the ontology concepts in the considered content by exploiting associations between concepts in the ontology and low-level features that can be automatically extracted. Such associations can be acquired either manually, semi-automatically (utilizing user feedback) or in a complete automatic way (employing machine learning techniques and ontology evolution strategies).
2. Identify relations between such instances (e.g., spatial, temporal, spatiotemporal ones).

3. Perform disambiguation. For example, due to similarities in appearance shared among different ontology concepts and the limitations in the multimedia processing algorithms, several regions in an image may have been identified as “Sea”. Knowledge about the context where the ontology concepts appear, their logical relations and semantics (e.g., a region depicting “Sea” can never be above one depicting “Sky”) enables to disambiguate such cases to a certain degree. Disambiguation is also required to resolve inconsistencies between instances identified by different modalities comprising the content (e.g., in an audiovisual content item, the auditory part (speech) and the visual one may produce contradicting annotations).
4. Identify (possible) instances of the ontology concepts in the content that corresponds to higher-abstraction semantics that cannot be directly identified by means of analysis. For example, annotations such as island, swimmer, scoring a goal, car overtake etc., require first the identification of more simple concepts, and the utilization of their logical relations in order to be detected.

Component dependencies

This component uses

- The *Ontology repository* component to obtain ontology information.
- The *Data repository* component to access and insert information into the content sources.
- The *Manual annotation* component to bootstrap learning.

4.6.4 Ontology populator component

This component is in charge of providing functionalities to automatically generate new instances in a given ontology from a data source. It links unique occurrences of instances in the content to instances of concepts in the ontology.

Functionalities provided

This component is similar in function to the *automatic annotation* component. However, it requires that instances be not only disambiguated (as with the *automatic annotation* component) but also that co-referring instances be identified. For example, if we find two occurrences of “John Smith" in the text, of which one refers to a person, and the other to the name of the beer, then our system should add the first as an instance of the concept Person, and the second as an instance of the concept "Beer". In contrast to the *automatic annotation* component, if we find an occurrence of "John Smith" in the text and an occurrence of "Mr Smith", the system must also identify whether they are referring to the same person or to two different people (or even that one occurrence is referring to the beer and the other to a person), and if they are co-referring, then only one instance should be added to the ontology. In a similar fashion, occurrences of concept instances in multimedia content that correspond to the same instance of concept need to be identified. For example, if we have two images for which the analysis shows that they both depict “John Smith”, or two video shots in which the same athlete is depicted, the system should add only one instance in the ontology. Furthermore, assuming an image whose area depicting Sky has been segmented into more than one region, the system should create one Sky instance and associate it with these regions.

Component dependencies

This component uses

- The *Ontology repository* component to insert ontology information.
- The *Data repository* component to obtain information from the content sources.
- The *Manual annotation* component to bootstrap learning.

4.6.5 Ontology instance generation implementations

The component that has more implementations found is the *Manual annotation* component having the rest very few implementations to choose.

4.6.6 Existing Implementations

The table below shows the implementations found for these dimensions:

Component	#	Implementations
Instance editor	2	GATE Ontology Editor, OCAT
Manual annotation	7	OCAT, OntoMat-Annotizer, M-OntoMat-Annotizer, PhotoStuff (Mindswap), AKTive Media - Ontology based annotation system, Ontolog, Magpie
Automatic annotation	3	KIM, AKTiveAgent, GATE ML
Ontology populator	3	CLIE, AKTive Futures, ALVIS

4.7 Semantic Web Services

This dimension includes those components that provide functionalities to discover, adapt/select, mediate, compose, choreograph, ground, and profile semantic web services.

The basic dependencies of the components in this dimension are shown in Figure 11:

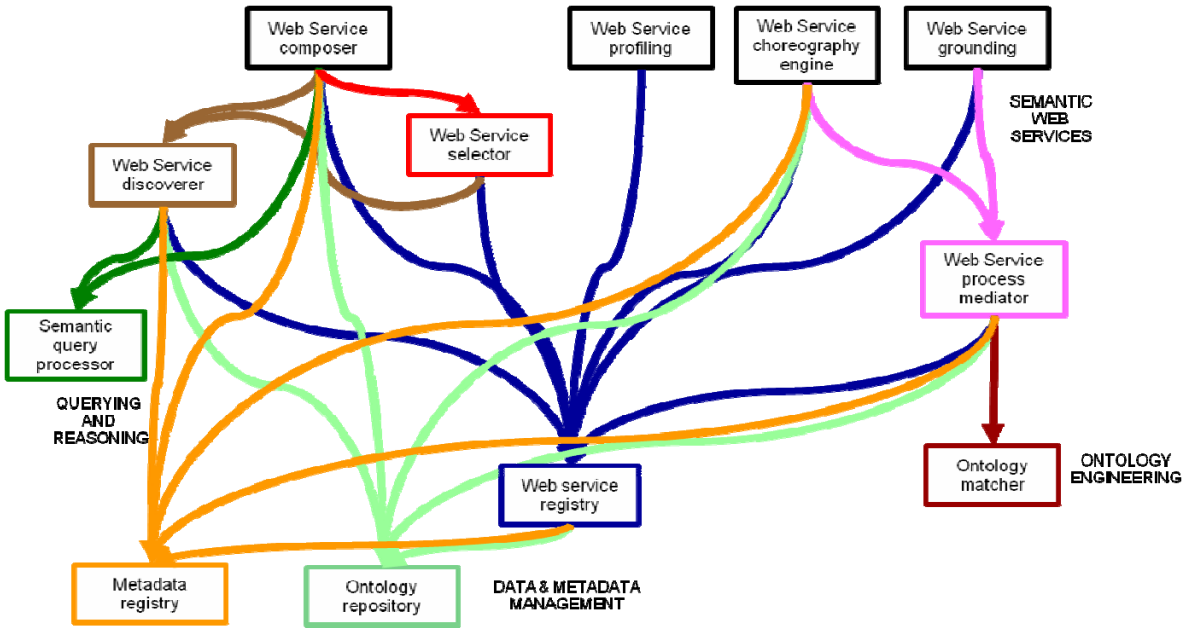

Figure 11. Dependencies of the components in the *Semantic Web Services* dimension.

4.7.1 Web Service discoverer component

This component will be in charge of providing functionalities to publish and search service registries, to control access to registries, and to distribute and delegate requests to other registries.

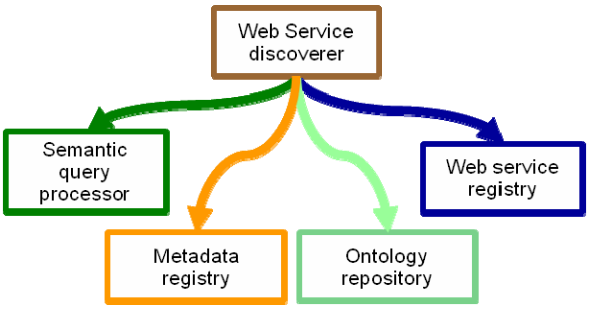

Figure 12. Dependencies of the components in the Semantic *Web Service Discoverer* Component.

Functionalities provided

The *Web Service discoverer* component provides a discovery engine based on keywords and existing annotations (WSDL, HTML docs, etc). The data set they operate on comes from publicly available Web service descriptions. Initially this has been limited to the information that can be obtained from the WSDL files. A search request can be expressed using keywords or advanced template search that allows querying for specific operation names or similar. WSDL documents can also be retrieved by URL. This phase will also include basic monitoring functionality for determining if the service specified in the given WSDL document is available.

Component Dependencies

This component uses

- The *Ontology repository* and *Metadata registry* components to perform service discovery.
- The *Semantic query processor* component in order to value the request and to infer correct discovery.
- The *Web Service registry* to access and store the Web Service data.

4.7.2 Web Service selector component

After discovering a set of potentially useful services, the *Web Service selector* component needs to check whether the services can actually fulfil the user's concrete goal and under what conditions. Those that cannot fulfil the goal are removed from the list of discovered services.

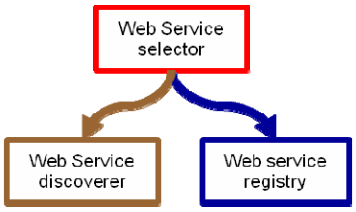

Figure 13. Dependencies of the components in the Semantic *Web Service Selector* component.

Functionalities provided

Negotiation or the process of checking whether and under what conditions a service can fulfil a concrete goal. It also encompasses the so-called filtering. By filtering we understand the process of narrowing the set of discovered services which provide the functionality requested, by considering only the services that have the appropriate non-functional properties requested by the user. Furthermore, building a ranking/order relation based on non-functional properties criteria like price, availability, etc. is also part of the filtering process. This process is called selection.
Negotiation, filtering and selection are tasks of the *Web Service selector* component.

Component Dependencies

This component uses

- The *Web service discoverer* component to discover the relevant set of Web services.
- The *Web Service registry* to access the Web Service data.

4.7.3 Web Service composer component

The *Web Service composer* component will be in charge of providing functionalities for designing a workflow of web services based on their choreography specifications.

Once the selection of the necessary Web services is done (by the *Web Service selector* component), the *Web Service composer* component will be in charge of the automatic composition of the latter Web services in order to provide new value-added Web services.

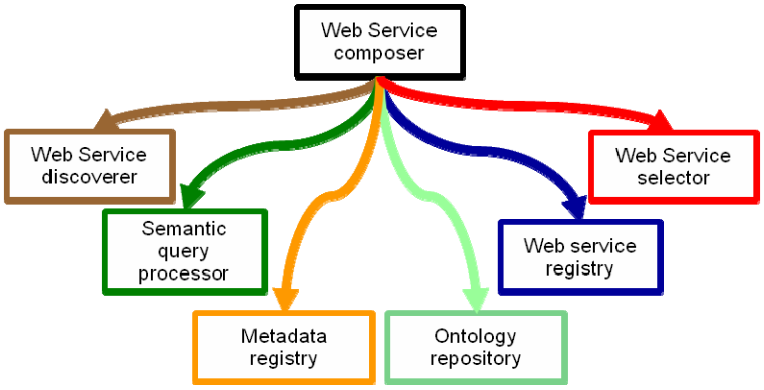

Figure 14. Dependencies of the components in the Semantic *Web Service Composer* component.

Functionalities provided

The *Web Service composer* component provides methods to do automatic Web Service composition, starting from web service descriptions at various levels of abstraction, specifically, the functional level and process level components, starting from requirements until the goal composition is achieved.

Component Dependencies

This component uses

- The *Web Service discoverer* to discover the relevant set of web services to compose.

- The *Web Service selector* to select appropriate Web services in the composition process.
- The *Ontology repository* and *Metadata registry* components to perform service composition.
- The *Semantic query processor* in order to infer correct composition.
- The *Web Service registry* to access the Web Service data.

4.7.4 Web Service choreography engine component

The *Web Service choreography engine* component will be in charge of providing functionalities to use the choreography descriptions of both the service requester and provider to drive the conversation between them.

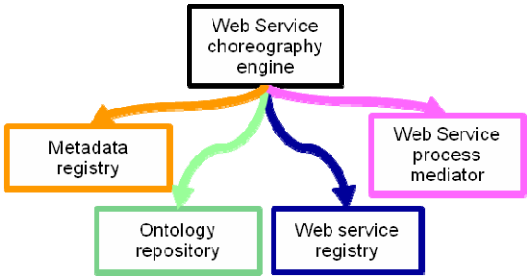

Figure 15. Dependencies of the components in the Semantic *Web service choreography engine* component.

Functionalities provided

The choreography part is meant to implement a process language which should allow for formal specifications of interactions and processes between the service providers and clients, define reasoning tasks that should be performed using this language, and implement an engine to support the execution of interactions, as well as to support reasoning in this language.

Component Dependencies

This component uses

- The *Ontology repository* and *Metadata registry* components to perform service choreography.
- The *Web Service process mediator* component.
- The *Web Service registry* to access the Web Service data.

4.7.5 Web Service process mediator component

The *Web Service process mediator* component will be in charge of providing functionalities to reconcile the public process heterogeneity that can appear during the invocation of web services. That is, ensuring that the public processes of the invoker and the invoked match.

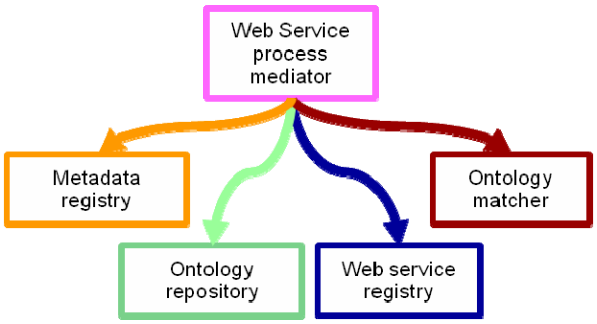

Figure 16. Dependencies of the components in the Semantic *Web service process mediator* component.

Functionalities provided

The main functionality of this component is mediation. Mediation aims at providing flexible mediation service at both data and process level. Data mediation provides automatic data transformation from the format used by the source party to the format required by the target party involved in conversation, while process mediation is concerned with the heterogeneity of the public processes of the participants in a conversation.

- **Data Mediation** provides automatic data transformation from the ontology used by the source party to the ontology required by the target party involved in conversation [22]. The Data Mediation Service has to support instance transformation from terms of one ontology to the terms of another ontology, based on the set of already created mappings between the two given ontologies.
- The **Process Mediator** service has the task of solving the communication (behavioural) mismatches that may occur during the communication between a requestor and a service provider [23]. The requestor is a Goal, while the provider is a Semantic Web Service; the Process Mediator's task is to accommodate the mismatches between the goal's requested Choreography and the Semantic Web Service's choreography.

Component Dependencies

This component uses

- The *Ontology repository* and *Metadata registry* components to perform service choreography.
- The *Ontology matcher*, the *Data translator* and the *Mediator generator* components to perform data mediation.
- The *Web Service registry* to access the Web Service data.

4.7.6 Web Service grounding component

This component is responsible for the communication between Web Services i.e., to send the necessary request messages and receive the responses.

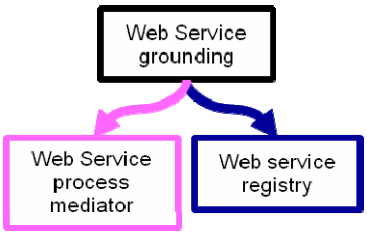

Figure 17. Dependencies of the components in the Semantic *Web service grounding* component.

Functionalities provided

Because internal communication within a Semantic Web Service Architecture uses semantic data and practically all currently deployed Web services use their specific XML formats, the External Communication component needs to translate the involved data forms. This translation is also known as data grounding [24]. Above that, this proposed architecture also needs to support concrete network protocols (HTTP, SOAP, other bindings) to be able to exchange messages with the Web service.

Component Dependencies

This component uses

- The *Web Service process mediator* component.
- The *Web Service registry* to access the Web Service data.

4.7.7 Web Service profiling component

The *Web Service profiling* component will be in charge of providing functionalities to create web service profiles based on their execution history.

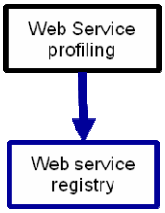

Figure 18. Dependencies of the components in the Semantic *Web service profiling* component.

Functionalities provided

The *Web Service profiling* component is responsible for creating service profiles. A service profile should be understood as a subset of non-functional parameters, mainly Quality of Service (QoS) attributes e.g. price, execution time, etc. characterizing a Web service. It allows services comparison on the basis of non-functional parameters and choosing the service most suited to a user's needs.

As it is, the *Web Service profiling* component should be responsible for collecting, computing, and providing other components with values of the selected non-functional parameters.

Component Dependencies

This component uses

- The *Web Service registry* to store the Web Service profiles.

4.7.8 Web Service Registry component

The *Web Service registry* component will be in charge of providing functionalities to register semantic Web services i.e., supporting semantic publication and discovery (based on some specific semantic Web service approach). Therefore Semantic Web services are referred to in an online collection of semantic Web services.

Functionalities provided

The *Web Service Registry* component is responsible for storing Semantic Web services according to their description and profile.

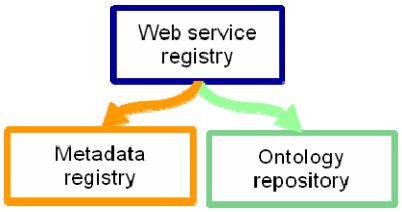

Figure 19. Dependencies of the components in the Semantic *Web service registry* component.

Component Dependencies

This component uses

- The *Ontology repository* and *Metadata registry* components to perform semantic registries.

4.7.9 Semantic Web Services implementations

There are Semantic Web Service frameworks, such as WSMX, that encompass the functionalities of the components of the *Semantic Web Services* dimension. Nevertheless, the components of these frameworks cannot interoperate with components of other frameworks (e.g. OWL-S vs WSMO).

With respect to the *Web Service profiling* component there are only two implementations and the profile creation functionality is available via Java API. Service execution data (log files) is required to perform service profiling. Other sources of information on Web services (feedback from users, initial service description given by service providers) may be also taken into account when creating the service profile. In general, the more data describing/evaluating the service available, the more adequate and precise the profile generated.

Most of the semantic Web service registries are private registries with restricted access. Therefore, any public registries of semantic Web services that we can access without a special authorization do not yet exist. All applications interacting with a registry assume an ad hoc registry of semantic Web services. For instance, any service discoverer or composer suggests discovering and composing a set of Web services from an ad hoc registry e.g., OWL Semantic Search Services, WSMX Discovery Framework. A first proposal of a public semantic service discovery is the OWL-S UDDI Matchmaker.

4.7.10 Existing implementations

The table below shows the implementations found for these dimensions:

Component	**#**	**Implementations**
Web service discoverer	4	Hybrid OWL-S Web Service Matchmaker - OWLS MX, The TUB OWL-S Matcher (The OWLSM), WSMX Discovery Framework, OWL Semantic Search Services (owl-semsearch)
Web service selector	1	WSMX Selector and Ranking Prototype
Web service composer	6	Kweb Semantic Web Service Composition, Semantic Web service composition through Cusal Link Composition, Composer, Semantic web services browser and composer, Web service Composition, Service Composition Engine (Developed within ASG)
Web service choreography engine	2	WSMX Choreography Engine, IRS-III
Web service process mediator	1	WSMX Process Mediation Prototype
Web service grounding	1	WSMX Communication Manager
Web service profiling	2	Service Profiler, Web Service profiling
Web service registry	2	OWL-S UDDI Matchmaker, OWLS-TC

5 Use Cases and the Semantic Web Framework

This chapter provides a description of how the Semantic Web Framework and its components could support the development of the Knowledge Web use cases, as are described in the deliverable *D1.1.4 v1: System and knowledge technology components for prototypical applications and business cases* [3].

For each case, we provide a brief description of the system to be implemented, a figure with the components of the Semantic Web Framework that could be used in the use case (including the dependencies identified between the components in the previous chapter), and the functionalities of the use case that are covered by these components.

In this second version of the Semantic Web Framework, use cases 3 and 4 have been reviewed according to the feedback collected from the use case partners. The rest of the use cases have been refined according to the changes made to the components (Chapter 4).

Appendix II includes a table showing the dependencies of the use cases with all the components of the Semantic Web Framework.

5.1 Use Case 1. Recruitment from Worldwidejobs

The aim of the online recruitment system is to facilitate filling open job vacancies by finding the best qualified and suitable candidate in the fastest time. The system will allow job seekers to overview all the offers published on the internet in the different portals and websites with open positions, and to assess their suitability to vacancies. The system will also allow companies to post their vacancies with a formal meaning, to gain diffusion of the vacancies and will allow semi-automatic matching between candidates and vacancies.

Figure 20 presents the components of the Semantic Web Framework that can support this use case and their dependencies.

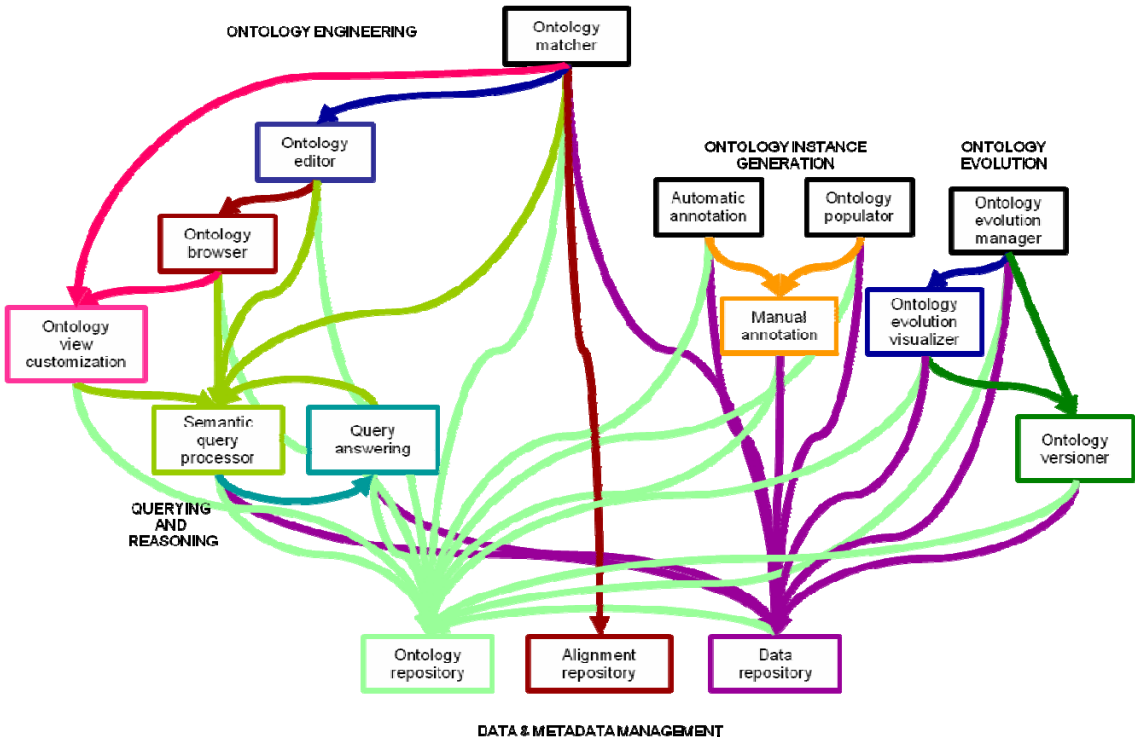

Figure 20. Dependencies between the SWF components in the use case 1.

In order to achieve all of the goals proposed in the business use case, the system could use the following SWF components:

- The *Ontology matcher* component for the automatic matching of job postings and job applications, and for managing the created alignments.
- The *Manual* and *Automatic annotation* components or the *Ontology populator* component for publishing the job postings using the employers' existing legacy infrastructure.
- The *Ontology editor* component for editing the system ontologies, since the information is subject to change and the ontologies should be updated. This component will also use the *Ontology browser*.
- The components of the *Ontology evolution* dimension for managing the evolution of the ontologies.
- The *Semantic query processor* and the *Query answering* components for supporting reasoning tasks in different components of the system.
- The *Ontology*, *Data* and *Alignment repository* components for managing the data storage and retrieval.

5.2 Use Case 2. B2C portals from France Telecom

The system's objective is to complete the commercial perimeter of the current holiday package offers, with some dynamically packaged solutions to meet the customers' expectations (holidays, week-end, all leisure services) by offering users a one-stop browsing and purchasing personalized tourism packages through a dynamic combination of various tourism offers (e.g., travel, accommodation, meals) from different providers.

Figure 21 presents the components of the Semantic Web Framework that can support this use case and their dependencies.

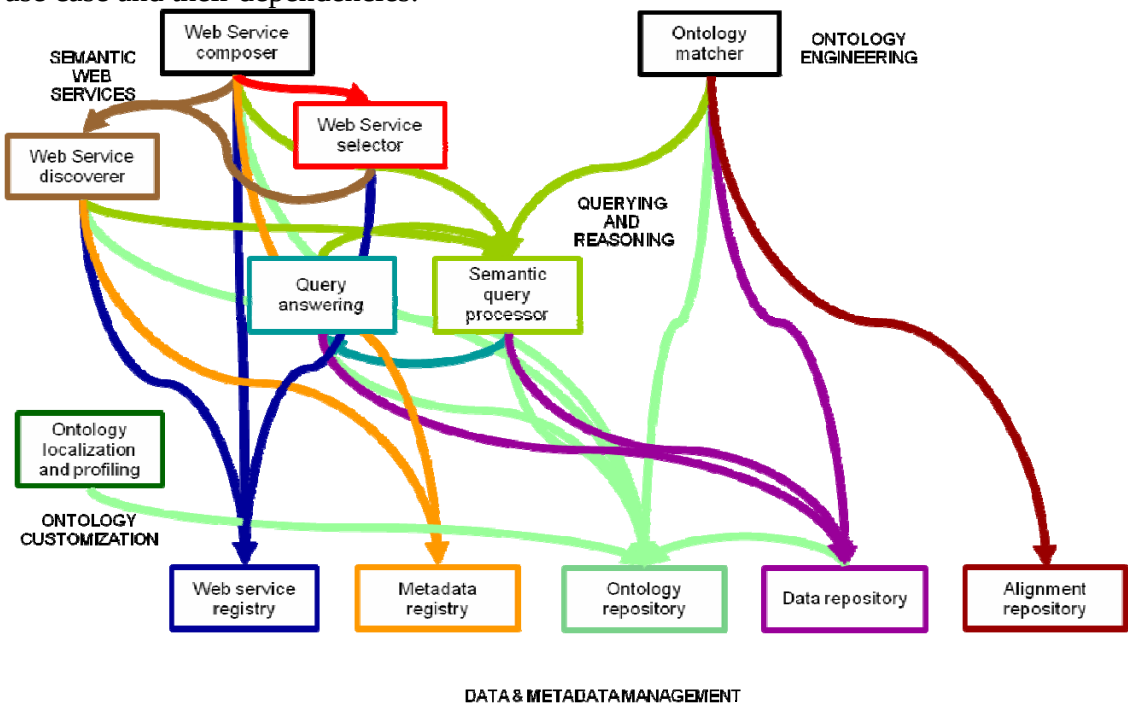

Figure 21. Dependencies between the SWF components in the use case 2.

In order to achieve all of the goals proposed in the business use case, the system could use the following SWF components:

- The integration in the system of new package offers or service providers and the aggregation into the system of content from these providers can be performed by using the *Ontology matcher* component for defining and managing the alignments of the provider ontologies to the system ontologies. The *Ontology matcher* component also generates wrappers for using the information from the providers in the system.
- The *Ontology profiler* component for deducting information about users from their requests and for inferring relations between concepts from the user profile.
- The components of the *Semantic Web Services* dimension can provide the mechanisms for implementing the system as a Service Oriented Architecture where some re-usable components are made available through web services interfaces.
- The *Semantic query processor* and the *Query answering* components for supporting reasoning tasks in different components of the system.
- The *Ontology*, *Data* and *Alignment repository* components for managing the data storage and retrieval.

5.3 Use Case 3. News aggregation from Neofonie

The system deals with the provision of an aggregated news service able to provide business clients with accurate search, thematic clustering, classification of news stories, and e-mail notification of stories of interest. The news sources are not just the main news

feeds and media outlets but also press releases, announcements on websites and other "alternative" sources.

Following the presentation of the framework and the resulting discussion with the use case provider Neofonie GmbH, the next figure shows the high level architecture of a proposed semantic solution for the use case.

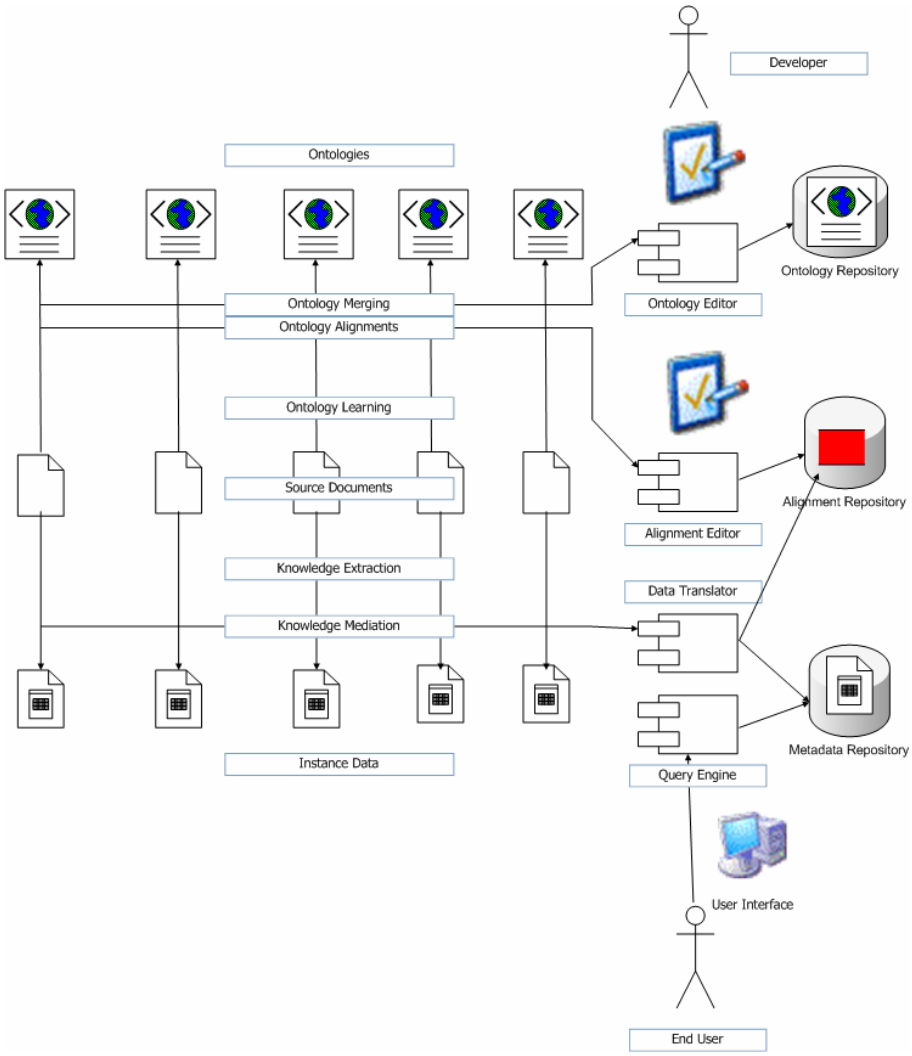

Figure 22. High level architecture of a proposed semantic solution for the use case 3.

Figure 23 presents the components of the Semantic Web Framework that can support this use case and their dependencies.

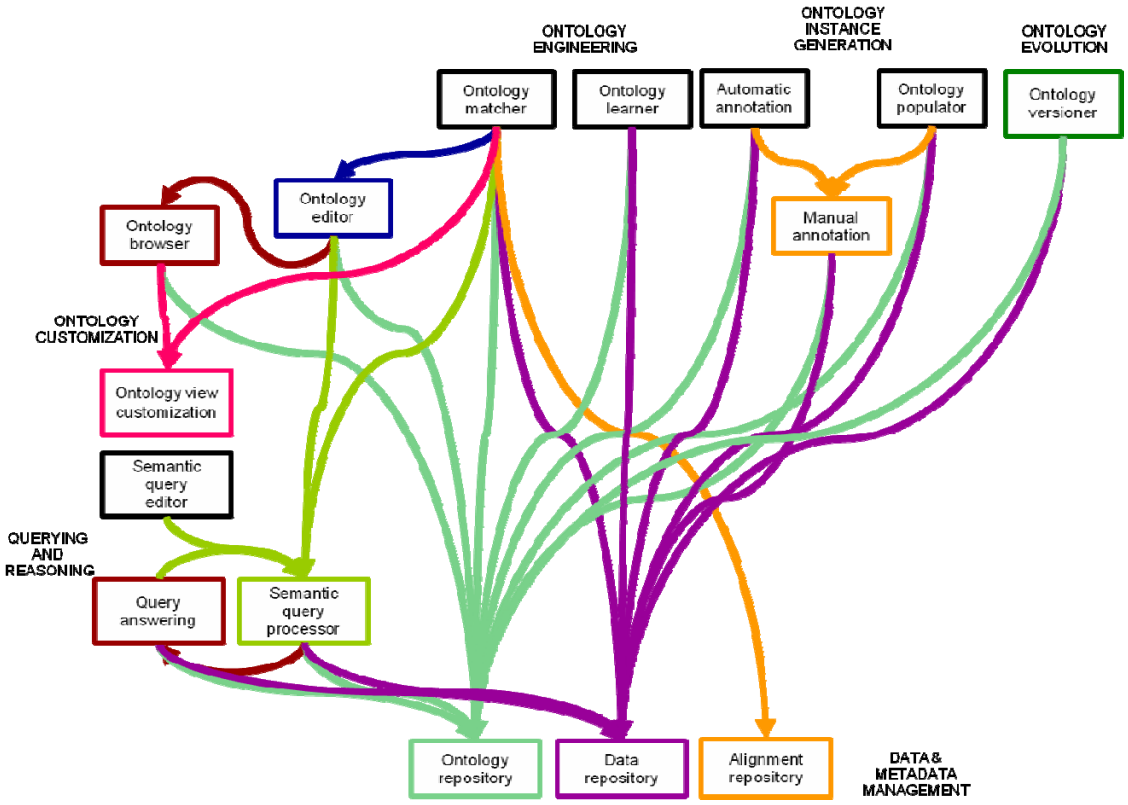

Figure 23. Dependencies between the SWF components in the use case 3.

In order to achieve all of the goals proposed in the business use case, the system could use the following SWF components:

- The *ontology repository*, the *data repository*, the *alignment repository* and the *metadata registry* store all the data necessary for the use case: the ontologies used for each source, the instance data extracted from these sources and the alignments that have been created between each source ontology.
- The *query answering*, the *semantic query processor* and the *semantic query editor* provide both the user interface support for formulating the query and displaying the results and the system-intern support for performing the query across the aligned instance data and extracting the results.
- Ontologies for representing the data of each source are created semi-automatically using ontology learning techniques through the *ontology learner* component. The initial ontology extraction is refined using the *ontology browser* component to view the ontology and the *ontology editor* component to complete the ontology manually.
- It is possible that with the use of the system over time, the ontologies will need to be revised as new concepts or properties gain relevance. Hence the *ontology versioner* component may be employed at a later stage in the system. Likewise, in the ontology extraction part, extracted terms may overlap with those of existing ontologies for related domains such as politics, sport etc.

- Given the existence of an ontology that represents terms from a certain source, knowledge extraction can take place. Instance data is generated through semi-automatic annotation approaches using the *automatic annotation* component, the *manual annotation* component for adding semantic data to news sources, and the *ontology population* component.
- Finally, two approaches to resolving searches can be considered. One is that queries are expressed in terms of one ontology and, at run time, they are mapped into the other ontologies of the sources; then they are executed across the different source data and the results are combined at the end. However, this approach is very resource intensive at query time. Given that we update the source data only periodically, it makes better sense to transform all source data into a core ontology, which can be built from the merge of all source ontologies. Then we generate first alignments between the source ontologies and a core ontology using the *ontology matcher* component. These alignments need manual proofing and correction. The alignments also help to refine the core ontology (e.g. stripping equivalent terms, as synonymy will be captured in the alignments). Given now a core ontology and alignments to the individual source ontologies, mediators can be generated for the transformation of instance data from any source in terms of the core ontology. Hence a core knowledge base is maintained against which the queries are executed.

5.4 Use Case 4. Product lifecycle management from Semtation

The system is intended to be used for developing and maintaining product catalogues throughout the product lifecycle, by explicitly modelling product knowledge according to an agreed, shared terminology for the product domain.

Following the presentation of the framework and the resulting discussion with the use case provider Semtation, Figure 24 presents the components of the Semantic Web Framework that can support this use case and their dependencies.

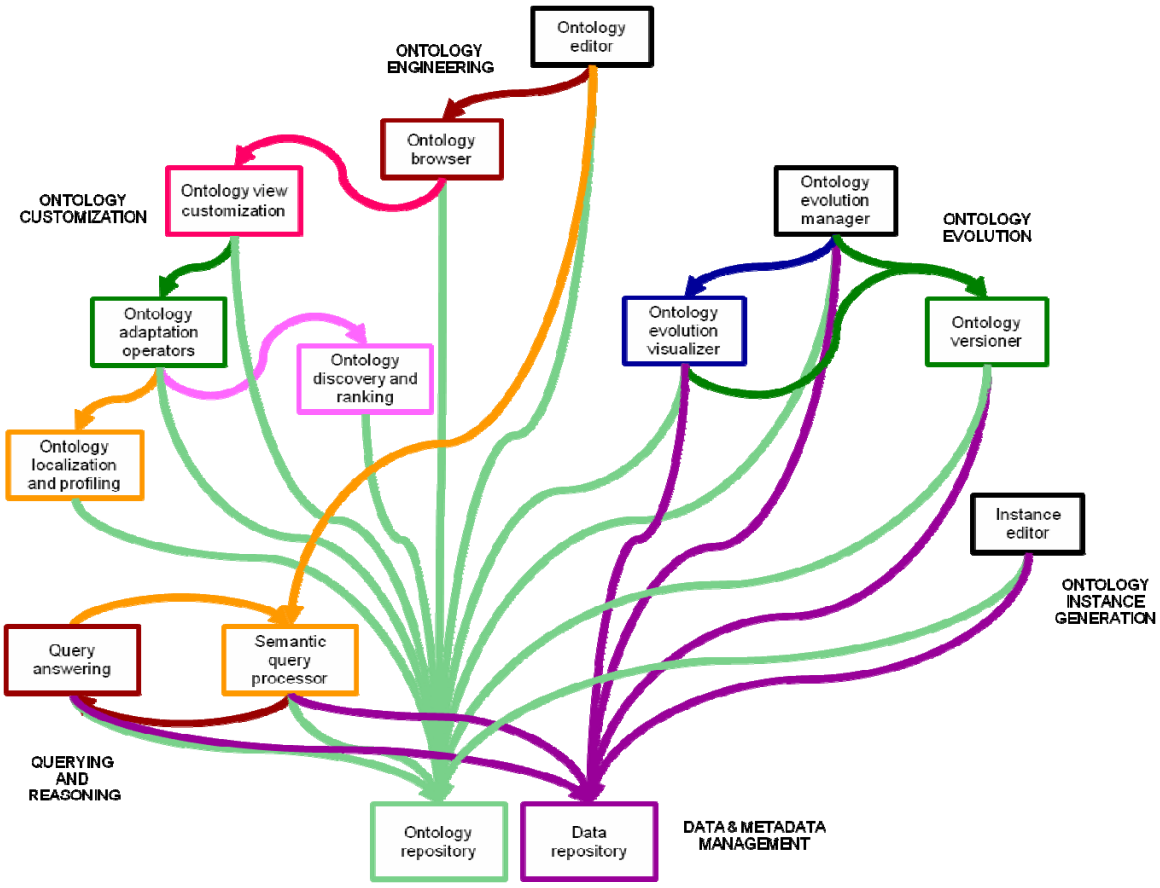

Figure 24. Dependencies between the SWF components in the use case 4.

In order to achieve all of the goals proposed in the business use case, the system could use the following SWF components:

- Product ontologies are stored in the *ontology repository* component.
- Product models are stored in the *data repository* component.
- While product ontologies may be standardised in the industry for use in such systems, and hence the ontology engineering effort reduced for individual enterprises, an *ontology browser* and *editor* component will allow understanding the ontology and modifying it when necessary.
- *Ontology customization* components are important in this use case to ensure that different users can access the data in appropriate ways, e.g. providing different views of the ontology depending on the department of the company.
- Given that a product portfolio will change over time, also in terms of its characteristics, *ontology evolution* components may also be necessary. Versioning management ensures compatibility between products described at different times by the evolving ontology.
- An *instance editor* component is used to generate the product models. In order to check against ontology/rule-defined restrictions (to keep product models consistent) a

semantic query processor component is used with a *query answering* component as the consistency checker.

5.5 Use Case 5. Managing Knowledge at Trenitalia

The knowledge management system should be able to support the activities of a major business unit in TSF Trenitalia, UTMR (Unità Tecnologie Materiale Rotabile), which is responsible for the design, maintenance and engineering of rolling stocks manufactured by external suppliers. Such a unit is composed of heterogeneous and specialized professional communities that need to manage locally their knowledge and, above all, to exchange knowledge across communities meaningfully and, particularly, to solve the semantic issues posed by the need to preserve linguistic heterogeneity while facilitating coordination and collaboration.

Figure 25 presents the components of the Semantic Web Framework that can support this use case and their dependencies.

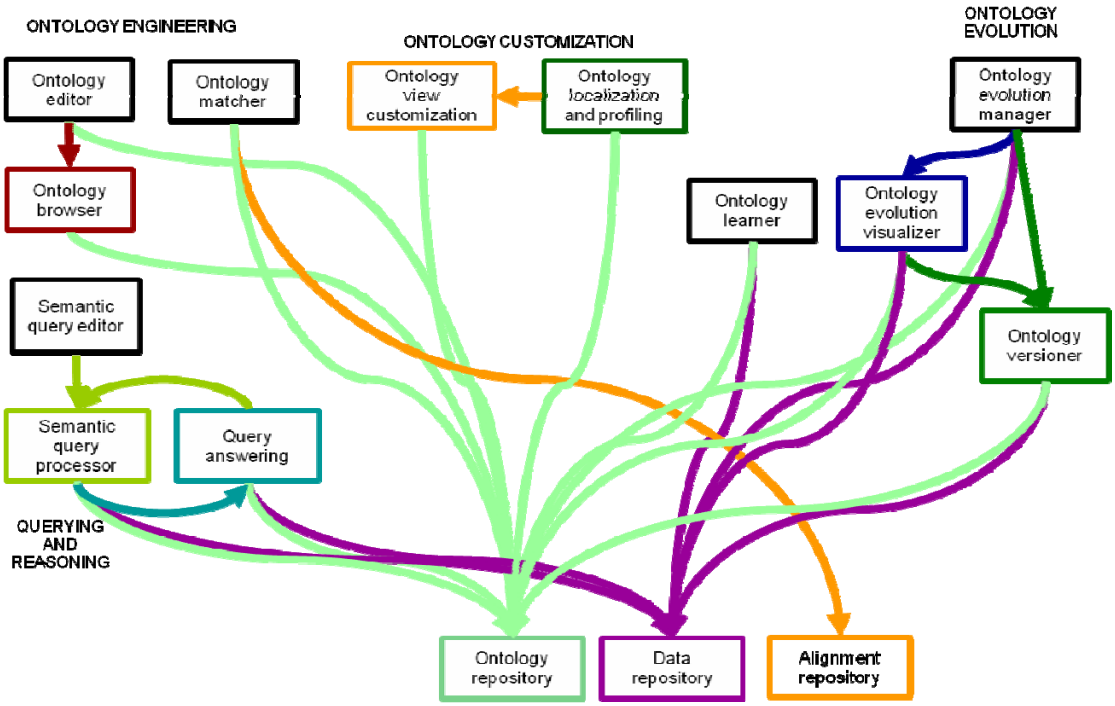

Figure 25. Dependencies between the SWF components in the use case 5.

In order to achieve all of the goals proposed in the business use case, the system could use the following SWF components:

- The *Ontology learner* component to discover semantic relationships and new structures.
- The three components in the *ontology evolution* dimension to track the evolution of individual ontologies.
- The *Ontology matcher* component to map entities of different ontologies.
- The *Ontology merger* to merge ontologies and form a new one.

- The *Ontology editor*, the *Ontology visualize* and the *Ontology browser* in order to allow users to evolve their personal ontologies.
- The three components in the *querying and reasoning* dimension to interpret the queries for retrieving knowledge, to ensure the uniqueness of each result, and to rank the results.
- The *Ontology profiler* to be able to consider different types of users.

5.6 Use Case 6. Integrated Access to Biological Data from Robotiker

The system should provide a unified point of access for different biological data repositories, and these can be accessible through internet (Nucleotide Sequences, amino acid sequences, etc.); corporate databases; results of experiments (DNA-chips); health cards; medical literature sites, etc.

The system should also support the generation and extraction of knowledge from biological data by means of ontologies, the combination of them (ontology merging) and/or the association of them (ontology mapping); this knowledge is to be exploited by means of annotations, intelligent agents, semantic web services and/or semantic grid.

Figure 26 presents the components of the Semantic Web Framework that can support this use case and their dependencies.

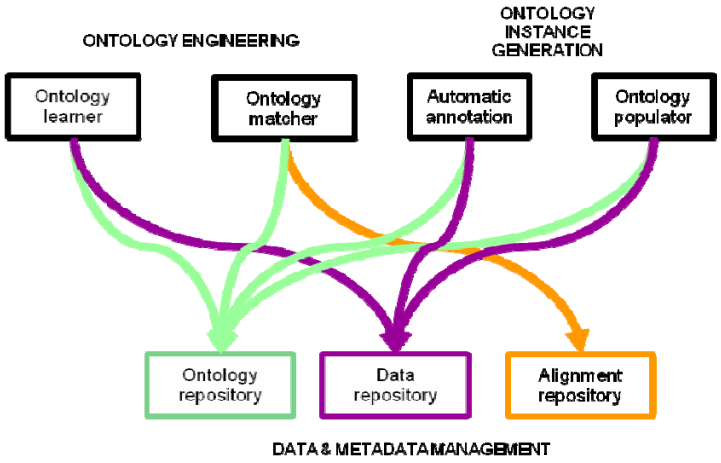

Figure 26. Dependencies between the SWF components in the use case 6.

In order to achieve all of the goals proposed in this business use case, the system could use the following SWF components:

- The *Alignment editor* component to allow domain experts to define the alignments between the ontologies to be merged.
- The *Ontology matcher* component to map similar concepts in different ontologies and to obtain a more complete ontology using several ontologies, at least one ontology for each data repository/domain to be aggregated.
- The *Automatic annotation* and the *Ontology populator* components to produce content metadata from the biological data sources, based on the defined ontologies.

- The *Ontology learner* component to extract knowledge from biological data repositories.

5.7 Use Case 7. Semantic Web needs for the Petroleum Industry

The current goals of the Institut Français du Pétrole (IFP) are to develop knowledge models for semantic memories and intelligence of CO2 mitigation, a key application target now that the Kyoto Protocol has entered into force. For this aim, ontologies of chemical processes (flowcharts, chemical compounds, structures, experimental results…) and of geosciences (geological models, geophysical data, wells…) are needed.

This business case deals with the management of large collection of project documents (in the form of texts, geological maps, software, subsurface models, data bases, etc.) in the field of geosciences. IFP goal is to access the documents produced by these projects in a structured manner, so that a new project can make the best use of the results produced by previous projects.

Figure 27 presents the components of the Semantic Web Framework that can support this use case and their dependencies.

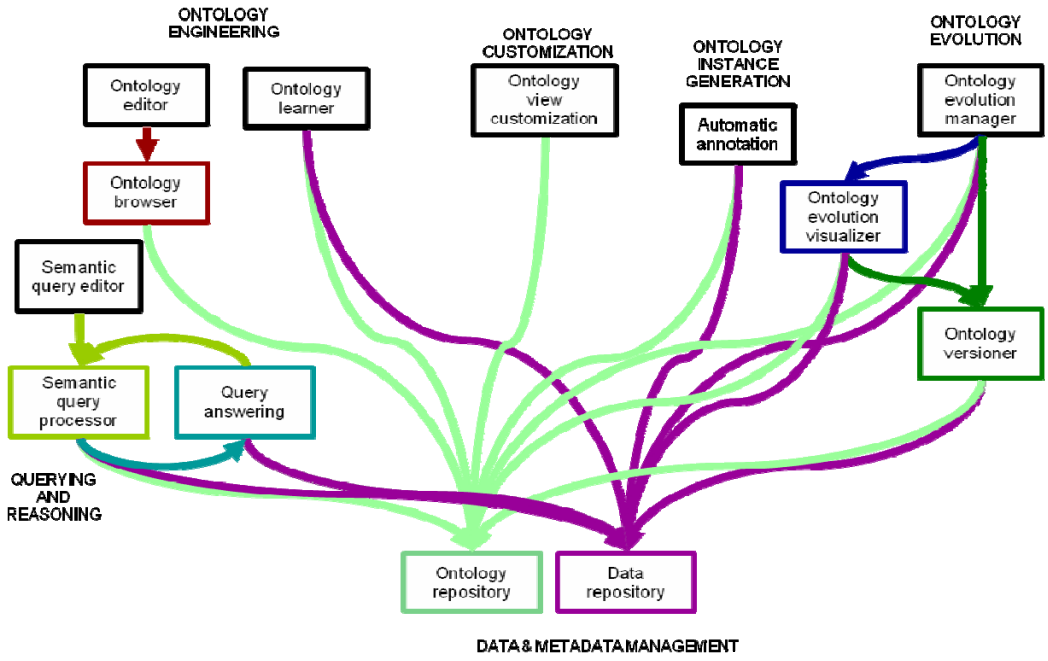

Figure 27. Dependencies between the SWF components in the use case 7.

In order to achieve all of the goals proposed in this business use case, the system could use the following SWF components:

- The *Ontology editor* and the O*ntology repository* to develop domain ontologies either from scratch or by re-using existing relevant ontologies.
- The *Ontology browser* to navigate through the ontology, and the *Ontology view customization* to visualize ontologies in an intuitive way
- The *Ontology learner* to determine concepts and their relations through data analysis.

- The *Automatic annotation* to annotate the produced documents (software, models, data bases, etc.) with semantic markers from ontologies.
- The three components in the *querying and reasoning* dimension to query the geosciences projects semantic memory to find relevant documents.
- The three components in the *ontology evolution* dimension to maintain large domain ontologies in complex evolving technical domains.

5.8 Use Case 8. Hospital Information System from L&C Global

This use case deals with the issue of database integration in the domain of healthcare. Health care organisations such as hospitals may have several dispersed data sources containing interrelated information. For example, there may be a central repository which contains administrative information of all patients registered at the hospital. Additionally, each division holds additional information about the diagnoses and treatments of the patients examined. As information stored about a patient in one division may be relevant to a professional seeking information from another division, a unified search is highly desired. Two challenges in this case are that the data may be stored in very different fashions, from totally unstructured text (e.g. notes written by the doctor) to highly structured repositories (e.g. relational databases), and that access must be achieved within a suitable time frame.

Figure 28 presents the components of the Semantic Web Framework that can support this use case and their dependencies.

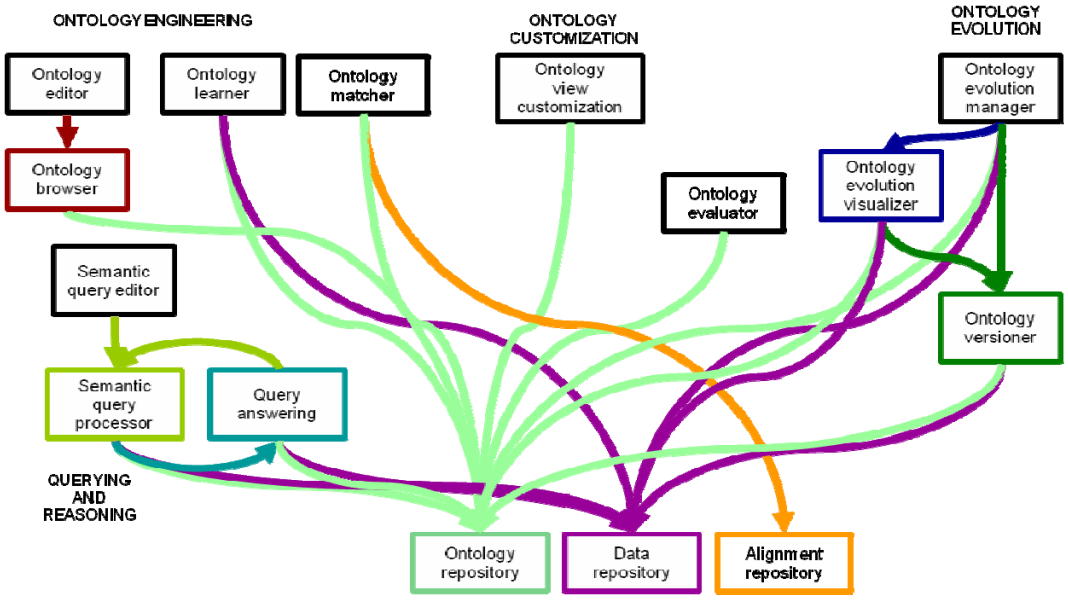

Figure 28. Dependencies between the SWF components in the use case 8.

In order to achieve all of the goals proposed in this business use case, the system could use the following SWF components:

- The *Ontology editor*, the *Ontology browser*, and the three components in the *ontology evolution* dimension for the development and continuing maintenance of ontologies.
- The *Ontology view customization* to visualize ontologies in an intuitive way.

- The *Ontology evaluator* to evaluate the data storage (e.g. removing redundancy).
- The *Ontology learner* to generate suggestions for concepts and properties through analysis of data corpus.
- The three components in the *querying and reasoning* dimension to retrieve knowledge.
- The *Ontology repository* and the *Data repository* to store instance data.
- The *Ontology matcher* to map different ontologies.
- The *Alignment repository* to store the alignments done between ontologies.

6 Conclusions and future work

The Semantic Web Framework (SWF) is intended to help developers to build Semantic Web applications and to diminish the cost of this development.

During the last two years, inside workpackage 1.2, eleven Knowledge Web partners have devoted their efforts to the definition of the SWF, which is an initial step to providing foundation for the large-scale development of Semantic Web applications. Here we present a first definition of the SWF as a component-based framework, describing the existing types of Semantic Web technology, their functionalities, and the dependencies between these technologies.

This first definition required a common consensus on vocabulary for defining semantic components, which in turn required a consensus on the names and definitions of all the components of the SWF.

Furthermore, to provide real-world users and application developers not just the theoretical components but ready-to-use implementations of them, we searched and described existing reusable Semantic Web tools and categorized them into SWF components.

We also had the opportunity to refine and validate the SWF, not just inside Knowledge Web but also within the companies that provided the eight use cases considered in Knowledge Web. We presented to these companies the SWF and our view of their use cases using the SWF, and contrasted our view with their own vision of the use case.

Although the SWF is useful as a reference and helps reuse existing technology, Semantic Web application developers will still have to develop applications and their functionalities.

Nevertheless, application development can be less expensive, as can be observed in the business use cases dealt with in Chapter 5, by first identifying the semantic functionalities needed and then reusing the corresponding components of the SWF.

Chapter 5 shows that some components such as the *Ontology repository*, the *Data repository* and the *Metadata registry* are used in all the use cases, whereas other components such as the *Alignment repository*, the *Query answering*, the *Semantic query processor*, the *Ontology editor*, *Ontology browser*, the *Ontology view customization*, the four components of the *Ontology evolution* dimension, and the *Ontology matcher* are used in all the use cases.

On the other hand, components such as the *Information directory manager*, the *Ontology evaluator*, the *Ontology discovery and ranking*, the *Ontology adaptation operators*, the *Instance editor* and all the components of the *Semantic Web Service* dimension are not used at all or are slightly used in the use cases.

Every component described in this document has a set of implementations and each of them covers totally or partially the functionality described. To sum up, 200 component implementations have been referenced. Some of these implementations have been taken

from the *Semantic Web Tools and Applications Information Repository*[4] developed in workpackage 1.4.

A future line of work will be to develop specifications of the components identified in this deliverable and of their interfaces and guidelines for implementing or reusing them.

One extension of the SWF will include a new dimension for social components. Work in this direction is being carried out in the Avanza project PLATA (FIT-350503-2007-6).

[4] http://cgi.csc.liv.ac.uk/KWebToolsSurvey/

References

1. *D1.2.4 Architecture of the Semantic Web Framework*. Raúl García-Castro, M. Carmen Suárez-Figueroa, Asunción Gómez-Pérez, Diana Maynard, Stefania Costache, Raúl Palma, Jérôme Euzenat, Freddy Lécué, Alain Léger, Tomas Vitvar, Michal Zaremba, Dominik Zyskowski, Monika Kaczmarek, Martin Dzbor, Jens Hartmann, Stamatia Dasiopoulou. Knowledge Web technical report, February 2007.
2. *D1.2.2: Semantic Web Framework requirements analysis.* Wolf Siberski, J. Euzenat, J. Hartmann, A. Léger, D. Maynard, J. Pan, M.C. Suárez-Figueroa, et al. Knowledge Web technical report, June 2005.
3. *D1.1.4 v1: System and knowledge technology components for prototypical applications and business cases* A. Léger, L. Nixon, F. Paulus, L. Rocuet, M. Mochol, Y. Kompatsiaris, V. Papastathis, S. Dasiopoulou, M. Jarrar, R. Cuel, M. Bonifacio. Knowledge Web technical report, June 2005.
4. *Component Software, Beyond Object Oriented Programming*. Clemens Szyperski. Addison-Wesley, 1998.
5. *Component-Based Software Engineering: Putting the Pieces Together*. George T. Heineman, William T. Councill. Addison-Wesley Professional, 2001.
6. *IEEE Recommended Practice for Architectural Description of Software Intensive Systems*. IEEE Std 1471-2000.
7. *Frameworks = (Components + Patterns)*. Ralph E. Johnson. Communications of the ACM 40(10): 39-42, October 1997.
8. *DSSA frequently asked questions*. W. Traz. ACM Software Engineering Notes 19(2): 52-56, April 1994.
9. *The Semantic Web*. T. Berners-Lee, J. Handler and O. Lassila. Scientific American. May 2001.
10. *Next Generation Semantic Web Applications*. Enrico Motta and Marta Sabou. Proc. of the 1st Asian Semantic Web Conference (ASWC), September 2006.
11. *D1.2 Analysis of the State-of-the-Art in Ontology-based Knowledge Management*. Peter Mika and Hans Akkermans. SWAP Project, February 2003.
12. *Software Architecture: Perspectives on an Emerging Discipline*. Mary Shaw and David Garlan. Prentice Hall. 1st edition, April 1996.
13. *Lifecycle-Support in Architectures for Ontology-Based Information Systems*. Thanh Tran, Peter Haase, Holger Lewen, Óscar Muñoz-García, Asunción Gómez-Pérez and Rudi Studer. Proceedings of the 6th International Semantic Web Conference pp. 508-522, November 2007.
14. *Software Reuse*. Charles W. Krueger. ACM Comput. Surveys 24(2): 131-183, June 1992.

15. *Description Logic Handbook: Theory, Implementation and Applications*. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider. Cambridge University Press, 2003.

16. *The role of semantics in e-government service model verification and evolution*. L. Stojanovic, A. Abecker, D. Apostolou, G. Mentzas, R. Studer. The Semantic Web meets eGovernment Symposium. 2006 AAAI Spring Symposium Series. Stanford University, California, USA, March 27-29, 2006.

17. *A Framework to Verify Knowledge Sharing Technology*. Gómez-Pérez A. 1996. Expert Systems with Application 11(4): 519–529.

18. *Evaluation of Ontologies*. Gómez-Pérez A. 2001. International Journal of Intelligent Systems 16(3):391–409.

19. *Semi-automatic data-driven ontology construction system*. Fortuna, B; Mladenic, D. and Grobelnik, M. In Proc. of the 9th Multiconference on Information Society. pp. 223-226. 2006

20. *User profiling for interest-focused browsing history*. M. Grcar, D. Mladenic and M. Grobelnik. In Proc. of the 9th Multiconference on Information Society. pp. 223-226. 2006

21. *Methods and Tools for Ontology Evolution*. L. Stojanovic. Dissertation. 2004. Referee: Rudi Studer, Christof Weinhardt, Asunción Gómez-Pérez.

22. *Using Uneven Margins SVM and Perceptron for Information Extraction*. Y. Li, K. Bontcheva, and H. Cunningham. In Proceedings of Ninth Conference on Computational Natural Language Learning (CoNLL-2005), 2005.

23. *Mediation Enabled Semantic Web Services Usage* E. Cimpian, A. Mocan, M. Stollberg, 1st Asian Semantic Web Conference (ASWC2006), Beijing, China, September 2006.

24. *Semantic Web Services Grounding* J. Kopecky, D. Roman, M. Moran, and D. Fensel. In Proc. of the Int'l Conference on Internet and Web Applications and Services (ICIW'06), Guadeloupe, February 2006.

25. *Ontology matching.* J. Euzenat and P. Shvaiko, Springer-Verlag, Berlin (DE), 2007.

26. *Digital Repositories Review: February 2005*. Rachel Heery, UKOLN, University of Bath and Sheila Anderson, Arts and Humanities Data Service, February 2006.

Appendix I Dependencies between the components of the Semantic Web Framework

This appendix includes tables that show the dependencies of all the components of the Semantic Web Framework with the components of each dimension.

	Information Directory Manager	Ontology Repository	Data Repository	Alignment Repository	Metadata Registry
Information Directory Manager		X	X	X	X
Ontology Repository					
Data Repository					
Alignment Repository					
Metadata Registry					
Query Answering	X				
Semantic Query Processor	X	X			
Semantic Query Editor					
Ontology Editor		X			
Ontology Browser		X			
Ontology Evaluator		X	X		
Ontology Learner		X	X		
Ontology Matcher		X	X	X	
Ontology Discovery & Ranking		X			
Ontology Localization & Profiling		X			
Ontology Adaptation Operators		X			
Ontology View Customization		X			
Ontology Evolution Manager		X	X		
Ontology Evolution Visualizer		X	X		
Ontology Versioner		X	X		
Instance Editor		X	X		
Manual Annotation		X	X		
Automatic Annotation		X	X		
Ontology Populator		X	X		
Web Service Registry		X			X
Web Service Discoverer		X			X
Web Service Selector					
Web Service Composer		X			X
Web Service Choreography Engine		X			X
Web Service Process Mediator		X			X
Web Service Grounding					
Web Service Profiling					

Table 1 Dependencies of the components with the components of the *Data and metadata management* dimension

	Query Answering	Semantic Query Processor	Semantic Query Editor
Information Directory Manager			
Ontology Repository			
Data Repository			
Alignment Repository			
Metadata Registry			
Query Answering			
Semantic Query Processor	X		
Semantic Query Editor		X	
Ontology Editor		X	
Ontology Browser			
Ontology Evaluator		X	
Ontology Learner			
Ontology Matcher		X	
Ontology Discovery & Ranking			
Ontology Localization & Profiling			
Ontology Adaptation Operators			
Ontology View Customization			
Ontology Evolution Manager			
Ontology Evolution Visualizer			
Ontology Versioner			
Instance Editor			
Manual Annotation			
Automatic Annotation			
Ontology Populator			
Web Service Registry			
Web Service Discoverer		X	
Web Service Selector			
Web Service Composer		X	
Web Service Choreography Engine			
Web Service Process Mediator			
Web Service Grounding			
WS Profiling			

Table 2 Dependencies of the components with the components of the *Querying and reasoning* dimension

	Ontology Editor	Ontology Browser	Ontology Evaluator	Ontology Learner	Ontology Matcher
Information Directory Manager					X
Ontology Repository					
Data Repository					
Alignment Repository					
Metadata Registry					
Query Answering					
Semantic Query Processor					
Semantic Query Editor					
Ontology Editor		X			
Ontology Browser					
Ontology Evaluator					
Ontology Learner					

Ontology Matcher	X				
Ontology Discovery & Ranking					
Ontology Localization & Profiling					
Ontology Adaptation Operators					
Ontology View Customization					
Ontology Evolution Manager					
Ontology Evolution Visualizer					
Ontology Versioner					
Instance Editor					
Manual Annotation					
Automatic Annotation					
Ontology Populator					
Web Service Discoverer					
Web Service Selector					
Web Service Composer					
Web Service Choreography Engine					
Web Service Process Mediator					X
Web Service Grounding					
Web Service Profiling					

Table 3 Dependencies of the components with the components of the Ontology Engineering dimension

	Ontology Discovery & Ranking	Ontology Localization & Profiling	Ontology Adaptation Operators	Ontology View Customization
Information Directory Manager				
Ontology Repository				
Data Repository				
Alignment Repository				
Metadata Registry				
Query Answering				
Semantic Query Processor				
Semantic Query Editor				
Ontology Editor				
Ontology Browser				X
Ontology Evaluator				
Ontology Learner				
Ontology Matcher				X
Ontology Discovery & Ranking				
Ontology Localization & Profiling				
Ontology Adaptation Operators	X	X		
Ontology View			X	

Customization				
Ontology Evolution Manager				
Ontology Evolution Visualizer				
Ontology Versioner				
Instance Editor				
Manual Annotation				
Automatic Annotation				
Ontology Populator				
Web Service Registry				
Web Service Discoverer				
Web Service Selector				
Web Service Composer				
Web Service Choreography Engine				
Web Service Process Mediator				
Web Service Grounding				
Web Service Profiling				

Table 4 Dependencies of the components with the components of the *Ontology customization* dimension

	Ontology Evolution Manager	Ontology Evolution Visualizer	Ontology Versioner
Information Directory Manager			
Ontology Repository			
Data Repository			
Alignment Repository			
Metadata Registry			
Query Answering			
Semantic Query Processor			
Semantic Query Editor			
Ontology Editor			
Ontology Browser			
Ontology Evaluator			
Ontology Learner			
Ontology Matcher			
Ontology Discovery & Ranking			
Ontology Localization & Profiling			
Ontology Adaptation Operators			
Ontology View Customization			
Ontology Evolution Manager		X	X
Ontology Evolution Visualizer			X
Ontology Versioner			
Instance Editor			
Manual Annotation			
Automatic Annotation			
Ontology Populator			
Web Service Discoverer			
Web Service Selector			
Web Service Composer			

Web Service Choreography Engine			
Web Service Process Mediator			
Web Service Grounding			
Web Service Profiling			

Table 5 Dependencies of the components with the components of the *Ontology evolution* dimension

	Instance Editor	Manual Annotation	Automatic Annotation	Ontology Populator
Information Directory Manager				
Ontology Repository				
Data Repository				
Alignment Repository				
Metadata Registry				
Query Answering				
Semantic Query Processor				
Semantic Query Editor				
Ontology Editor				
Ontology Browser				
Ontology Evaluator				
Ontology Learner				
Ontology Matcher				
Ontology Discovery & Ranking				
Ontology Localization & Profiling				
Ontology Adaptation Operators				
Ontology View Customization				
Ontology Evolution Manager				
Ontology Evolution Visualizer				
Ontology Versioner				
Instance Editor				
Manual Annotation				
Automatic Annotation		X		
Ontology Populator		X		
Web Service Registry				
Web Service Discoverer				
Web Service Selector				
Web Service Composer				
Web Service Choreography Engine				
Web Service Process Mediator				
Web Service Grounding				
Web Service Profiling				

Table 6 Dependencies of the components with the components of the *Ontology instance generation* dimension

	Web Service Registry	Web Service Discoverer	Web Service Selector	Web Service Composer	Web Service Choreography Engine	Web Service Process Mediator	Web Service Grounding	Web Service Profiling
Information Directory Manager								
Ontology Repository								
Data								

Repository								
Alignment Repository								
Metadata Registry								
Query Answering								
Semantic Query Processor								
Semantic Query Editor								
Ontology Editor								
Ontology Browser								
Ontology Evaluator								
Ontology Learner								
Ontology Matcher								
Ontology Discovery & Ranking								
Ontology Localization & Profiling								
Ontology Adaptation Operators								
Ontology View Customization								
Ontology Evolution Manager								
Ontology Evolution Visualizer								
Ontology Versioner								
Instance Editor								
Manual Annotation								
Automatic Annotation								
Ontology Populator								
Web Service Registry								
Web Service Discoverer	X							
Web Service	X	X						

Selector								
Web Service Composer	X	X	X					
Web Service Choreography Engine	X					X		
Web Service Process Mediator	X							
Web Service Grounding	X					X		
Web Service Profiling	X							

Table 7 Dependencies of the components with the components of the *Semantic Web Services* dimension

Appendix II Dependencies between the use cases and the Semantic Web Framework

This appendix includes a table that shows the dependencies of the use cases and all the components of the Semantic Web Framework.

	UC 1	UC 2	UC 3	UC 4	UC 5	UC 6	UC 7	UC 8
Information Directory Manager								
Ontology Repository	X	X	X	X	X	X	X	X
Data Repository	X	X	X	X	X	X	X	X
Alignment Repository	X	X	X		X	X		X
Metadata Registry		X						
Query Answering	X	X	X	X	X		X	X
Semantic Query Processor	X	X	X	X	X		X	X
Semantic Query Editor			X		X		X	X
Ontology Editor	X		X	X	X		X	X
Ontology Browser	X		X	X	X		X	X
Ontology Evaluator								X
Ontology Learner			X		X	X	X	X
Ontology Matcher	X	X	X		X	X		X
Ontology Discovery & Ranking				X				
Ontology Localization & Profiling		X		X	X			
Ontology Adaptation Operators				X				
Ontology View Customization	X		X	X	X		X	X
Ontology Evolution Manager	X			X	X		X	X
Ontology Evolution Visualizer	X			X	X		X	X
Ontology Versioner	X		X	X	X		X	X
Instance Editor				X				
Manual Annotation	X		X					
Automatic Annotation	X		X			X	X	
Ontology Populator	X		X			X		
Web Service Registry		X						
Web Service Discoverer		X						
Web Service Selector		X						
Web Service Composer		X						
Web Service Choreography Engine								
Web Service Process Mediator								
Web Service Grounding								
Web Service Profiling								

Table 8 Dependencies of the use cases with the components of the Semantic Web Framework

Appendix III Implementations of the Semantic Web Framework components

This appendix includes the implementations found for the SWF components described in Chapter 4.

6.1 Data and Metadata Management

6.1.1 Information directory manager component

Name:	**Aduna Metadata Server**
URL:	http://www.aduna-software.com/solutions/metadata_server/overview.view
Type of implementation:	Application
Multiple components:	Yes. It is also a Data and Metadata Repository.
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Accessing both data and metadata. User interface and Web interface
Metadata Registry used: Required/Optional: Type of interface:	Metadata Server Required Programming Interface

Name:	**Alvis**
URL:	http://www.alvis.info/alvis/
Type of implementation:	Application
Multiple components:	Yes. It is also a data and metadata repository and also contains a query answering component.
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Storing and accessing data and metadata Web interface
Data Repository used: Required/Optional: Type of interface:	Zebra Required Programming interface

Name:	**Beagle++**
URL:	http://beagle.l3s.de
Type of implementation:	Application
Multiple components:	Yes. It is a data and metadata repository and contains a semantic query processor.
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Store and access metadata. User interface
Data Repository used: Required/Optional: Interface: Type of interface:	Lucene Required Lucene API Programming interface
Metadata Registry used: Required/Optional: Interface: Type of interface:	Sesame Required Sesame API Programming interface

Name:	**Gnowsis**
URL:	http://www.gnowsis.org/

Type of implementation:	Application
Multiple components:	Yes. It is a data, metadata,ontology store and also a query answering component.
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Store data and metadata and representational ontologies. User interface
Data Repository used: Required/Optional: Interface: Type of interface:	Aperture Required Java Api Programming Interface
Metadata Registry used: Required/Optional: Interface: Type of interface:	Sesame Repository Required Java Api Programming Interface

Name:	**Haystack**
URL:	http://haystack.csail.mit.edu/home.html
Type of implementation:	Application
Multiple components:	Yes. It is a data and metadata storage.
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Store metadata and browse through it. User interface

Name:	**OWLIM**
URL:	http://www.ontotext.com/owlim/
Type of implementation:	Application
Multiple components:	Yes. Storage, querying, reasoning over metadata
Representation formalisms:	RDF/S, OWL
Functionalities provided: Type of interface:	Storage, querying, reasoning over metadata. SAIL over Sesame(sync) User interface
Metadata Registry used: Required/Optional: Interface: Type of interface:	Sesame Required Sesame API Programming Interface

6.1.2 Ontology repository component

The implementations of the *ontology repository* component that we consider are those specialized in semantic web resources (i.e., ontologies, RDF schemas, etc.). They can be classified in 2 different types: centralized or decentralized applications.

Centralized ontology repositories

Name:	**Jena**
URL:	http://jena.sourceforge.net/
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Reading and writing RDF in RDF/XML, N3 and N-Triples In-memory and persistent storage SPARQL query engine Inference support RDF/OWL support API

Name:	**KAON2**
URL:	http://kaon2.semanticweb.org/
Type of implementation:	Application
Multiple components:	Yes
Representation formalisms:	OWL-DL and F-Logic
Functionalities provided:	Management of OWL-DL, SWRL, and F-Logic ontologies, Inference engine for answering conjunctive queries (expressed using SPARQL syntax), DIG interface Extraction of ontology instances from relational databases Supports remote access through RMI
Type of interface:	API

Name:	**Sesame**
URL:	http://www.openrdf.org/
Type of implementation:	Application
Multiple components:	Yes
Representation formalisms:	RDF
Functionalities provided:	RDF Schema querying RDF Schema storing RDF Schema inferencing Supports both local and remote access(through HTTP or RMI) Supports several query languages
Type of interface:	API

Name:	**Ontology Server**
URL:	http://www.starlab.vub.ac.be/research/dogma/OntologyServer.htm
Type of implementation:	Application
Multiple components:	No
Functionalities provided:	Get ontologies/concepts Add ontologies/concepts
Type of interface:	API

Name:	**RDF Server**
URL:	http://semanticweb.gr/rdftp/
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF
Functionalities provided:	Add ontologies Query Update
Type of interface:	API

Name:	**Knowledge zone**
URL:	http://smi-protege.stanford.edu:8080/KnowledgeZone/
Type of implementation:	Web portal
Multiple components:	No
Representation formalisms:	OWL
Functionalities provided:	Search by Keyword Browse by Topic Submit new Ontology and related Metadata Rating System

Type of interface:	Web interface

Name:	**Onthology**
URL:	http://www.onthology.org/
Type of implementation:	Web portal
Multiple components:	Yes
Representation formalisms:	OWL
Functionalities provided:	Search by some metadata Browse by some metadata Submit new Ontology and related Metadata Export Repository Rating System
Type of interface:	Web interface

Name:	**OntoSelect**
URL:	http://olp.dfki.de/ontoselect/
Type of implementation:	Web portal
Multiple components:	No
Functionalities provided:	Search by keywords Browse ontologies Document-based automatic ontology selection Multilingual label support Submit new ontology
Type of interface:	Web interface

Name:	**DAML Ontology Library**
URL:	http://www.daml.org/ontologies/
Type of implementation:	Web portal
Multiple components:	No
Functionalities provided:	Browse ontologies Submit new ontology
Type of interface:	Web interface

Name:	**SchemaWeb**
URL:	http://www.schemaweb.info/
Type of implementation:	Web portal
Multiple components:	No
Representation formalisms:	RDF
Functionalities provided:	Browse ontologies Search by keywords Query triples Submit new ontology
Type of interface:	Web interface

Name:	**ONTOSEARCH2**
URL:	http://www.ontosearch.org/
Type of implementation:	Web portal
Multiple components:	No
Representation formalisms:	OWL
Functionalities provided:	Search by keywords SPARQL query support Submit new ontology
Type of interface:	Web interface

Name:	**ProtégéOntologiesLibrary**
URL:	http://protege.cim3.net/cgi-bin/wiki.pl?ProtegeOntologiesLibrary
Type of implementation:	Web portal
Multiple components:	No
Functionalities provided: Type of interface:	Browse ontologies Submit new ontology Web interface

Name:	**Ontolingua**
URL:	http://www-ksl-svc.stanford.edu:5915/&service=FRAME-EDITOR
Type of implementation:	Web portal
Multiple components:	No
Functionalities provided: Type of interface:	Browse ontologies Search by keywords Submit new ontology Web interface

Decentralized ontology repositories

Name:	**OntStore**
URL:	http://ui.sav.sk/parcom/index.html
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	OWL
Functionalities provided: Type of interface:	Query RDF triples Add ontology API

Name:	**RDFPeer**
URL:	http://www.isi.edu/index.php
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Query RDF triples Add ontology API

Name:	**RDF2GO**
URL:	http://wiki.ontoworld.org/wiki/RDF2Go
Type of implementation:	Program Library
Multiple components:	Yes. Also offers querying over metadata.
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Storage and querying over metadata Programming interface

6.1.3 Data repository component

Name:	**DSpace**
URL:	http://dspace.org/
Type of implementation:	Application
Multiple components:	No.
Functionalities provided: Type of interface:	Storing data. Web-based user interface

Name:	**Lucene**
URL:	http://lucene.apache.org
Type of implementation:	Application
Multiple components:	Yes. It also has a querying component.
Functionalities provided: Type of interface:	Storage and querying of textual data. Programming and user interface

Name:	**Zebra**
URL:	http://www.indexdata.dk/zebra/
Type of implementation:	Application
Multiple components:	Yes. It also allows querying for data.
Functionalities provided: Type of interface:	Storage and querying of the data User interface

6.1.4 Alignment repository component

Name:	**COMA++**
URL:	http://dbs.uni-leipzig.de/Research/coma.html
Type of implementation:	Application and Web Application
Multiple components:	Yes. It is also an ontology matching tool.
Representation formalisms:	OWL, XSD
Functionalities provided: Type of interface:	*Alignment repository*, alignment tool User interface

Name:	**Alignment API and Alignment Server**
URL:	http://alignapi.gforge.inria.fr/
Type of implementation:	Application and API
Multiple components:	Yes. It is also an ontology alignment tool.
Representation formalisms:	RDF, XML
Functionalities provided: Type of interface:	*Alignment repository*, alignment tool User, programming interface and as a web service.

6.1.5 Metadata registry component

Name:	**3store**
URL:	http://inanna.ecs.soton.ac.uk/3store/
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Store RDF triples RDQL query interface via Web or directly with the C library

Name:	**AllegroGraph**
URL:	http://www.franz.com/products/allegrograph/
Type of implementation:	Application
Multiple components:	Yes. It is also an ontology repository and it has a query & reasoning tool.
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Metadata repository Programming interface

Name:	**Boca**
URL:	http://sourceforge.net/project/showfiles.php?group_id=181986&pack

	age_id=210881
Type of implementation:	Application
Multiple components:	No.
Representation formalisms:	RDF
Functionalities provided: Type of interface:	RDF store User interface

Name:	**Brahms**
URL:	http://lsdis.cs.uga.edu/projects/semdis/brahms/
Type of implementation:	Application
Multiple components:	No.
Representation formalisms:	RDF/S
Functionalities provided: Type of interface:	Metadata storage User interface

Name:	**Hawk**
URL:	http://swat.cse.lehigh.edu/projects/index.html#hawk
Type of implementation:	Application
Multiple components:	No.
Representation formalisms:	OWL
Functionalities provided: Type of interface:	Storing OWL data Programming interface

Name:	**The open metadata registry (prototype 1)**
URL:	http://www.dlib.org/dlib/may02/wagner/05wagner.html
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Search RDF schemas Browse RDF schemas Register RDF schema Web Interface

Name:	**The open metadata registry (prototype 2)**
URL:	http://www.dlib.org/dlib/may02/wagner/05wagner.html
Type of implementation:	Java Servlet Application
Multiple components:	No
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Search RDF schemas Browse RDF schemas Register RDF schema Web Interface

Name:	**The open metadata registry (prototype 3)**
URL:	http://www.dlib.org/dlib/may02/wagner/05wagner.html
Type of implementation:	Java Servlet Application
Multiple components:	No
Representation formalisms:	RDF
Functionalities provided:	Search RDF schemas Browse RDF schemas Register RDF schema Login Import

Type of interface:	Web Interface

Name:	**OASIS ebXML Registry**
URL:	http://ebxmlrr.sourceforge.net/
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	XML
Functionalities provided:	Role Based Access Control Cataloging of XML Content HTTP Interface to Registry Content-based Event Notification Registry Managed Version Control Parameterized Stored Queries
Type of interface:	Web Interface, Java User Interface

Name:	**Oyster**
URL:	http://oyster.ontoware.org
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF
Functionalities provided:	Search ontologies by metadata Register ontology metadata Modify ontology metadata Import/export metadata Extract Metadata from ontology files
Type of interface:	Java Graphical User Interface

Name:	**Oyster2**
URL:	http://oyster2.ontoware.org
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	OWL
Functionalities provided:	Search ontologies by metadata Register ontology metadata Modify ontology metadata Import/export metadata Extract Metadata from ontology files
Type of interface:	API, Java Graphical User Interface

Name:	**Kowari**
URL:	http://www.kowari.org/
Type of implementation:	Application
Multiple components:	Yes. It also has a querying component
Representation formalisms:	RDF, OWL
Functionalities provided:	Storage, retrieval and analysis of metadata
Type of interface:	User interface

Name:	**RDFGateway**
URL:	http://www.intellidimension.com/
Type of implementation:	Application
Multiple components:	Yes. It also has a querying component.
Representation formalisms:	RDF,OWL
Functionalities provided:	Store and query metadata
Type of interface:	Programming and user interface

Name:	**RDF2GO**
URL:	http://wiki.ontoworld.org/wiki/RDF2Go
Type of implementation:	Program Library
Multiple components:	Yes. Also offers querying over metadata.
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Storage and querying over metadata Programming interface

Name:	**RDFStore**
URL:	http://rdfstore.sourceforge.net/
Type of implementation:	Application
Multiple components:	No.
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Storage of metadata User and programming interface

Name:	**SemWeb**
URL:	http://razor.occams.info/code/semweb/
Type of implementation:	Library
Multiple components:	Yes. It also has a querying component.
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Storing and querying over metadata. Programming interface

Name:	**YARS**
URL:	http://sw.deri.org/2004/06/yars/
Type of implementation:	Application
Multiple components:	Yes. It also has a querying module
Representation formalisms:	RDF, N3
Functionalities provided: Type of interface:	Storing and querying metadata Programming

6.2 Querying and Reasoning

6.2.1 Query answering component

Name:	**AJAX Client for SPARQL**
URL:	http://xmlarmyknife.org/docs/rdf/sparql/ajax.html
Type of implementation:	AJAX client
Multiple components:	No.
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Query RDF Programming interface

Name:	**Bor**
URL:	http://www.ontotext.com/bor/
Type of implementation:	Library
Multiple components:	No.
Representation formalisms:	DAML+OIL
Functionalities provided: Type of interface:	Reasoner Programming interface

Name:	**Corese**

URL:	http://www.inria.fr/acacia/corese
Type of implementation:	Application
Multiple components:	Yes. It is a semantic web search engine.
Representation formalisms:	RDF(S), OWL Lite, RDF Rules, SPARQL (and its XML Result Format) The underlying formalism is Conceptual Graph - Written in Java 1.5
Functionalities provided:	Query consistency Query rewriting Checking if a query pattern matches an ontology. Selection of ontology concepts that satisfy query constraints Extraction of ontology parts relevant to the particular query Description of ontology concepts Restriction of results number Identifying sources that contain information relevant to the query Request information from the identified sources Supports the user in formulating a query Provides a user-friendly query language (SPARQL) Provides a user-friendly representation of results. In addition: SPARQL query processing with RDFS entailment, query RDF Schema, approximate search with similarity function, aggregation (group, count, sum, etc.), expression in select, path. Corese RDF Inference Rule Language (forward chaining) RDFS type inference (classify resources according to property signature)
Type of interface:	API, JSP Semantic Tag Library for building Semantic Web Servers (library name: Sewese) Web Service

Name:	**KAONP2P**
URL:	http://kaonp2p.ontoware.org
Type of implementation:	Application
Multiple components:	Yes
Representation formalisms:	OWL
Functionalities provided:	Expressive Reasoning ability Supports mappings between various domain ontologies Dynamic User Interface
Type of interface:	Java Graphical User Interface

Name:	**KAONWeb**
URL:	http://kaonweb.ontoware.org
Type of implementation:	Application
Multiple components:	Yes
Representation formalisms:	OWL
Functionalities provided:	Expressive Reasoning ability Supports mappings between various domain ontologies
Type of interface:	Web Interface

Name:	**Oyster2**
URL:	http://oyster2.ontoware.org
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	OWL
Functionalities provided:	Search ontologies by metadata Register ontology metadata

Type of interface:	Modify ontology metadata Import/export metadata Extract Metadata from ontology files API, Java Graphical User Interface

6.2.2 Semantic query processor component

Name:	**AeroText**
URL:	http://www.lockheedmartin.com/wms/findPage.do?dsp=fec&ci=11255&sc=400
Type of implementation:	Application
Multiple components:	Yes. It is also a query answering component.
Functionalities provided:	Answering to queries related to concepts, not only documents, over a database.

Name:	**Sesame**
URL:	http://www.openrdf.org/
Type of implementation:	Application
Multiple components:	Yes
Representation formalisms:	RDF
Functionalities provided: Type of interface:	RDF Schema querying RDF Schema storing RDF Schema inferencing Supports both local and remote access(through HTTP or RMI) Supports several query languages API

6.2.3 Semantic query editor component

Name:	**Ontogator**
URL:	http://www.seco.tkk.fi/projects/semweb/dist.php
Type of implementation:	Application
Multiple components:	Yes. It allows the user to create a query, to manually/visually modify it, and of course it answers the query.
Representation formalisms:	RDF/XML
Functionalities provided: Type of interface:	Query, edit query in RDF. User interface

Name:	**SemSearch**
URL:	http://semanticweb.kmi.open.ac.uk:8080/ksw/pages/semantic_searching.jsp
Type of implementation:	Application
Multiple components:	Yes. It includes a query answering and a query editor.
Functionalities provided: Type of interface:	Writing a query and answering it. User interface
Semantic query processor used: Required/Optional: Interface: Type of interface:	Sesame Required Sesame API Programming Interface

6.3 Ontology Engineering

6.3.1 Ontology editor component

The *Ontology editor* component implementations (ontology editors from now on) can be classified in 2 different types. The most common one is that of applications whose main goal is ontology edition and the less frequent are ontology edition plugins of larger applications.

Ontology editors that only deal with one specific ontology have not been considered.

Name:	**Altova Semanticworks**
URL:	http://www.altova.com/products/semanticworks/semantic_web_rdf_owl_editor.html
Type of implementation:	Application
Multiple components:	Yes. It is an ontology engineering environment
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Edit ontologies User interface
Ontology repository used: Required/Optional:	Local filesystem Required

Name:	**DODDLE: Domain ontology rapid development environment**
URL:	http://www.yamaguchi.comp.ae.keio.ac.jp/mmm/doddle/
Type of implementation:	Application
Multiple components:	Yes. It is an ontology learner and an ontology editor
Representation formalisms:	OWL Lite
Functionalities provided: Type of interface:	Edit ontologies User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Required Jena API (PI)

Name:	**DOE**
URL:	http://homepages.cwi.nl/~troncy/DOE/
Type of implementation:	Application
Multiple components:	Yes. It is an ontology browser
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Edit ontologies User interface
Ontology repository used: Required/Optional:	Local filesystem Required

Name:	**DOME**
URL:	http://dome.sourceforge.net/
Type of implementation:	Application
Multiple components:	Yes. It is an ontology engineering environment
Representation formalisms:	WSML
Functionalities provided: Type of interface:	Edit ontologies User interface
Ontology repository used: Required/Optional:	Local filesystem Required

Name:	**Fenfire**

URL:	http://fenfire.org/apps/editing.html
Type of implementation:	Application
Multiple components:	Yes. It is an ontology browser
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Edit ontologies User interface
Ontology repository used: Required/Optional:	Local filesystem Required

Name:	**Graphl**
URL:	http://home.subnet.at/flo/mv/graphl/
Type of implementation:	Application
Multiple components:	Yes. It is an ontology browser
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Edit ontologies User interface
Ontology repository used: Required/Optional:	Local filesystem Required
Ontology repository used: Required/Optional:	Web server Optional

Name:	**GrOWL**
URL:	http://ecoinformatics.uvm.edu/technologies/growl-knowledge-modeler.html
Type of implementation:	Application
Multiple components:	Yes. It is an ontology browser
Representation formalisms:	OWL
Functionalities provided: Type of interface:	Edit ontologies User interface
Ontology repository used: Required/Optional:	Local filesystem Required

Name:	**IBM Integrated Ontology Development Toolkit**
URL:	http://www.alphaworks.ibm.com/tech/semanticstk
Type of implementation:	Application
Multiple components:	Yes. It is an ontology engineering environment
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Edit ontologies Web interface
Semantic query processor used: Required/Optional: Interface: Type of interface:	Minerva Required EMF Ontology Definition Metamodel (EODM) API Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Minerva Required EMF Ontology Definition Metamodel (EODM) API Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Required EMF Ontology Definition Metamodel (EODM) API Programming interface

Name:	**Infered**

URL:	http://www.intellidimension.com/
Type of implementation:	Application
Multiple components:	Yes. It is an ontology engineering environment
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Edit ontologies Web interface
Ontology repository used: Required/Optional:	RDF Gateway Required
Ontology repository used: Required/Optional:	Local filesystem Required

Name:	**IsaViz**
URL:	http://www.w3.org/2001/11/IsaViz/
Type of implementation:	Application
Multiple components:	Yes. It is an ontology browser
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Edit ontologies User interface
Semantic query processor used: Required/Optional: Interface: Type of interface:	Jena Required Jena API Programming interface
Semantic query processor used: Required/Optional: Interface: Type of interface:	Sesame Optional Sesame API Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Required Jena API Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Web server Optional Jena API Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Sesame Optional Sesame API Programming interface

Name:	**KAON OI Modeler**
URL:	http://kaon.semanticweb.org/
Type of implementation:	Application
Multiple components:	Yes. It is an ontology engineering environment
Representation formalisms:	RDF(S)
Functionalities provided: Type of interface:	Edit ontologies User interface
Ontology repository used: Required/Optional:	Local filesystem Required

Name:	**Linkfactory**
URL:	http://www.landcglobal.com/pages/linkfactory.php
Type of implementation:	Application
Multiple components:	Yes. It is an ontology engineering environment
Representation formalisms:	RDF(S), OWL

Functionalities provided: Type of interface:	Edit ontologies User interface
Ontology repository used: Required/Optional:	Local filesystem Required
Ontology repository used: Required/Optional:	Oracle, Sybase and SQLServer Optional

Name:	**Ontotrack**
URL:	http://www.informatik.uni-ulm.de/ki/ontotrack/
Type of implementation:	Application
Multiple components:	Yes. It is an ontology browser
Representation formalisms:	OWL
Functionalities provided: Type of interface:	Edit ontologies User interface
Semantic query processor used: Required/Optional: Interface: Type of interface:	RACER Required RACER API Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Required Jena API Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Web server Optional Jena API Programming interface

Name:	**Powl**
URL:	http://aksw.informatik.uni-leipzig.de/Projects/Powl
Type of implementation:	Application
Multiple components:	Yes. It is an ontology engineering environment
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Edit ontologies User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Powl store Required RAP – RDF API for PHP Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Powl store Required Powl RDFS and OWL API for PHP Programming interface

Name:	**Protégé**
URL:	http://protege.stanford.edu/
Type of implementation:	Application
Multiple components:	Yes. It is an ontology engineering environment
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Edit ontologies User interface
Ontology browser used: Required/Optional: Interface:	ezOWL, FCAView, GrOWL Tab Widget, Jambalaya, OntoViz, OWLViz, TGVizTab Optional Protégé plugin APIs

Type of interface:	Programming interface
Ontology repository used: Required/Optional:	Local filesystem Required
Ontology repository used: Required/Optional:	Sesame Optional
Ontology repository used: Required/Optional:	JDBC Backend Optional

Name:	**Rhodonite**
URL:	http://rhodonite.angelite.nl/
Type of implementation:	Application
Multiple components:	Yes. It is an ontology browser
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Edit ontologies User interface
Ontology repository used: Required/Optional:	Local filesystem Required
Ontology repository used: Required/Optional:	Web server Optional

Name:	**SemTalk**
Type of implementation:	MS Visio plugin
Multiple components:	Yes. It is an ontology engineering environment
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Edit ontologies User interface
Semantic query processor used: Required/Optional:	Pellet Optional
Ontology repository used: Required/Optional:	Local filesystem Required

Name:	**SWOOP**
URL:	http://www.mindswap.org/2004/SWOOP/
Type of implementation:	Application
Multiple components:	Yes. It is an ontology engineering environment
Representation formalisms:	OWL
Functionalities provided: Type of interface:	Edit ontologies User interface
Semantic query processor used: Required/Optional: Interface: Type of interface:	Pellet Required Proprietary plugin based system Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Required Manchester OWL API Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Web server Optional Manchester OWL API Programming interface

Name:	**Topbraid composer**
URL:	http://www.topbraidcomposer.com/
Type of implementation:	Application

Multiple components:	Yes. It is an ontology engineering environment
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Edit ontologies User interface
Semantic query processor used: Required/Optional: Interface: Type of interface:	Pellet Required DIG interface Programming interface
Semantic query processor used: Required/Optional: Interface: Type of interface:	Any DIG-based reasoner Optional DIG interface Programming interface
Ontology repository used: Required/Optional:	Local filesystem Required
Ontology repository used: Required/Optional:	Web server Optional

Name:	**WebODE**
URL:	http://webode.dia.fi.upm.es/WebODEWeb/index.html
Type of implementation:	Application
Multiple components:	Yes. It is an ontology engineering environment
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Edit ontologies User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	WebODE server Required WebODE API Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Optional WebODE API Programming interface

Name:	**DogmaModeler**
Type of implementation:	Application
Multiple components:	Yes. It is an ontology engineering environment
Functionalities provided:	DogmaModeler is software tool for modeling and engineering ontologies. It supports among other things: (1) the development, browsing, and management of domain and application axiomatizations, and axiomatization libraries; (2) the modeling of application axiomatizations using the ORM graphical notation, and the automatic generation of the corresponding ORM-ML; (3) Maps ORM diagrams into DIG and uses Racer for reasoning; (3) the verbalization of application axiomatizations into pseudo natural language (supporting flexible verbalization templates for English, Dutch, Arabic, and Russian, for example) that allows non-experts to check, validate, or build axiomatizations; (4) the automatic composition of axiomatization modules, through a well-defined composition operator; (5) the validation of the syntax and semantics of application axiomatizations; (6) the incorporating of linguistic resources in ontology engineering; (7) a simple approach of multilingual lexicalization of ontologies; (8) the automatic mapping of ORM schemes into X-Forms and HTML-Forms; etc.
Type of interface:	User interface

Name:	**ICOM**
URL:	http://www.inf.unibz.it/~franconi/icom/
Type of implementation:	Application
Representation formalisms:	OWL-DL
Functionalities provided:	The ontology language supported by ICOM can express: - standard UML or Entity-Relationship models, extended with definitions attached to entities and relations by means of view expressions over other entities and relationships in the ontology; - rich class of (interschema) integrity constraints, as inclusion and equivalence dependencies between view expressions involving entities and relationships possibly belonging to different schemas. The expressive power of ICOM is equivalent to OWL-DL without nominals; ICOM has an export function to OWL-DL. ICOM reasons with (multiple) diagrams by encoding them in a single description logic knowledge base, and shows the result of any deductions such as inferred links, new stricter constraints, and inconsistent entities or relationships.
Type of interface:	User interface

6.3.2 Ontology browser component

The *Ontology browser* component implementations (ontology browsers from now on) can be classified into 3 different types: applications whose main goal is ontology browsing, ontology browsing plugins of larger applications, and ontology development tools that provide ontology browsing functionalities. In this section, we have not considered implementations of ontology development tools, as they are included in the implementations of the *Ontology editor* component. Furthermore, ontology browsers that only deal with one specific ontology have not been considered.

Ontology browsers

Name:	**Brownsauce**
URL:	http://brownsauce.sourceforge.net/
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Browse RDF Web interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Web server Required Jena API Programming interface

Name:	**BrowseRDF**
URL:	https://launchpad.net/browserdf
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Browse RDF Web interface
Ontology repository used: Required/Optional: Interface:	Web server Required ActiveRDF API

Type of interface:	Programming interface

Name:	**Drive RDF Browser**
URL:	http://www.driverdf.org/articles/rdfbrowser.html
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Browse ontologies User interface
Ontology repository used: Required/Optional:	Local filesystem Required
Ontology repository used: Required/Optional:	Web server Optional

Name:	**Disco**
URL:	http://sites.wiwiss.fu-berlin.de/suhl/bizer/ng4j/disco/
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Browse ontologies Web interface
Ontology repository used: Required/Optional:	Web server Required

Name:	**Horus**
URL:	http://sites.wiwiss.fu-berlin.de/suhl/bizer/rdfapi/tutorial/horus/index.htm
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Browse RDF Web interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Required RAP – RDF API for PHP Programming interface

Name:	**Longwell**
URL:	http://simile.mit.edu/wiki/Longwell
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Browse ontologies Web interface
Ontology repository used: Required/Optional:	Local filesystem Required

Name:	**OINK**
URL:	http://wiki.nrcc.noklab.com/SwapMe/OINK
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Browse ontologies User interface

Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Required Wilbur API Programming interface

Name:	**RDF Gravity**
URL:	http://semweb.salzburgresearch.at/apps/rdf-gravity/index.html
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Browse ontologies User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Required Jena API Programming interface

Name:	**Tabulator**
URL:	http://www.w3.org/2005/ajar/tab
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Browse ontologies User interface
Ontology repository used: Required/Optional:	Web server Required

Name:	**Welkin**
URL:	http://simile.mit.edu/welkin/
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Browse ontologies User interface
Ontology repository used: Required/Optional:	Local filesystem Required

Ontology browsing plugins

Name:	**Jambalaya**
URL:	http://www.thechiselgroup.org/jambalaya
Type of implementation:	Protégé plugin
Multiple components:	No
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Browse ontologies User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Protégé Required Protégé plugin APIs Programming interface

Name:	**Ontosphere 3D**
URL:	http://ontosphere3d.sourceforge.net/
Type of implementation:	Protégé plugin
Multiple components:	No

Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Browse ontologies User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Protégé Required Protégé plugin APIs Programming interface

Name:	**OntoViz**
URL:	http://protege.cim3.net/cgi-bin/wiki.pl?OntoViz
Type of implementation:	Protégé plugin
Multiple components:	No
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Browse ontologies User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Protégé Required Protégé plugin APIs Programming interface

Name:	**OWLViz**
URL:	http://www.co-ode.org/downloads/owlviz/co-ode-index.php
Type of implementation:	Protégé plugin
Multiple components:	No
Representation formalisms:	OWL
Functionalities provided: Type of interface:	Browse ontologies User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Protégé Required Protégé plugin APIs Programming interface

Name:	**TGVizTab**
URL:	http://www.ecs.soton.ac.uk/~ha/TGVizTab/TGVizTab.htm
Type of implementation:	Protégé plugin
Multiple components:	No
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Browse ontologies User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Protégé Required Protégé plugin APIs Programming interface

6.3.3 Ontology evaluator component

The *Ontology evaluator* component implementations (ontology evaluators from now on) are either applications (standalone or web) or program libraries. In the case of program libraries, usually one small application has been developed using the program library to allow users to evaluate ontologies.

The implementations of the Semantic query processor component can be used to evaluate ontologies using their subsumption, classification and consistency checking

functionalities. In this section we have not considered these implementations, as they are included in the implementations of the *Semantic query processor* component.

Name:	**ARP: Another RDF Parser**
URL:	http://www.hpl.hp.com/personal/jjc/arp/
Type of implementation:	Program library
Multiple components:	No
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Ontology evaluation Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Required Jena API Programming Interface

Name:	**CLEANONTO**
URL:	http://www.csd.abdn.ac.uk/~qreul/software.html
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	OWL
Functionalities provided: Type of interface:	Ontology evaluation User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Required Jena API Programming Interface
Data repository used: Required/Optional: Interface: Type of interface:	WordNet Required JWNL API Programming interface

Name:	**ConsVISor**
URL:	http://www.vistology.com/consvisor/
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	OWL, RDF, DAML
Functionalities provided: Type of interface:	Ontology evaluation User interface
Ontology repository used: Required/Optional:	Web server Required

Name:	**Eyeball**
URL:	http://jena.sourceforge.net/Eyeball
Type of implementation:	Program library
Multiple components:	No
Representation formalisms:	RDF, OWL
Functionalities provided: Type of interface:	Ontology evaluation Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Required Jena API Programming Interface

Name:	**ODEval**

URL:	http://minsky.dia.fi.upm.es/odeval
Type of implementation:	Program library
Multiple components:	No
Representation formalisms:	RDF(S), DAML+OIL, OWL
Functionalities provided: Type of interface:	Ontology evaluation Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	WebODE server Required WebODE API Programming Interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Optional WebODE API Programming Interface

Name:	**OWL API**
URL:	http://owlapi.sourceforge.net/OWLAPI/
Type of implementation:	Program library
Multiple components:	Yes, it is an Ontology repository
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Ontology evaluation Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Required OWL API Programming Interface

Name:	**Semantic Web/RDF Library for C#/.NET**
URL:	http://rdfabout.com/demo/validator/
Type of implementation:	Program library
Multiple components:	Yes, it is an Ontology repository
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Ontology evaluation Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Required SemWeb C# RDF Programming Interface

Name:	**Validating RDF Parser**
URL:	http://139.91.183.30:9090/RDF/VRP/
Type of implementation:	Program library
Multiple components:	Yes, it is an Ontology repository
Representation formalisms:	RDF(S)
Functionalities provided: Type of interface:	Ontology evaluation Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Required VRP API Programming Interface

6.3.4 Ontology learner component

The *Ontology learner* component implementations (ontology learners from now on) are either standalone applications or program libraries. Some of these ontology learners are part of ontology engineering environments that provide other functionalities.

Name:	**DODDLE: Domain ontology rapid development environment**
URL:	http://www.yamaguchi.comp.ae.keio.ac.jp/mmm/doddle/
Type of implementation:	Application
Multiple components:	Yes. It is an Ontology editor and an Ontology browser
Representation formalisms:	OWL Lite
Functionalities provided: Type of interface:	Ontology learning User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Required Jena API Programming Interface
Data repository used: Required/Optional:	Local filesystem Required

Name:	**KEA: Keyphrases Extraction Algorithm**
URL:	http://www.nzdl.org/Kea/
Type of implementation:	Program library
Multiple components:	No
Representation formalisms:	SKOS
Functionalities provided: Type of interface:	Ontology learning Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Required Jena API Programming Interface
Data repository used: Required/Optional:	Local filesystem Required

Name:	**OntoLearn Tool**
URL:	http://lcl.di.uniroma1.it/tools.jsp
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	OWL
Functionalities provided: Type of interface:	Ontology learning User interface
Ontology repository used: Required/Optional:	Local filesystem Required
Data repository used: Required/Optional:	Local filesystem Required
Data repository used: Required/Optional:	WordNet Required
Data repository used: Required/Optional:	SemCor Required

Name:	**Text2Onto**
URL:	http://ontoware.org/projects/text2onto/
Type of implementation:	Application

Multiple components:	No
Representation formalisms:	RDF(S), OWL, F-Logic
Functionalities provided: Type of interface:	Ontology learning User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	KAON Required KAON API Programming interface
Data repository used: Required/Optional:	Local filesystem Required
Data repository used: Required/Optional:	WordNet Required

Name:	**TERMINAE**
URL:	http://www-lipn.univ-paris13.fr/~szulman/TERMINAE.html
Type of implementation:	Application
Multiple components:	Yes, it is an Ontology editor and Ontology browser
Representation formalisms:	RDF(S), OWL, OIL, XML
Functionalities provided: Type of interface:	Ontology learning User interface
Ontology repository used: Required/Optional:	Local filesystem Required
Data repository used: Required/Optional:	Local filesystem Required

6.3.5 Ontology matcher component

Below, the ontology matchers modules are classified under two categories: matchers that provide the basic function of taking two ontologies and generating an alignment, and a framework that integrates many functions around alignments.

Ontology matchers

Name:	**AMW**
URL:	http://www.eclipse.org/gmt/amw/
Type:	semi-automatic
Input formalisms:	UML
Output formalisms:	alignment
Functionalities provided: Type of interface:	Editor/Transformer/Data translator Plug-in
Dependencies:	Eclipse

Name:	**AUTOMS**
URL:	http://www.icsd.aegean.gr/ai-lab/projects/AUTOMS/
Type:	standalone/semi-automatic
Input formalisms:	OWL
Output formalisms:	OWL
Functionalities provided: Type of interface:	Matcher/Merger CLI
Dependencies:	WordNet

Name:	**CMS**
URL:	http://www.aktors.org/crosi/
Type:	standalone
Input formalisms:	OWL

Output formalisms:	Alignment format/OWL/SKOS
Functionalities provided: Type of interface:	Matcher API/Web(servlet)/CLI
Dependencies:	WordNet, Jena

Name:	**CtxMatch**
URL:	http://dit.unitn.it/~zanobini/downloads.html
Type:	semi-automatic
Input formalisms:	OWL/Taxonomy
Output formalisms:	Alignment
Functionalities provided: Type of interface:	Matcher CLI
Dependencies:	WordNet, SAT solvers

Name:	**eTuner/iMap/Glue/LSD**
URL:	http://anhai.cs.uiuc.edu/home/projects/schema-matching.html
Type:	standalone
Input formalisms:	BDSchema/XML Schema/taxonomy
Output formalisms:	Alignment
Functionalities provided: Type of interface:	Matcher CLI

Name:	**Falcon-AO**
URL:	http://xobjects.seu.edu.cn/project/falcon/falcon.htm
Type:	standalone
Input formalisms:	OWL/RDF
Output formalisms:	Alignment format
Functionalities provided: Type of interface:	Matcher CLI

Name:	**NOM, QOM, APFEL**
URL:	http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/
Type:	standalone
Input formalisms:	OWL/RDF
Output formalisms:	Alignment
Functionalities provided: Type of interface:	Matcher GUI/API/Plug-in
Dependencies:	FOAM, KAON2

Name:	**H-Match**
URL:	http://islab.dico.unimi.it/hmatch/
Type:	standalone
Input formalisms:	OWL
Output formalisms:	Alignment
Functionalities provided: Type of interface:	Matcher/Mediator generator GUI/API/Web/WS/CLI/Plug-in

Name:	**LOM**
URL:	http://reliant.teknowledge.com/DAML/
Type:	standalone/semi-automatic
Input formalisms:	OWL/DAML
Output formalisms:	Alignment
Functionalities provided: Type of interface:	Server/Matcher/Transformer WS

Name:	**MapOnto**
URL:	http://www.cs.toronto.edu/semanticweb/maponto/index.html
Type:	standalone/semi-automatic
Input formalisms:	OWL/DBSchema/XML Schema
Output formalisms:	Rules
Functionalities provided: Type of interface:	Matcher/Merger Plug-in
Dependencies:	Protégé

Name:	**MetaQuerier**
URL:	http://metaquerier.cs.uiuc.edu/
Type:	semi-automatic
Input formalisms:	Web interface
Output formalisms:	Query answers
Functionalities provided: Type of interface:	Matcher/Server/Query engine WS

Name:	**MoA**
URL:	http://mknows.etri.re.kr/moa/
Type:	standalone/semi-automatic
Input formalisms:	OWL
Output formalisms:	OWL
Functionalities provided: Type of interface:	Matcher/Editor GUI/CLI

Name:	**OLA**
URL:	http://www.iro.umontreal.ca/~owlola/alignment.html
Type:	standalone
Input formalisms:	OWL
Output formalisms:	Alignment format
Functionalities provided: Type of interface:	Matcher/ API/CLI
Dependencies:	Alignment API, OWL API

Name:	**S-Match**
URL:	http://dit.unitn.it/~accord/
Type:	standalone/semi-automatic
Input formalisms:	OWL/Taxonomy/XML Schema
Output formalisms:	Alignment
Functionalities provided: Type of interface:	Matcher/ API/CLI
Dependencies:	WordNet, SAT solvers

Name:	**SAMBO**
URL:	http://www.ida.liu.se/~iislab/projects/SAMBO/
Type:	standalone/semi-automatic
Input formalisms:	OWL/DAML+OIL
Output formalisms:	Alignment/OWL/DAML+OIL
Functionalities provided: Type of interface:	Matcher/Merger Web

Name:	**Similarity Flooding**
URL:	http://www-db.stanford.edu/~melnik/mm/sfa/

Type:	semi-automatic
Input formalisms:	DB Schema/XML Schema
Output formalisms:	Alignment
Functionalities provided: Type of interface:	Matcher API
Dependencies:	Contained in Rondo system

Name:	**ToMAS/Clio**
URL:	http://www.cs.toronto.edu/db/clio/
Type:	semi-automatic
Input formalisms:	DB Schema/XML Schema
Output formalisms:	SQL Queries
Functionalities provided: Type of interface:	Editor/Matcher/Data translator GUI

Name:	**OntoBuilder**
URL:	http://iew3.technion.ac.il/OntoBuilder/
Type:	semi-automatic
Input formalisms:	Web forms/XML Schema
Output formalisms:	Mediator
Functionalities provided: Type of interface:	Matcher/Data translator/Mediator generator GUI/Web/WS/CLI

Name:	**OntoMerge**
URL:	http://cs-www.cs.yale.edu/homes/dvm/daml/ontology-translation.html
Type:	standalone/semi-automatic
Input formalisms:	OWL
Output formalisms:	OWL
Functionalities provided: Type of interface:	Matcher/Merger Web

Alignment frameworks

Name:	**Alignment API & Alignment server**
URL:	http://co4.inrialpes.fr/align/align.html
Type:	standalone
Input formalisms:	OWL
Output formalisms:	Alignment format
Functionalities provided: Type of interface:	Matcher/Server/API/Transformer/Merger/Data translator/Mediator generator API/Web/WS/CLI
Dependencies:	MySQL (for server)

Name:	**COMA & COMA++**
URL:	http://dbs.uni-leipzig.de/Research/coma.html
Type:	semi-automatic
Input formalisms:	OWL/DBSchema/XML Schema
Output formalisms:	Alignment
Functionalities provided: Type of interface:	Matcher/Data translator GUI

Name:	**FOAM**
URL:	http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/
Type:	standalone

Input formalisms:	OWL/RDF
Output formalisms:	Alignment
Functionalities provided: Type of interface:	Matcher GUI/API/Plug-in

Name:	**PROMPT**
URL:	http://protege.stanford.edu/plugins/prompt/prompt.html
Type:	standalone/semi-automatic
Input formalisms:	OWL/RDF
Output formalisms:	OWL/RDF
Functionalities provided: Type of interface:	Matcher/Editor/Transformer/Merger/Data translator GUI/Plug-in
Dependencies:	Protégé

Name:	**Rondo**
URL:	http://infolab.stanford.edu/~melnik/mm/rondo/
Type:	semi-automatic
Input formalisms:	DB Schema/XML Schema
Output formalisms:	OWL
Functionalities provided: Type of interface:	Matcher/Transformer/Merger GUI

Name:	**Chimaera**
URL:	http://www.ksl.stanford.edu/software/chimaera/
Type:	semi-automatic
Input formalisms:	OWL
Output formalisms:	OWL
Functionalities provided: Type of interface:	Editor GUI

Name:	**MAFRA**
URL:	http://sourceforge.net/projects/mafra-toolkit/
Type:	standalone/semi-automatic
Input formalisms:	OWL/RDF
Output formalisms:	Rules
Functionalities provided: Type of interface:	Matcher/API GUI/API

6.4 Ontology Customization

The *Ontology customization* component implementations can be classified into several different types: applications whose main goal is the construction and/or the gathering of inputs for ontology customization, ontology customization operators, ontology visualization plugins of larger applications, and standalone application that provide ontology customization functionalities.

6.4.1 Ontology localization and profiling component

Name:	**OntoGen**
URL:	http://ontogen.ijs.si/
Type of implementation:	Application
Multiple components:	Yes
Representation formalisms:	RDF, OWL
Functionalities provided:	Extend ontology using a custom text corpus

Type of interface:	standalone
Ontology repository used:	Local, standalone
Required/Optional:	Required
Interface:	Proprietary
Type of interface:	Programming interface, user interface

Name:	**Calendar Apprentice**
URL:	http://citeseer.ist.psu.edu/481999.html
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF, (OWL) Rules
Functionalities provided:	Derive rules for customizing ontology instance selection
Type of interface:	standalone
Ontology repository used:	local
Required/Optional:	optional
Interface:	API
Type of interface:	Programming interface

Name:	**Personal WebWatcher**
URL:	http://www-ai.ijs.si/DunjaMladenic/pww.html
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF and custom
Functionalities provided:	Create profiles of how users interact with ontological instances and propose link recommendations
Type of interface:	Web interface and a local application
Ontology repository used:	Local filesystem
Required/Optional:	Required
Interface:	API
Type of interface:	Programming interface

Name:	**Document Atlas**
URL:	http://docatlas.ijs.si/
Type of implementation:	Application
Multiple components:	Yes.
Representation formalisms:	OWL and custom
Functionalities provided:	Create custom rich visualization of ontologies and structured information about text corpora
Type of interface:	Web interface, local application
Ontology repository used:	Local
Required/Optional:	Optional
Interface:	API and GUI
Type of interface:	Programming and user interfaces

6.4.2 Ontology discovery and ranking component

Name:	**Watson**
URL:	http://watson.open.ac.uk
Type of implementation:	Application, framework
Multiple components:	Yes
Representation formalisms:	RDF(S), OWL
Functionalities provided:	Discover ontologies, describe ontologies, assess ontology quality on multiple measures, search ontologies
Type of interface:	Web interface

Ontology repository used: Required/Optional: Interface: Type of interface:	Web server based on Jena and custom DB-s Required API-s, web interfaces Programming and user interfaces

Name:	**Swoogle**
URL:	http://swoogle.umbc.edu/
Type of implementation:	Application
Multiple components:	Yes
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Collect and search ontologies API and User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem (?) Required API, GUI Programming and user interfaces

6.4.3 Ontology adaptation operators component

Name:	**ONION**
URL:	http://infolab.stanford.edu/~prasen9/
Type of implementation:	Application
Multiple components:	Yes
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Semi-automatic derivation of rules and mappings using operators to extend ontologies, ontology composition User interface
Ontology repository used: Required/Optional:	Local filesystem Optional

Name:	**PROMPT**
URL:	http://protege.stanford.edu/plugins/prompt/prompt.html
Type of implementation:	Algorithm(s)
Multiple components:	yes
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Creating ontologies by applying various operators onto the existing ontologies and data Web interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local Optional API Plugin (e.g. in Protégé)

Name:	**Chimaera**
URL:	http://www-ksl.stanford.edu/software/chimaera/
Type of implementation:	Web application
Multiple components:	Yes
Representation formalisms:	RDF(S), OWL (OKBC compliant)
Functionalities provided: Type of interface:	Create ontologies using a range of predefined operators mainly for merging User interface
Ontology repository used: Required/Optional: Interface:	Protégé and local (user submitted) Optional Web application (GUI)

Type of interface:	User interface

Name:	**FONTE**
URL:	http://www.aifb.uni-karlsruhe.de/~sst/Research/Publications/K-Cap-03.pdf
Type of implementation:	Application
Multiple components:	No
Representation formalisms:	RDF(S)
Functionalities provided: Type of interface:	Factorize and merge ontologies from different domains by means of 'Cartesian product' like operator User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local (user submitted) Optional GUI User interface

6.4.4 Ontology view customization component

Name:	**Longwell**
URL:	http://simile.mit.edu/longwell/
Type of implementation:	Web application, framework
Multiple components:	Yes
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Browse and navigate through ontologies in a faceted manner User interface and stylesheets
Ontology repository used: Required/Optional: Interface: Type of interface:	Custom Required GUI, API Programming and also user interfaces

Name:	**TGVizTab**
URL:	http://www.ecs.soton.ac.uk/~ha/TGVizTab/TGVizTab.htm
Type of implementation:	Protégé plugin
Multiple components:	No
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Browse ontologies User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Protégé Required Protégé plugin APIs Programming interface

Name:	**OntoViz**
URL:	http://protege.cim3.net/cgi-bin/wiki.pl?OntoViz
Type of implementation:	Protégé plugin
Multiple components:	No
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Browse ontologies User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Protégé Required Protégé plugin APIs Programming interface

Name:	**Jambalaya**
URL:	http://www.thechiselgroup.org/jambalaya
Type of implementation:	Protégé plugin
Multiple components:	No
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Browse ontologies User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Protégé Required Protégé plugin APIs Programming interface

Name:	**OWLViz**
URL:	http://www.co-ode.org/downloads/owlviz/co-ode-index.php
Type of implementation:	Protégé plugin
Multiple components:	No
Representation formalisms:	OWL
Functionalities provided: Type of interface:	Browse ontologies User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Protégé Required Protégé plugin APIs Programming interface

Name:	**/facet**
URL:	http://swik.net/slashfacet
Type of implementation:	Web application, framework
Multiple components:	Yes
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Browse and navigate through ontologies in a faceted manner User interface and stylesheets
Ontology repository used: Required/Optional: Interface: Type of interface:	Custom Required GUI, API Programming and also user interfaces

Name:	**mSpace**
URL:	http://mspace.fm/
Type of implementation:	Web application, framework
Multiple components:	Yes
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Browse and navigate through ontologies in a faceted manner, define facets based on ontological rules/properties User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Custom, 3store Required GUI, API Programming and also user interfaces

Name:	**VIKI**
Type of implementation:	Web application, framework
Multiple components:	Yes
Representation formalisms:	RDF(S), OWL

Functionalities provided: Type of interface:	Browse and navigate through ontologies in a faceted manner User interface and stylesheets
Ontology repository used: Required/Optional: Interface: Type of interface:	Custom Required GUI, API Programming and also user interfaces

Name:	**CropCircles**
URL:	http://ontoworld.org/wiki/CropCircles:_Topology_Sensitive_Visualization_of_OWL_Class_Hierarchies
Type of implementation:	Standalone application, algorithm
Multiple components:	Yes
Representation formalisms:	OWL
Functionalities provided: Type of interface:	Visualize and navigate through ontologies in by means of ontology import (inclusion) User interface
Ontology repository used: Required/Optional: Interface:	Custom Optional GUI

Name:	**CS AKTive Space**
URL:	http://cs.aktivespace.org/
Type of implementation:	Web application
Multiple components:	Yes
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Browse and navigate through ontological instances in a faceted manner and using different visualization metaphors User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Custom, 3Store Required GUI User interface

Name:	**SpaceTree**
Type of implementation:	Standalone application, algorithm
Multiple components:	Yes
Representation formalisms:	OWL
Functionalities provided: Type of interface:	Visualize and navigate through ontologies by means of hypertree like structures User interface
Ontology repository used: Required/Optional: Interface:	Custom Optional GUI

Name:	**TreeMap**
Type of implementation:	Standalone application, algorithm
Multiple components:	Yes
Representation formalisms:	OWL
Functionalities provided: Type of interface:	Visualize and navigate through ontologies by means of tree structures mapped onto geometrical regions User interface
Ontology repository used: Required/Optional: Interface:	Custom Optional GUI

Name:	**Spotlight**
URL:	http://kmi.open.ac.uk/people/paulm/rae/ht05.pdf
Type of implementation:	Standalone application, algorithm
Multiple components:	No
Representation formalisms:	OCML (OWL via import)
Functionalities provided: Type of interface:	Visualize and navigate through ontological instances by means of their semantic and usage relevance, proximity User interface
Ontology repository used: Required/Optional: Interface:	Custom Required GUI

Name:	**IsaViz**
URL:	http://www.w3.org/2001/11/IsaViz/
Type of implementation:	Application, plugin
Multiple components:	Yes. It is also an RDF editor.
Representation formalisms:	RDF
Functionalities provided: Type of interface:	Browse RDF User interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local filesystem Required Jena API Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Web server Optional Jena API Programming interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Sesame Optional Sesame API Programming interface

6.5 Ontology Evolution

6.5.1 Ontology versioner component

The Ontology versioner component implementations usually present a library that implements the essential functions for storing ontology versions, difference computation (either syntactic or semantic), querying of multiple versions, change management (e.g. user commits, check-outs, branching, etc.). Other components of the evolution dimension may build on these libraries.

Name:	**SemVersion**
URL:	http://ontoware.org/projects/semversion/
Type of implementation:	Program library
Multiple components:	No
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Version ontologies Programmatic interface
Ontology and data repository used: Required/Optional: Interface:	RDF2Go Required Program library

Type of interface:	Programmatic interface

Name:	**DIP ontology versioning**
URL:	http://www.omwg.org/tools/versioning/v1.0/versioning.zip
Type of implementation:	Program library
Multiple components:	No
Representation formalisms:	WSML
Functionalities provided: Type of interface:	Version ontologies Programmatic interface

6.5.2 Ontology evolution visualizer component

The Ontology evolution visualizer component implementations usually present a user interface that allows browsing an ontology in the context of its multiple versions, comparing visually different ontologies, and possibly performing some versioning operations within the visual interface (e.g. merging of branches). Other components of the evolution dimension may incorporate this interface.

Name:	**SemVersion Protégé plug-in**
URL:	http://ontoware.org/projects/semversion/
Type of implementation:	Plug-in
Multiple components:	Yes – versioner and visualizer
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Version ontologies, browse the versions User interface
Ontology and data repository used: Required/Optional: Interface: Type of interface:	RDF2Go, Protégé native repositories Required Program libraries Programmatic interfaces

Name:	**PROMPT, PROMPTDiff**
URL:	http://protege.stanford.edu/plugins/prompt/prompt.html
Type of implementation:	Plug-in
Multiple components:	Yes – ontology merging and difference visualisation tool
Representation formalisms:	OWL, other formats supported in Protégé
Functionalities provided: Type of interface:	Browse differences between different versions User interface
Ontology and data repository used: Required/Optional: Interface: Type of interface:	Protégé native repositories Required Program library Programming interface

6.5.3 Ontology evolution manager component

The Ontology evolution manager component implementations are usually incorporated into complex ontology development and evolution framework, providing either APIs or user (web of standalone) interfaces. The component wraps the functionalities of lower-level components (ontology versioner, visualizer).

Name:	**KAON**
URL:	http://kaon.semanticweb.org/
Type of implementation:	Application, library

Multiple components:	Yes – It is an ontology development platform
Representation formalisms:	KAON knowledge model, RDF(S), OWL extensions
Functionalities provided: Type of interface:	Ontology development, browsing, reasoning, version management using transaction mechanism User interfaces, programmatic interface
Ontology and data repository used: Required/Optional: Interface: Type of interface:	KAON native Required Program library Programming interface

Name:	**DOME**
URL:	http://dome.sourceforge.net
Type of implementation:	Application
Multiple components:	Yes – an ontology management environment
Representation formalisms:	WSML
Functionalities provided: Type of interface:	Ontology development, browsing, version management (DIP versioning used), ... User interface

Name:	**MarcOnt Portal**
URL:	http://portal.marcont.org
Type of implementation:	Application, service interfaces planned
Multiple components:	Yes – ontology merging and difference visualisation tool
Representation formalisms:	OWL, other formats supported in Protégé
Functionalities provided: Type of interface:	Collaborative ontology development, ontology version visualization and management Web interface, service interfaces planned
Ontology and data repository used: Required/Optional: Interface: Type of interface:	Jena Required Program library Programming interface

Name:	**Linkfactory**
URL:	http://www.landcglobal.com/pages/linkfactory.php
Type of implementation:	Application
Multiple components:	Yes. It is an ontology development tool
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Browse ontologies, manage versions in a database User interface
Ontology repository used: Required/Optional:	Oracle, Sybase and SQLServer Optional

Name:	**Powl**
URL:	http://aksw.informatik.uni-leipzig.de/Projects/Powl
Type of implementation:	Application
Multiple components:	Yes. It is an ontology development tool
Representation formalisms:	RDF(S), OWL
Functionalities provided: Type of interface:	Browse ontologies, manage versions (syntactic) User interface
Components used: Required/Optional: Interface:	Powl store Required RAP – RDF API for PHP

Type of interface:	Programming interface

6.6 Ontology Instance Generation

6.6.1 Instance editor component

Name:	**GATE Ontology Editor**
URL:	www.gate.ac.uk
Type of implementation:	GATE plug-in
Multiple components:	No
Representation formalisms:	A combination of limited OWL Lite and unconstrained RDFS
Functionalities provided: Type of interface:	Create, Browse, Modify ontology GUI
Ontology repository used: Required/Optional: Interface: Type of interface:	In memory Required GATE Ontology API Programming Interface

Name:	**OCAT**
URL:	www.gate.ac.uk
Type of implementation:	GATE plug-in
Multiple components:	NO
Representation formalisms:	A combination of limited OWL Lite and unconstrained RDFS
Functionalities provided: Type of interface:	Creation of new manual instances from any text GUI
Ontology repository used: Required/Optional: Interface: Type of interface:	In memory Required GATE Ontology API Programming Interface
Data Repository used: Required/Optional: Interface: Type of interface:	GATE Documents Required GATE API Programming Interface

6.6.2 Manual annotation component

Name:	**OCAT**
URL:	www.gate.ac.uk
Type of implementation:	GATE plug-in
Multiple components:	NO
Representation formalisms:	a combination of limited OWL Lite and unconstrained RDFS
Functionalities provided: Type of interface:	Concept, instance and property annotations GUI
Ontology repository used: Required/Optional: Interface: Type of interface:	GATE Ontology API – a wrapper for OWLIM Required GATE plug-in API and GUI
Data Repository used: Required/Optional: Interface: Type of interface:	GATE Documents Required GATE API Programming Interface

Name:	**OntoMat-Annotizer**
URL:	http://annotation.semanticweb.org/ontomat/index.html

Type of implementation:	Application
Multiple components:	NO
Representation formalisms:	OWL
Functionalities provided: Type of interface:	Web page annotation tool, creating and maintaining ontology-based OWL-Mark-ups, Ontology Browsing GUI
Ontology repository used: Required/Optional: Interface: Type of interface:	Ontobroker's underlying F-Logic based inference engine SilRI Required GUI User Interface

Name:	**M-OntoMat-Annotizer**
URL:	http://www.acemedia.org/aceMedia/results/software/m-ontomat-annotizer.html
Type of implementation:	Application
Multiple components:	NO
Representation formalisms:	RDFS
Functionalities provided: Type of interface:	Region-based image & video annotation, creation of RDFS ontology-based mark-ups, association of ontology concepts with prototype instances of MPEG-7 visual descriptors, ontology browsing GUI

Name:	**PhotoStuff (Mindswap)**
URL:	http://www.mindswap.org/2003/PhotoStuff/
Type of implementation:	Application
Multiple components:	NO
Representation formalisms:	RDFS and OWL
Functionalities provided: Type of interface:	Region-based image annotation, RDFS/OWL ontology mark-ups, keyword-based metadata search, ontology browsing GUI
Ontology repository used: Required/Optional: Type of interface:	Sesame/Kowari Optional API

Name:	**AKTive Media - Ontology based annotation system**
URL:	http://www.dcs.shef.ac.uk/~ajay/html/cresearch.html
Type of implementation:	Application
Multiple components:	NO
Representation formalisms:	RDFS, DAML and OWL
Functionalities provided: Type of interface:	Image (region-based) and text annotation, RDFS/DAML/OWL ontology mark-ups, ontology browsing, SPARQL query/retrieval GUI

Name:	**Magpie**
URL:	http://kmi.open.ac.uk/projects/magpie
Type of implementation:	Internet Browser plugin
Functionalities provided:	Web and Semantic Web are usually seen as two fairly independent technologies. Magpie uses KMi's ontology infrastructure and expertise in handling ontologies to semantically markup web documents on the fly. Magpie technology is lightweight, yet flexible and providing

	sufficiently robust and open features for semantically enriched web browsing. Magpie as a web browser plugin aims to identify and filter out the concepts-of-interest from any webpage it is given. The current set of concepts can be influenced by a selection of a particular ontology of concepts and relations. In addition to identifying the concepts that are relevant from the perspective of a particular ontology, each such concept may provide an applicable set of relations or commands that can be executed. These are accessible via contextual semantic menus. Magpie is available for Internet Explorer and Mozilla/Firefox, and has been deployed in several commercial scenarios, the most recent one being semantic browsing support in the Food and Agriculture Organization of the United Nations (FAO).

Name:	**Ontolog**
URL:	http://www.idi.ntnu.no/~heggland/ontolog/
Type of implementation:	Application
Multiple components:	NO
Representation formalisms:	RDFS
Functionalities provided: Type of interface:	Video annotation, RDFS ontology mark-up, ontology creation, ontology browsing GUI

6.6.3 Automatic annotation component

Name:	**KIM**
URL:	http://www.ontotext.com/kim/
Type of implementation:	Web Interface
Multiple components:	Yes
Representation formalisms:	a combination of limited OWL Lite and unconstrained RDFS
Functionalities provided: Type of interface:	Automatic instance annotation, indexing and searching Web Interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Web Server Required KIM API Programming Interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Local File System Required KIM API Programming Interface
Data repository used: Required/Optional: Interface: Type of interface:	GATE Documents Required GATE API Programming Interface

Name:	**AKTAgent**
Multiple components:	Yes
Representation formalisms:	a combination of limited OWL Lite and unconstrained RDFS
Functionalities provided:	AKTAgent extends and enhances the functionalities provided by the KIM platform that provides semantic annotation, indexing and retrieval of documents. In AKTAgent, users create and store queries that are periodically submitted to a search engine. In this way agents search for documents that match the users long term interests. Unlike similar applications (such GoogleAlert) the use of semantic web

	technology permits users to specify semantic queries according to an ontology (the one provided by KIM for the annotation of the resources). This allows users to retrieve information more accurately than search engines that express queries based on natural language only. In addition, these user-specified queries and their results can be used to further enhance the indexing and extraction process of the search engine.

Name:	**GATE ML**
URL:	http://www.gate.ac.uk
Type of implementation:	API
Multiple components:	Yes
Representation formalisms:	Machine Learning Model
Functionalities provided: Type of interface:	Automatic annotation API and GUI
Ontology repository used: Required/Optional: Interface: Type of interface:	Local File System Required GATE Learning API Programming Interface
Data repository used: Required/Optional: Interface: Type of interface:	GATE Documents Required GATE API Programming Interface

6.6.4 Ontology populator component

Name:	**CLIE (Controlled Language Information Extraction)**
URL:	Not public yet
Type of implementation:	GATE Application
Multiple components:	YES
Representation formalisms:	a combination of limited OWL Lite and unconstrained RDFS
Functionalities provided: Type of interface:	Ontology population GUI & API
Ontology repository used: Required/Optional: Interface: Type of interface:	In Memory Required GATE Ontology API Programming Interface
Data repository used: Required/Optional: Interface: Type of interface:	GATE Documents Required GATE API Programming Interface

Name:	**ALVIS**
URL:	http://www.alvis.info/alvis/
Functionalities provided:	ALVIS allows application-domain experts to link together individual sites so that can they can form a search network, by providing means to develop complementary, distributed components, together with bridges to existing topic-specific search sites. The system relies on a semantic-based search engine that is intended to automatically build and maintain its own semantic structure from input primitive ontologies. Although making use of ontologies, the semantic structure is created semi-automatically using statistical and machine learning methods for the purpose of returning better search results. The distributed system is intended to operate with heterogeneous search servers, using query topics as a routing mechanism, and using

	distributed methods for ranking and semantic-based processing.

Name:	**AKTive Futures**
URL:	http://triplestore.aktors.org/demo/AKTiveFutures/
Functionalities provided:	The portal is meant to support analyst work by providing a means to analyse a large information space. The portal provides a conceptual open hypertext interface that annotates external resources using a domain ontology, and it is complemented by a graphing tool that allows the analysis of trends in temporal data in the context of relevant contemporary events. An ontology of business drivers provides a common framework used to mediate information gathered from different sources. The ontology allows drawing inferences that are used to indicate the relevance of datasets to key drivers. Data is gathered from a variety of freely-available sources, transformed into RDF/XML using the vocabulary defined by the domain ontology, and stored in an RDF triplestore that provides a query interface to the other system components.

6.7 Semantic Web Services

6.7.1 Web Service discoverer component

Name:	**Hybrid OWL-S Web Service Matchmaker – OWLS MX**
URL:	http://www-ags.dfki.uni-sb.de/~klusch/owls-mx/
Type of implementation:	Open source
Multiple components:	No
Representation formalisms:	OWL
Functionalities provided:	OWLS-MX is a hybrid semantic Web service matchmaker that retrieves services for a given query written in OWL-S, and based on imported ontologies in the W3C recommended ontology web language OWL. For this purpose, the OWLS-MX matchmaker performs pure profile based service IO-matching but it combines crisp logic-based semantic matching with syntactic token-based similarity metrics to obtain the best of both worlds - description logics and information retrieval. The "X" in OWLS-MX stands for five different instances (M0 to M4) of the generic hybrid matching scheme OWLS-MX, depending on whether and which syntactic similarity metric is used. The OWLS-MX matchmaker is fully implemented in Java, uses the OWL-DL description logic reasoner Pellet for logic based filtering, the cosine, loss-of-information, extended Jacquard, and Jensen-Shannon information divergence based similarity metrics for complementary approximate matching.
Type of interface:	Java, API
Ontology repository used: Required/Optional: Interface: Type of interface:	Ad Hoc Required OWL files None
Web Service Registry used: Required/Optional: Interface: Type of interface:	Test collection OWLS-TC Required OWL files None

Name:	**The TUB OWL-S Matcher (The OWLSM)**
URL:	http://owlsm.projects.semwebcentral.org/
Type of implementation:	Java

Multiple components:	No
Representation formalisms:	OWL-S, OWL
Functionalities provided: Type of interface:	The OWL-S Matcher is a Java implementation of a matchmaking algorithm for matching OWL-S descriptions. OWL-S is an upper ontology that defines a vocabulary for describing services. OWL-S can be used to define classifications for the elements and characteristics of a Web service. OWL-S is based on the Web Ontology Language (OWL). The matchmaker compares two descriptions (one form a service requester and another by the service provider) and identifies different relations between the two descriptions (e.g. "match" or "no match") API
Ontology repository used: Required/Optional: Interface: Type of interface:	OWL ontologies Required None None
Web Service Registry used: Required/Optional: Interface: Type of interface:	OWL-S services Required None None

Name:	**WSMX Discovery Framework**
URL:	http://sourceforge.net/projects/wsmx/
Type of implementation:	Open Source
Multiple components:	No
Representation formalisms:	WSML (DL/Rule/Flight)
Functionalities provided: Type of interface:	Discovery of Semantic Web Services API
Ontology repository used: Required/Optional: Interface: Type of interface:	Dynamic Web Locator and WSMX internal repository Optional WSMO4J API
Web Service Registry used: Required/Optional: Interface: Type of interface:	Dynamic Web Locator and WSMX internal repository Optional WSMO4J API
Metadata Registry used: Required/Optional: Interface: Type of interface:	Dynamic Web Locator and WSMX internal repository Optional WSMO4J API
Semantic Query Processor used: Required/Optional: Interface: Type of interface:	WSML2Reasoner Framework. Can be used with one of the following reasoners: MINS, IRIS, KAON2, Pellet Required WSML2Reasoner API

Name:	**OWL Semantic Search Services (owl-semsearch)**
URL:	http://projects.semwebcentral.org/projects/owl-semsearch/
Type of implementation:	Open Source
Multiple components:	No
Representation formalisms:	OWL
Functionalities provided:	OWL Semantic Search Services crawls and indexes DAML/OWL content on the Web. Users submit logical queries that are answered with exact data. It can broaden queries with simple inference, such

Type of interface:	as equivalence, inversion, generalization and specialization API
Web Service Registry used: Required/Optional: Interface: Type of interface:	Ad hoc repository Optional OWL-S interface API

6.7.2 Web Service selector component

Name:	**WSMX Selector and Ranking Prototype**
URL:	http://sourceforge.net/projects/wsmx/
Type of implementation:	Software Component - Open Source
Multiple components:	YES
Representation formalisms:	WSML
Functionalities provided: Type of interface:	Ranking of services based on non-functional properties and selection API
Web Service Discoverer used: Required/Optional: Interface: Type of interface:	Any WSMX Discovery component Required API Programming Interface

6.7.3 Web Service composer component

Name:	**Kweb- Semantic Web Service Composition**
URL:	http://www.astroproject.org/downloads/kweb/
Type of implementation:	Open Source
Multiple components:	Yes
Representation formalisms:	WSMO, BPEL
Functionalities provided: Type of interface:	Discovery of the most relevant services to perform the end to end composition. First-step of composition i.e., Functional Level composition through AI planning. Second-step of composition i.e., Process Level composition. Execution of the whole composition as a BPEL process. API
Ontology repository used: Required/Optional: Interface: Type of interface:	UDDI Required API/URI/SWS jUDDI
Web Service Registry used: Required/Optional: Interface: Type of interface:	UDDI Required API/URI/SWS jUDDI

Name:	**Semantic Web service composition through Causal Link Composition**
URL:	ftp://huitrier.rd.francetelecom.com
Type of implementation:	Software Component
Multiple components:	Yes
Representation formalisms:	OWL, WSMO, BPEL
Functionalities provided:	Discovery of the most relevant services to perform the end to end composition First-step of composition i.e., Functional Level composition through AI planning i.e., Causal Link composition between Web services. Generation of a picture of the generated process i.e., jpeg, png, dot… Execution of the whole composition as a BPEL process.

Type of interface:	API
Ontology repository used: Required/Optional: Interface: Type of interface:	UDDI Required API/URI/SWS jUDDI
Web Service Registry used: Required/Optional: Interface: Type of interface:	UDDI Required API/URI/SWS jUDDI

Name:	**Composer**
URL:	http://www.mindswap.org/2005/composer/downloads/ and http://svn.mindswap.org/composer/
Type of implementation:	Open Source
Multiple components:	No
Representation formalisms:	OWL
Functionalities provided: Type of interface:	A prototype that guides a user in the dynamic composition of web services. Semi-automatic process includes presenting matching services to the user at each step of a composition, filtering the possibilities by using semantic descriptions of the services. The generated composition is then directly executable through the WSDL grounding of the services. API, java

Name:	**Semantic web services browser and composer**
Type of implementation:	Commercial
Multiple components:	Yes
Representation formalisms:	OWL-S
Functionalities provided:	The Semantic Web Services Browser and composer is a system for searching, retrieving, invoking and composing semantic web services. Using a SESAME-based registry, users can store OWL-S descriptions of web services and link them to an ontology of services categories, which is displayed in the browser. The user can search or browse this ontology to find the service that they require. They can then invoke this service directly or use it as a basis to begin the composition of a more complex service. The composition module gives the users a graphical view of the web service and allows them to select input or outputs. The system will then automatically search all other web services in the repository to find services that have semantically equivalent input/outputs, which could be linked to create a composite service. This can be repeated until the user has built the required composition. Any non-matching inputs can be entered manually, and the composition invoked. The Browser offers the facility to combine Web Services so that the data output of one service can be fed into the input of another, thus creating a new composite Web Service. Currently, the Browser assumes that the data types of these inputs and outputs are the same. More realistically, a mediation function would be required to convert between differing data types.

Name:	**Web Service Composition**
URL:	Work in progress – being developed within SUPER project
Type of implementation:	Not licensed yet (hence, it cannot be distributed)
Multiple components:	No
Representation formalisms:	Accepts WSMO Descriptions (excluding Mediators)

Functionalities provided: Type of interface:	Validation of the input WSMO descriptions that assure they comply with the underlying formalism on which the tool is based. Upon successful validation, the WSMO elements are translated and fed in to the composition tool. Using similar techniques as for AI Planning, the tool searches for a solution within the input pool of Web Services which fulfils the required Goal. The tool takes into consideration the background ontology used by the input descriptions. Plugin matches are considered. The input web services are assumed to be the result of a discovery process. Types of searching techniques: Blind Search, Heuristic, Filtering (using a pruning technique), Full (using both Heuristic and Pruning) Future work will include: Adding expressive constructs to the background theory such that searching for a solution is kept in polynomial time, Parallelization of the output solution, Consideration of Business Policies during the composition process API

Name:	**Service Composition Engine (Developed within ASG)**
URL:	http://asg-platform.org/cgi-bin/twiki/view/Public/ Prototype Demo
Type of implementation:	Open Source (LGPL)
Multiple components:	Yes (within the ASG Context only)
Representation formalisms:	WSMO
Functionalities provided: Type of interface:	Automatic web service composition based on semantic descriptions Uses an Extended Hill Climbing Heuristic search to find a solution. Generates WS-BPEL descriptions Re-Composition in case of Invocation Errors API
Web Service Discoverer used: Required/Optional: Interface: Type of interface:	ASG Discovery Database Required API Programming Interface
Ontology repository used: Required/Optional: Interface: Type of interface:	Internal Repository Required API Programming Interface
Web Service Registry used: Required/Optional: Interface: Type of interface:	Internal Repository Required API Programming Interface

6.7.4 Web Service choreography engine component

Name:	**WSMX Choreography Engine**
URL:	http://sourceforge.net/projects/wsmx/
Type of implementation:	Open Source
Multiple components:	No
Representation formalisms:	WSML
Functionalities provided: Type of interface:	Choreography Management API
Ontology repository used: Required/Optional: Interface: Type of interface:	Local repository Required API Programming Interface

Web Service Process Mediator used: Required/Optional: Interface: Type of interface:	WSMX Process Mediator Optional API Programming Interface
Web Service Registry used: Required/Optional: Interface: Type of interface:	Local Repository Required API Programming Interface

Name:	**IRS-III**
URL:	http://kmi.open.ac.uk/projects/irs/
Type of implementation:	Not applicable
Multiple components:	Yes
Representation formalisms:	OCML
Functionalities provided: Type of interface:	Semantic Web Service Execution Management API/URI/SWS
Ontology repository used: Required/Optional: Interface: Type of interface:	Internal repository Required API/URI/SWS Programming Interface and Browser for user interaction
Web Service Process Mediator used: Required/Optional: Interface: Type of interface:	Internal Mediator Component Required API/URI/SWS Programming Interface and Browser for user interaction
Web Service Repository used: Required/Optional: Interface: Type of interface:	Internal Repository Required API/URI/SWS Programming Interface and Browser for user interaction
Metadata Registry used: Required/Optional: Interface: Type of interface:	Internal (same as ontology repository) Required API/URI/SWS Programming Interface and Browser for user interaction

6.7.5 Web Service process mediator component

Name:	**WSMX Process Mediation Prototype**
URL:	http://sourceforge.net/projects/wsmx/
Type of implementation:	Open Source
Multiple components:	Yes
Representation formalisms:	WSML
Functionalities provided: Type of interface:	Mediation between two WSMO choreographies API
Ontology repository used: Required/Optional: Interface: Type of interface:	Local repository Required API Programming Interface
Ontology Matcher used: Required/Optional: Interface: Type of interface:	WSMX Data Mediator Required API Programming interface

6.7.6 Web Service grounding component

Name:	**WSMX Communication Manager**

URL:	http://sourceforge.net/projects/wsmx/
Type of implementation:	Open Source
Multiple components:	No
Representation formalisms:	WSML / Java
Functionalities provided: Type of interface:	Grounding management for wsml to xml and xml to wsml. (Xml to wsml is based on wsml descriptions, while wsml to xml is based on ad hoc Java components.) API
Web Service Registry used: Required/Optional: Interface: Type of interface:	Local Repository Required API Programming Interface

6.7.7 Web Service profiling component

Name:	**Service Profiler**
URL:	https://subversion.asg-platform.org/svn/branches/M30SourceCodeInstaller/Profiling
Type of implementation:	Java component
Multiple components:	No
Representation formalisms:	XML Schema
Functionalities provided: Type of interface:	Profile creation Programming interface in Java

Name:	**Web Service profiling**
URL:	http://sourceforge.net/projects/wsmx/
Type of implementation:	Software Component - Open Source
Multiple components:	NO
Representation formalisms:	WSML
Functionalities provided: Type of interface:	Monitoring - collecting, computing, and providing other components with values of selected non-functional parameters API
Web Service Registry used: Required/Optional: Interface: Type of interface:	WSMX internal repository Optional WSMO4J API

6.7.8 Web Service Registry component

Name:	**OWL-S UDDI Matchmaker**
URL:	http://projects.semwebcentral.org/projects/owl-s-uddi-mm/ and http://www.daml.ri.cmu.edu/matchmaker/
Type of implementation:	Open Source
Multiple components:	No
Representation formalisms:	Java, OWL
Functionalities provided: Type of interface:	OWL-S/UDDI matchmaker combines UDDI's proliferation into the web service infrastructure and OWL-S's explicit semantic description of the web service. Matchmaker is implemented as an extension of the jUDDI which is an open source Java implementation of the Universal Description, Discovery, and Integration (UDDI) specification for Web Services. API
Ontology Repository used: Required/Optional: Interface:	OWL ontologies Required API

Type of interface:	OWL API

Name:	**OWLS-TC**
URL:	http://projects.semwebcentral.org/frs/download.php/255/owls-tc2.zip or http://projects.semwebcentral.org/projects/owls-tc/
Type of implementation:	Open source
Multiple components:	No
Representation formalisms:	OWL-S
Functionalities provided: Type of interface:	A set of semantic Web services together with their ontologies. OWLS-TC version 2.1 None