Journal of Universal Computer Science, vol. 15, no. 17 (2009), 3201-3230
submitted: 25/2/09, accepted: 25/9/09, appeared: 1/11/09

Extending SD-Core for Ontology-based Data Integration

Ismael Navas-Delgado
(University of Málaga
Málaga
Spain
ismael@lcc.uma.es)

José F. Aldana-Montes
(University of Málaga
Málaga
Spain
jfam@lcc.uma.es)

Abstract: This paper describes the main elements of a functional platform for building Semantic Web Applications, the Semantic Directory (additional information can be found at http://khaos.uma.es/SD-Core). A Semantic Directory provides a resource directory, in which web resources are registered and their semantics published. Using the Semantic Directories we provide a solution for publishing the semantics of resources, and interoperating them with some other applications in the same or different domains. The main idea behind this proposal is to help developers build Semantic Web applications by providing them with functional components for this task. This paper also describes some applications that have been developed using an SD-Core extension: SD-Data. Then, we describe the instantiation of the Khaos Ontology-based Mediation Framework (KOMF) in the Systems Biology domain. This framework provides an architecture that enables the research on the development of ontology-based mediators. Thus, an ontology-based mediator has been produced that has demonstrated its utility in two applications developed in the Amine System Project using the SD-Data for registering semantics: AMMO-Prot and SBMM Assistant. The use of ontologies is limited in the current version of the mediator, but its development as a framework enables the implementation of improvements based on the use of reasoning.
Key Words: Metadata, knowledge management
Category: M.1, M.8

1 Introduction

Semantic Web research has been ongoing since the initial proposal of Tim Berners Lee. Nowadays, this research is producing technology that is being integrated in enterprise applications. In this context, the development of Semantic Web based applications has had to address several problems: to choose a component for dealing with ontologies, to deal with ontology relationships (usually available as ontology alignments) and to relate non-semantic resources with semantics through annotation tasks. These new issues in software development have caused developers significant problems when estimating the real cost of applications, and reusing existing components.

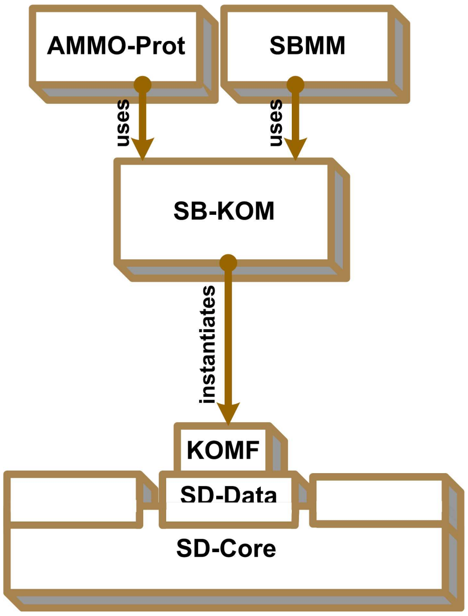

Figure 1: Generic infrastructure and its main extensions.

The essential role of middleware is to manage the complexity and heterogeneity of distributed infrastructures. On the one hand, middleware offers programming abstractions that hide some of the complexities of building a distributed application. On the other hand, a complex software infrastructure is necessary to implement these abstractions. Instead of the programmer having to deal with every aspect of a distributed application, it is the middleware that takes care of some of them.

Ontologies serve various needs in the Semantic Web, such as storage or exchange of data corresponding to an ontology, ontology-based reasoning or ontology-based navigation. When building a complex Semantic Web application, designers may prefer to combine different existing software modules. Thus, our proposal aims to develop a middleware infrastructure which will hide details of the semantics from programmers by providing a set of working components.

An infrastructure is generally a set of interconnected structural elements that provide the framework supporting an entire structure. From our point of view, this is a set of components which provide the basic elements to develop more complex applications.

The huge amount of information and the complexities of handling it have given rise to a lot of research concerning practical approaches to the Semantic Web. One of the central problems in this Web is the efficient integration of data transmitted over the wide area networks. The goal of a data integration system is to provide uniform access to a set of heterogeneous data sources, freeing the user from problems caused by different locations, query languages and different database protocols.

The need for data integration started when the number of applications and data repositories began to grow rapidly. The first approaches appeared in the 80s, and formed the basis for the research in this area. The evolution continued over mediator based systems, such as AMOS II [Risch et al. 2001], DISCO [Tomasic et al. 1997], TSIMMIS [Garcia-Molina et al. 21997] and Garlic [Haas et al. 1996]. Then, agent technology was used in some systems like InfoSleuth [Ksiezyk et al. 2001] and MOMIS [Beneventano et al. 2000]. Finally, the new technologies appearing have been used in data integration: XML (MIX [Bornhovd et al. 1999]), ontologies (OBSERVER [Mena et al. 1996]). Finally, more sophisticated and even commercial systems have appeared, such as FeDeRate [Prudhommeaux 2007], Virtuoso [OpenLink Software 2010], SDS [In Silico Discovery 2010], Concept Based Systems like [Sattler et al. 2005] and Semantic Web Middleware for Virtual Data Integration on the Web [Langegger et al. 2008]. Studying the most relevant data integration systems has allowed us to determine the main elements of a data integration system, and so to extract the pattern for building this kind of system.

In this paper we present the main elements of a functional infrastructure for implementing Semantic Web Applications (Figure 1). The goal of the proposed infrastructure is to provide useful components for registering and managing ontologies and their relationships, and also metadata regarding the resources committed or annotated with the ontologies registered, which means a practical step towards building applications in the Semantic Web. The main advantage of using an infrastructure for the development of Semantic Web applications is that software developers can reuse components, reducing the implementation costs.

The Semantic Directory [Navas-Delgado et al. 2008] [Navas-Delgado et al. 2008B] concept has been implemented as a set of interfaces for managing semantics: **SD-Core** (`http://khaos.uma.es/SD-Core`)[1]. The SD-Core interfaces are extended with new functionalities for a specific type of resource (data providing resources). This extension, **SD-Data** can be used for data integration. In this sense, we present an application type, Ontology-Based Mediation, using a framework that is able to integrate data from disperse data sources: the Khaos Ontology-based Mediation Framework (**KOMF**). This framework has been instantiated to develop a mediator for Systems Biology: the Systems Bi-

[1] We have developed SD-Core as a set of Web Services, so it can be made available to a wide rage of applications needing to use resource semantics. In addition, we have developed an installable version that will include SD-Core in an existing Web Server or in a new one, which will be installed if there is no other one available. This auto-installable version will include a Web interface for testing the SD-Core. In the future this software will be made available as an Open Source project

ology Khaos Ontology-based Mediator (**SB-KOM**), which uses ontologies as integration schemas, and takes advantage of simple reasoning to perform the query rewriting. The development as a framework allows us to introduce new improvements in the mediator gradually, increasing the use of semantics to optimize the query processing and rewriting. This mediator has been used to build two end-user applications: **AMMO-Prot** (The ASP Model Finder) and **SBMM** (The Systems Biology Model Manager). The main contributions of our proposal are:

- The proposed infrastructure provides the minimum set of components required to develop a wide range of applications in the Semantic Web.
- The use of a common infrastructure for developing Semantic Web applications enables the possibility of interconnecting these applications (taking advantage of the semantics) and allowing these applications interoperability.
- The development of applications only involves developing components which extend the infrastructure and provide new functionalities for more complex applications.
- The semantics of Semantic Directories is made explicit and publicly available, so the Semantic Web applications can interoperate directly with them.
- The interfaces provided are also described semantically so they can be combined with other semantically described applications.

Section 2 describes SD-Core showing the characteristics of its generic components. Then, we describe related works (*Section 3*). The developed extension and its applications are described in *Section 4*. *Section 5* describes KOMF (the Khaos Ontology-based Mediation Framework). Applications in Systems Biology are presented in *Section 6*. Finally, we will discuss the main advantages of our proposal (*Section 7*) and the conclusions reached with respect to its implementation and use (*Section 8*).

2 Related Work

In this context the Semantic Web Framework is the most similar work to this paper. However, it is in its initial stage (framework design), while we have successfully applied our infrastructure to create Semantic Web applications in real scenarios. The Semantic Web Framework has a structure in which applications are described using simple components. Thus, the Semantic Web Framework provides a classification and analysis of existing tools, but it does not define even component interfaces. The Semantic Web Framework classifies the components in dimensions that have been chosen from the developers' experience.

Table 1: The Semantic Web Framework dependencies

	1	2	3	4	5	6	7
1. Data and Metadata Management	-		X				
2. Querying and Reasoning	X	-					
3. Ontology Engineering	X	X	-	X			
4. Ontology Customization	X			-			
5. Ontology Evolution	X				-		
6. Ontology Instance Generation	X					-	
7. Semantic Web Services	X	X	X				-

Each component is described by defining its dependencies with other components [Leger et al. 2007], and then a list of use cases is presented. The list of dependencies between dimensions is shown in Table 1. This list has the special characteristic that the first dimension is a requirement for the other ones.

Each use case describes how several components of the framework can be composed to solve a specific problem. From these use cases and the component dependencies we can deduce that some blocks of components can be grouped, because they usually act together. In this case, these groups fully match those of our infrastructure proposal.

Our approach overcomes the design of a generic proposal by describing bigger components because analysis of the possible Semantic Web applications indicates that some combinations of simple components are shared in all of these applications.

Due to the similarity between the two proposals it will be possible to adapt our components for use in the Knowledge Web Semantic Web Framework as soon as the final API is available, and components from this project could be used to produce alternative implementations of our infrastructure.

Another related application is the Onthology (`http://www.onthology.org`) central repository, which also uses OMV to manage ontology metadata. The main focus of this work is to provide a centralized server to manage ontologies that have been accepted in a specific domain. Oyster [Palma et al. 2006] is a complementary application that provides a decentralized system for those users who need to exchange up-to-data information with other users. However, neither system focusses on providing a way of annotating resources with the ontologies managed by their systems.

Universal Description, Discovery and Integration (UDDI, `http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm`) is a platform-independent, Extensible Markup Language (XML)-based registry for businesses worldwide to list themselves on the Internet. UDDI is an open industry initiative, enabling businesses to publish service listings and discover each other and define how the services or software applications interact over the Internet. A UDDI business registration consists of three components: White Pages, Yellow Pages and Green Pages. It is designed to be interrogated by Simple Object Access Protocol (SOAP) messages and to provide access to Web Services Description Language (WSDL) documents describing the protocol bindings and message formats required to interact with the web services listed in its directory. SD-Core is a more generic approach based on semantics to register online resources. In this case the use of semantics allows users to locate resources depending on domain ontologies (that the user should know). This generic infrastructure can also be extended to deal with more specific resources, and in this sense this paper presents SD-Data, an extension to deal with data providing resources.

3 Semantic Directory Core

This section presents the generic infrastructure for the development of Semantic Web applications. The analysis of different architectural proposals and Semantic Web applications makes it clear that a Semantic Web application must have these characteristics:

- Ontologies must be used to introduce semantics.
- As a single and common ontology is not available for most of the domains, ontology management and alignment is necessary.
- Resources are annotated with different ontologies, even in the same domain.
- Resources need to be located by means of the defined semantics.

Summarizing the list of requirements, we can deduce that ontology and resource managers are necessary components for most of the applications. In addition, we can find relationships between ontologies and resources (this is one of the main characteristics of Semantic Web applications). These relationships are rich, and we can take advantage of them in the development of Semantic Web applications.

The proposed infrastructure is based on a resource directory, called Semantic Directory Core, SD-Core (Figure 2). We define the SD-Core as "a set of core elements to build Semantic Web applications, and it is made available as a server to register semantics, providing services to query and browse all the registered semantics". In order to formally define the elements that the SD-Core will manage, we have defined its internal elements by means of metadata ontologies.

SD-Core is semantically described to to allow semantic interoperability with other Semantic Web applications. The semantics will be described in terms of

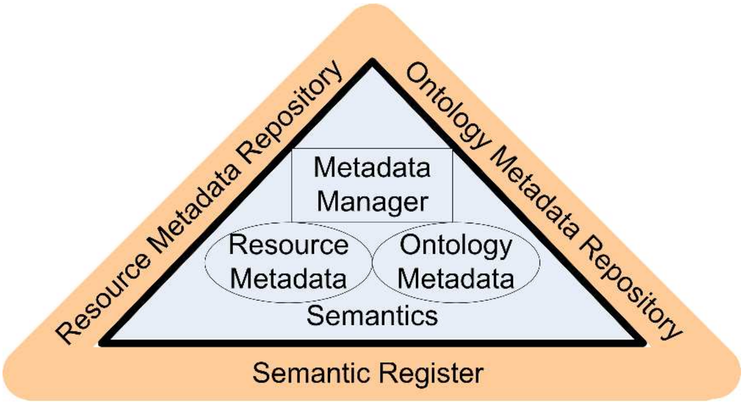

Figure 2: The Semantic Directory Core (SD-Core), a Distributed Semantic Web Infrastructure. The internal elements of the SD-Core could be changed to adapt to different kinds of resources.

two metadata ontologies, *Ontology Metadata* and *Resource Metadata*. These publicly available ontologies are internally managed through a *Metadata Manager*. In order to reach a useful implementation, we have to define these metadata ontologies and also which metadata manager we are going to use in the SD-Core (Figure 2).

The SD-Core is composed of three interfaces (Figure 2), which tends to be the minimum set of elements for building a wide range of applications for the Semantic Web. The SD-Core provides: 1) the *Ontology Metadata Repository* Interface; 2) the *Semantic Register* Interface; and 3) the *Resource Metadata Repository* Interface.

The *Ontology Metadata Repository* interface provides a way of registering ontologies and then locating the useful ones for an application. This first interface offers different types of access to the information on resources related to ontologies registered in the SD-Core. The basic access operation is to search ontologies in the *Ontology Metadata*. Where a reasoner is used as *Metadata Manager* in the SD-Core, the search capabilities will be improved.

The following methods are provided by the *Ontology Metadata Repository* Interface, enabling users to register and browse ontologies:

– *registerOntology*. These are methods for registering ontologies in the SD-Core where the semantics are published using the parameters provided. This information will be included as instances of the Ontology Metadata ontology.

– *getOntology*. This set of methods provides a way of retrieving ontologies registered in the SD-Core.

- *listOntologies*. These methods provide the list of all the registered ontologies or some of them depending on the search parameter provided.

The *Semantic Register* Interface is in charge of relating resources with several of the registered ontologies. When registering a resource, these interface implementations will generate metadata in the *Resource Metadata* containing the relationships between this resource and previously registered ontologies. Thus, it will be possible to take advantage of registered resources by means of the ontologies registered in the SD-Core. Once a resource has been registered, the *SD-Core monitor* (the application in charge of ensuring that all components work correctly) will repetitively test if it is available. In the case where the resource is not available (reachable) it is marked as not available temporarily. Thus, if a user/application asks for resources complying with certain characteristics, unavailable resources' URLs will not be returned. If the SD-Core monitor detects that the resource is available, its state is updated.

This interface provides the following method, which allows the resource owner to register the resource:

- *registerResource*. This method allows users to register resources related with previously registered ontologies.

The *Resource Metadata Repository* Interface is an interface for accessing information about resources, which provides methods for locating resources based on their URL, name, relationships with domain ontologies, etc. The basic access operation is to search resources in the *Resource Metadata*. The following list briefly describes the main methods provided by this interface:

- *getRelatedElements*. This method lists the related concepts of a given concept of an ontology registered in the semantic directory.

- *listResources*. This set of methods enables the location of all the resources or those related with one specific ontology.

4 SD-Data: An Extension of the SD-Core to deal with Data Providing Resources

In order to make use of the SD-Core to deal with data providing resources we have decided to extend the proposed infrastructure to provide additional methods for this kind of application. This specialization is called the SD-Data (Figure 3). The SD-Data has been successfully applied in the development of a Semantic Web framework for data integration: The Khaos Ontology-based Mediation Framework (KOMF). It uses SD-Data for managing semantics, and has been instantiated as a mediator for Systems Biology and used to develop two end-user applications.

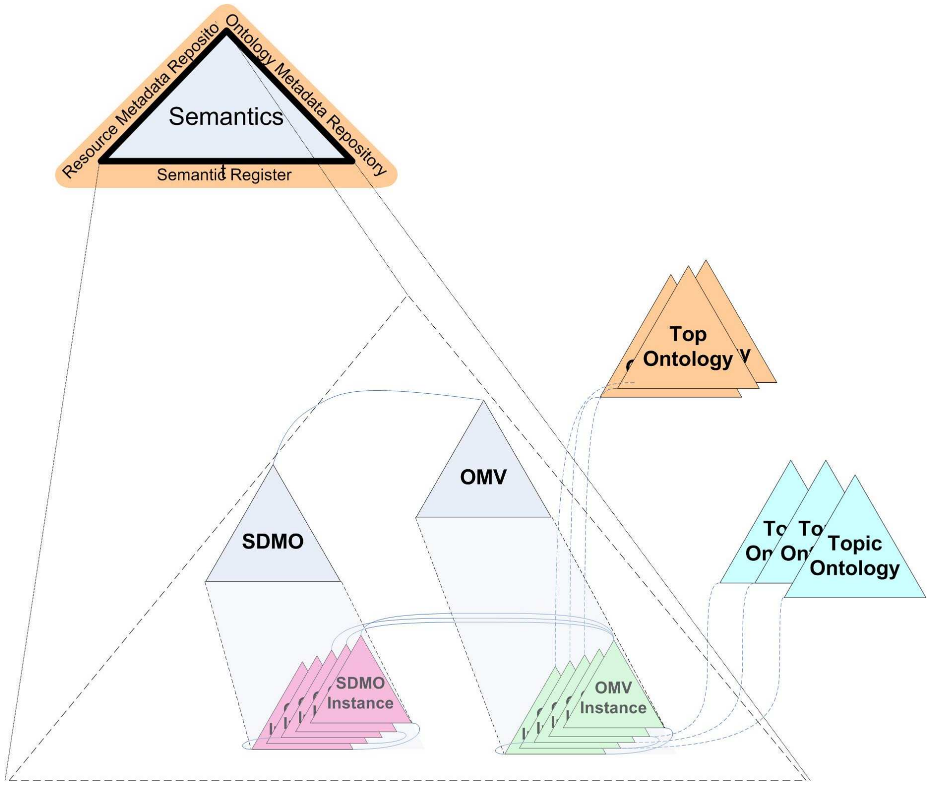

Figure 3: SD-Data Internal Elements. Metadata included as metadata ontologies allows us to define the semantics of resources and ontologies related to them.

4.1 SD-Data Metadata

The SD-Data includes two inter-related ontologies (OMV [Hartmann et al. 2005] and SDMO), which describe the internal semantics of the Semantic Directory (see Figure 3). The main advantage of using ontologies to represent metadata is that this metadata can be managed by tools ranging from a simple ontology parser to a complex ontology reasoner, enabling the use by intelligent agents thanks to the use of explicit semantics. We use OMV to register additional information about ontologies to help users locate and use them (we have registered the ontologies obtained from searches in SWoogle [Ding et al. 2004] and Google). The metadata scheme contains further elements describing various aspects related to the creation, management and use of an ontology.

SDMO is the ontology in charge of registering information about resources and relationships between these resources and ontologies registered in the SD-

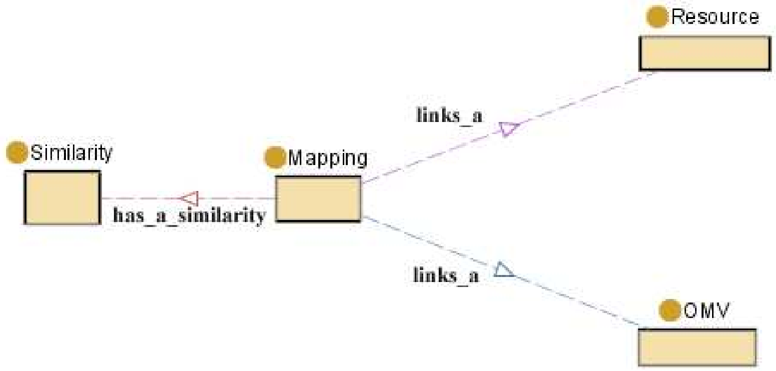

Figure 4: Semantic Directory Metadata Ontology, SDMO. Class "User" is not shown because it is not related with the other classes. This class is used to check user access to the SD-Core.

Data. SDMO and OMV are related by a class included in SDMO, which provides a way of relating resources (SDMO instances) with registered ontologies (OMV instances). The current version of SDMO is composed of five classes (Figure 4): OMV, Resource, Mapping, Similarity and User:

- OMV: this class is used to link resources with registered ontologies (as instances of the OMV ontology). It contains the ontology name and URL.
- Resource: this class is used to store information (query capabilities, schema, query interface, name and URI) about resources.
- Mapping: this class is used to set the relationships between resources and ontologies. Each mapping is related with a similarity instance that establishes the similarity between ontology concepts and resource elements.
- Similarity: the similarity class contains three properties (concept1, concept2 and similarityValue) to establish the similarity between an ontology concept and a resource element.
- User: this class is added in order to deal with users in the applications.

As we have used the OMV metadata ontology in the SD-Data, the registration of ontologies will imply the creation of some OMV class instances. We have chosen *OntologyImplementation* because we assume that ontologies used to annotate resources will be those that have been implemented in a specific language (OWL in our case). This concept has several properties for describing ontologies, and we have selected the most relevant ones for our purposes:

- **implementationName**. This property defines the name of the ontology and the value is retrieved from the input parameters of the registration method. Two different ontologies may have the same name.
- **ontologyURL**. It allows us to set the URL of the ontology. This information is provided in the registration process and it will be used to differentiate two different ontologies.
- **numClasses**. It enables storage of the number of classes contained in the ontology. This information, which is obtained by reading the ontology from the URL provided, could be used for providing statistic data.
- **numIndividuals**. This property defines the number of instances that the ontology has in its definition. This information is obtained directly from the ontology.
- **version**. The version of the ontology could be set using this property. This information can be obtained from the ontology definition. However, not all the ontology providers annotate the version in the same way. For this reason we have developed a heuristic program that tries to find this information in the ontology definition.
- **implementationAcronym**. This property allows us to define a short name for the ontology.
- **imports**. It establishes references to other ontologies. This information is obtained in the registration process from the ontology definition, and will allow us to find explicit relationships between different ontologies.

4.2 SD-Data Design

In order to deal with the characteristics of data providing resources, SD-Data includes some additional methods. Thus, the Semantic Register includes methods for registering resources, indicating which method is suitable for querying, getting the schema and knowing the query capabilities.

The resources registered must have a schema for representing their data. As the current tendency in the Semantic Web is to use XML or RDF as data interchange languages, this kind of resource must provide an *XMLSchema* or an *RDF-Schema*, which will be provided in the registration call. This assumption limits the type of resources that can be registered in the SD-Data. However, resources providing information with OWL instances could also be registered with few changes in the implementation.

These resources have to be data providing resources, so they have a specific schema that can be directly related with one or more registered ontologies.

Additional methods have been added to the *Resource Metadata Repository* to retrieve information related with this kind of resource:

- *getSchema*. This method locates the schema of a resource registered in the directory.
- *listMappings*. These methods list the mappings of a resource with one or more ontologies of the SD-Data. The resources are located taking into account the possible input parameters.

As a first approach for the *Metadata Manager* we have used Jena (`http://jena.sourceforge.net/`). However, access to the registered information is possible using a reasoner like Racer [Baader et al. 2003] or DBOWL [Roldan-Garcia et al. 2008].

The use of this or another reasoner can be carried out using the DIG API (`http://dl.kr.org/dig/`). In this way, another DIG compliant reasoner can be used instead of Racer or DBOWL. The use of a reasoner implies that the system will have an additional overhead because of the reasoner activities. However, this kind of tool can be installed in a separate machine thereby avoiding the system overhead.

SD-Data is considered a Semantic Web application because it provides basic elements for dealing with semantics without any additional component. However, more complex applications can be developed using SD-Data. Thus, Semantic Aware applications use SD-Data to find the semantics of resources registered in them, accessing the information through the existing resources. These resources have to be registered in SD-Data, but this will not involve making changes in them. On the other hand non Semantic Aware applications can directly access the resources.

4.3 SD-Data Use Cases

This section describes some examples of the use of SD-Data. Thus, the result of registering ontologies and resources is shown.

We can register an ontology with the call to "registerOntology" using the URL "`http://mobi.yaco.es/andalucia.rdf/andalucia-tourism.owl`" and the name "AndaluciaTourism". This registration process will generate an instance of the OntologyImplementation concept. This instance will have some datatype property values such as: implementationName (AndaluciaTourism), ontologyURL (http://mobi.yaco.es/andalucia.rdf/andalucia-tourism.owl) and numClasses (15).

However, ontologies with more information will generate instances with more metadata. For example, the registration of the ontology at "`http://www.co-ode-org/ontologies/pizza/2007/02/12/pizza.owl`" will generate an instance with this metadata:

– implementationName (pizza), ontologyURL (`.../pizza.owl`), implementationAcronym (QuatroQueijos@pt), versionInfo (v.1.4. Added Food class ...), numClasses (143) and numIndividuals (5).

In addition, more complex ontologies will provide us with more information. Thus, the registration of an ontology with "import" elements will generate one instance for each "import" element and then the corresponding properties in the original ontology metadata. For example the registration of the ontology at "`http://iridl.Ideo.columbia.edu/ontologies/interface.owl`" will generate the following instances:

– implementationName (interface), implementationAcronym (Contributor@-en-US), ontologyURL (`.../interface.owl`), numClasses (103), numIndividuals (1551), imports (Ont1, Ont2).

– Ont1: implementationAcronym (Rights@en-US), ontologyURL (`http://purl.org/dc/terms`) and numClasses (24).

– Ont2: implementationAcronym (Contributor@en-US), ontologyURL (`.../iriterms.owl`), numClasses (18), numIndividuals (5) and imports (Ont3, Ont4, Ont5).

– Ont3: implementationAcronym (Contributor@en-US), ontologyURL (`.../rdfcache.owl`), numClasses (4) and numIndividuals (1).

– Ont4: implementationAcronym (Contributor@en-US), ontologyURL (`http://purl.org/dc/elements/1.1/`), numClasses (0) and numIndividuals (0).

– Ont5: implementationAcronym (Contributor@en-US), ontologyURL (`.../iricrosswalk.owl`), numClasses (10) and numIndividuals (4).

On the other hand, SD-Data extends the SD-Core to enable additional ways of registering resources. Thus, the different methods provided enable registration at different levels: making the relationships with registered ontologies explicit or allowing SD-Data to calculate them by means of an internal matching tool. The registration of resources will generate instances of the *Mapping* and *Similarity* classes, indicating the relationships between the resource and registered ontologies. This information could be obtained from the input parameters of the registration process, or could be obtained automatically using a matching tool.

For example, we can register a data source with information on different towns in Andalucía. For this we will use the call: *registerResource ("townDescription", "http://www.juntadeandalucia.es/iea/sima/index.htm", "http://-mobi.yaco.es/andalucia.rdf/andalucia-tourism.owl", mappings).*

The mappings vector is a list of similarities in which each similarity is a triple: (*resourceElements*, *ontologyConcepts* and *similarityValue*). *ResourceElements* is

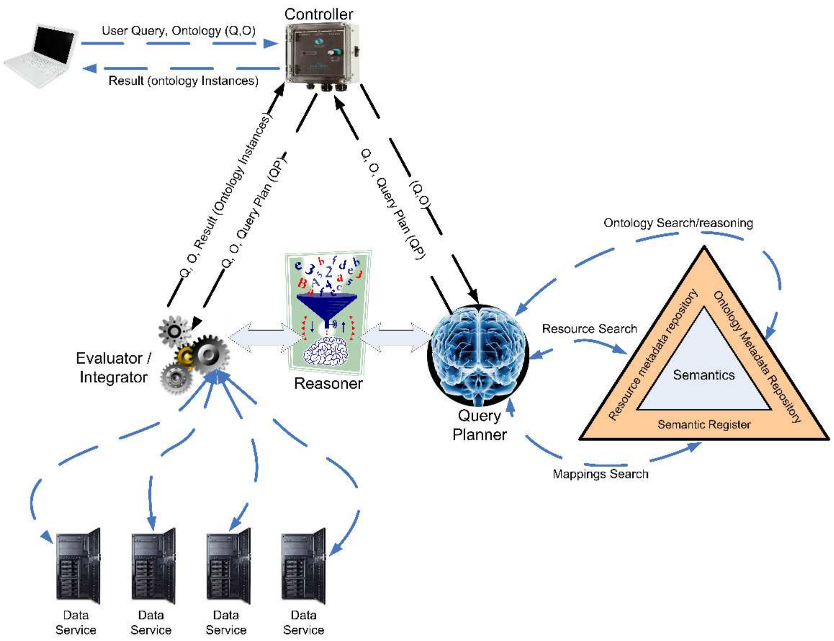

Figure 5: KOMF architecture. The architecture includes as its main elements the Controller, Query Planner and Evaluator/Integrator. A reasoner could be used to improve the query planning and the information integration.

an XPath expression with the part of the resource that is related with the ontology. *OntologyConcepts* is a query expressed in terms of the ontology, and the similarity value expresses how similar the concepts included in this query are with respect to the elements included in *resourceElements*.

5 KOMF: the Khaos Ontology-based Mediation Framework

In this section, we propose an ontology-based mediator framework (the Khaos Ontology-based Mediation Framework, KOMF) which uses SD-Data, a generic infrastructure to register and manage ontologies, their relationships and also information relating to the resources.

In the proposed framework (see Figure 5) our goal is to provide access to the data using a common data model, and a common query language. Our architecture provides a semantically coherent model representation of the combined data from the wrapped data sources and transparent access to the combined data through queries to the mediating view.

In this context, wrappers are an important part of the internal elements of Data Services [Navas-Delgado et al. 2005]. A wrapper accepts queries from the mediator, translates the query into an appropriate query for the individual source, performs any additional processing and returns the results to the mediator. The user/programmer does not need to make individual interfaces for each data source. An interface to a data source is one that translates data into a common data model used by the mediator helping the user to access each data source.

In our case we have chosen XML, in a broad sense: XML, XML-Schema and XQuery, as the common data model. However, it is also possible to deal with RDF (RDFSchema and SPARQL). The development of Data Services requiring the development of a wrapper has been studied previously [Navas-Delgado et al. 2005]. Biological data sources are usually public and downloadable. In these cases we have designed some patterns to retrieve a data source stored as a flat file to store it in an XML database.

Data Services, independently of the development process, are distributed software applications that receive queries in XQuery/SPARQL and return XML/RDF documents. Thus, the process of registering them in a semantic directory implies finding a set of mappings between one or several ontologies and the data service schema (expressed as an XMLSchema/RDFS document). These mappings will be the key elements to integrate all the data sources, and they will be the way resource semantics are made explicit.

As the proposal is to use ontologies to integrate data, we have chosen a GAV (Global as View) approach [Halevy et al. 2001]. In GAV, each source is related to the global schema (ontology in our case) by means of mappings. Moreover, the use of ontologies will allow us to take advantage of reasoning mechanisms to improve the query rewriting.

The mappings we use are defined as pairs (P_t, Q_o). P_t is a set of path expressions on the resource schema, and Q_o a query expression in terms of the ontology. In a first approach we have chosen XPath as the language to express P_t, and conjunctive queries to represent Q_o.

The architecture of the proposed Ontology-Based Mediation Framework (Figure 5) is composed of three main components:

- *The Controller*: This component assumes the role of the infrastructure between all mediator components and interacts with the user interface, providing query results described in terms of one of the ontologies registered in the semantic directory.

- *The Query Planner*: This component elaborates a query plan (QP) for the user query. The planner has been implemented including the most basic reasoning to take advantage of described semantics (subsumption and classification). Thus, if a query includes a concept this query will be expanded to include the semantic descendants. The mappings (stored in SD-Data) are also important in this process and they are used to find if the query pattern matches one or more patterns in the mappings.

– *The Query Solver/Integrator*: This component analyzes the query plan (QP) received from the query planner, and performs the corresponding call to the data services involved in the sub-queries $(S_{Q1}, \ldots, S_{Qn})$ of the query plan $(R_1, \ldots, R_n)$. This component will obtain a set of XML documents from different data services. Results from data services $(R_1, \ldots, R_n)$ are composed by this component, obtaining the results of the user query. This component uses the mappings to translate the XML document to ontology instances, and then a conjunctive query evaluator is applied to the set of instances found.

The use of a reasoner will provide the possibility of improving the planning and integration process taking advantage of the explicitly defined semantics.

The *Controller* provides two different methods:

– **query(userQuery, domainOntology): int**. This method accepts a query and returns the query identifier to check the status of the query. Thus, the implementations of this component will provide an asynchronous behavior to applications using the mediator.

– **status(queryId): int**. This method returns different values for possible status of the query evaluation and can return the following different status:
  - *[0]* Started.
  - *[1]* Planned. The query has been sent to the *Query Planner* and the *Query Planner* has returned a correct query plan.
  - *[2]* Solved. The query plan has been resolved, but results have not yet been integrated and produced as instances.
  - *[3]* Integrated. The sub-query results have been integrated.
  - *[4]* Finished.

The *Query Planner* provides the following method:

– **plan(query): QueryPlan**. This method returns a query plan for a given query.

The *Query Solver/Integrator* provides a method for executing a query plan:

– **execute(queryPlan): Instances**. This method returns a set of instances as a result of executing a query plan.

6 Systems Biology Applications

In the context of Life Sciences, the framework of Systems Biology is being merged [Kitano et al. 2002]. It is supported by all high-throughput methods which generate large amounts of data that simply cannot be processed by the human mind. This field includes a wide variety of concepts and methods but in general, it can be considered the analysis of living systems, through the study of the relationships among the elements in response to genetic or environmental perturbations, with the ultimate goal of understanding the system as a whole. A "system" can be considered at different levels, from a metabolic pathway or gene regulatory network to a cell, tissue, organism or ecosystem. The number of information repositories and services for biological elements (molecules, cells, etc) is growing exponentially. Consequently, Systems Biology is the archetype of a knowledge-intensive application domain for which the Semantic Web should be particularly interesting.

6.1 SB-KOM: The Systems Biology Khaos Ontology-based Mediator

KOMF has been successfully instantiated in the context of Systems Biology for integrating disperse data sources [Navas-Delgado et al. 2008C] and for integrating metabolic information. As shown in Figure 5 the architecture of KOMF is composed of four main components. The following sections describe the implementation of these components for the cited domain.

6.1.1 The Controller

The controller component receives user's requests (coming from from some user applications which perform the role of the mediator client), and evaluates them to obtain the result of their requests. In our approach, the controller creates different threads for different user's requests, and assumes the role of the infrastructure between the mediator components.

Once received, the request is sent to the planner component and one or several (queries) plans (QP) obtained. A query plan is a set of queries to be executed over different data services and information about how to compose them. The controller chooses one and it is sent to the Query Solver/Integrator" component. This component answers the calls to the different data services involved in the plan. Finally, the XML documents obtained in the calls to the evaluator are composed to build the set of instances that are returned to the user application. This component traces the status of each query. Thus, applications can use this trace to inform users on the query resolution.

```
"Ans(E,O,P,S,V) :- Enzyme(E), ecNumber(E,"ecNumber"),
                   CellularOrganism(O),
                   organismName(O,"organism"),
                   belongTo(E,O), Substrate(S),
                   naturalSubstrate(E,S), Product(P),
                   naturalProducts(E,P), EnzymeVariant(V),
                   hasVariant(E,V), belongTo(V,O);"
```

Figure 6: Query Example for retrieving information about enzymes

6.1.2 The Query Planner

This component is by far one of the most fundamental pillars in the action of elaborating one or several query plans and it computes the user query from different data. Plans generated by this component specify the data sources from which the information can be retrieved and in which order they must be accessed.

The evaluation of these queries depends on the query plans themselves and absolutely must be generated expeditiously. Before starting to explain the algorithm we must define the query language for the queries that this component will accept. Queries are expressed as conjunctive predicates, with three main categories: classes, datatype properties that link individuals to data values and object properties that link individuals to individuals. The technique used is based on Sideways Information Passing Strategies (SIPS) [Bancilhon et al. 1986].

According to this language, there will be different types of mappings in the semantic directory. The classes will be connected to the XPath of one or several XML resource elements. On the other hand, datatype properties will be connected to those two expressions: the first one corresponds to the class and the second to the property. The object properties will be related to the active classes XPath in the property. One possible example of a query is that shown in Figure 6.

In this query, we have five classes (*Enzyme*, *Organism*, *Substrate*, *Product* and *EnzymeVariant*), two datatype properties (*ecNumber* and *organismName*) and four object properties (*naturalSubstrate*, *naturalProducts*, *hasVariant* and *belong_to*).

This query will return instances of a class Enzyme whose Ec number is "ecNumber", and which are related to:

- The "organism" CellularOrganism by means of the relation belong_to.
- Some Substrate by means of the relation naturalSubstrate.
- Some Product by means of the relation naturalProducts.

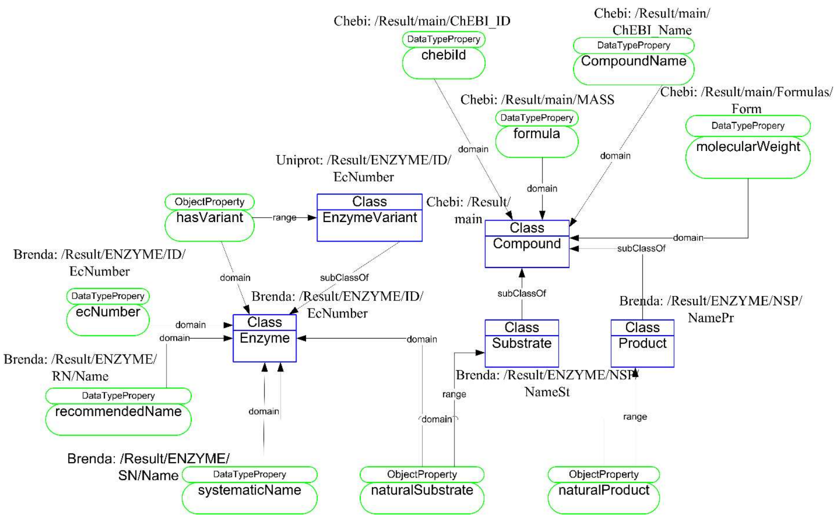

Figure 7: Mappings between the domain ontology and registered resources.

– Some EnzymeVariant by means of the relation hasVariant, this EnzymeVariant has to be related to the same Organism which the enzyme is related to by means of the relation belong_to.

For the query example and mapping shown in Figure 7, the algorithm first finds all query predicates, getting *Enzyme(E)*, *ecNumber(E, "ecNumber")*, *CellularOrganism(O)*, *organismName(O, "organism")* , *belongTo(E, O)*, *Substrate(S)*, *naturalSubstrate(E, S)*, *Product(P)*, *naturalProducts(E, P)*, *EnzymeVariant(V)*, *hasVariant(E, V)*, *belongTo(V, O)*.

Then, the SD-Data is queried to find if the used concepts have descendants. In this case, the query is expanded to search for information related with these descendants. The reasoner is used in this case by the SD-Data to infer the required knowledge.

Then the predicates which have common parameters are grouped together. This produces a large set of combinations such as:

– *Enzyme(E), ecNumber(E,"ecNumber")*

– *Enzyme(E), belongTo(E,O)*

– *Enzyme(E), naturalSubstrate(E,S)*

– *Enzyme(E), naturalProducts(E,S)*

– *Enzyme(E), hasVariant(E,V)*

– *Enzyme(E), ecNumber(E,"ecNumber"), belongTo(E,O)*

– *Enzyme(E), ecNumber(E,"ecNumber"), naturalSubstrate(E,S)*

– *Enzyme(E), ecNumber(E,"ecNumber"), naturalProducts(E,S)*

– *Enzyme(E), ecNumber(E,"ecNumber"), hasVariant(E,V)*

– *Enzyme(E), ecNumber(E,"ecNumber"), belongTo(E,O), naturalSubstrate(E,S)*

– *Enzyme(E), ecNumber(E,"ecNumber"), belongTo(E,O), naturalProducts(E,S)*

– *Enzyme(E), ecNumber(E,"ecNumber"), belongTo(E,O), hasVariant(E,V)*

– *Enzyme(E), ecNumber(E,"ecNumber"), belongTo(E,O), naturalSubstrate(E,S), naturalProducts(E,S)*

– *Enzyme(E), ecNumber(E,"ecNumber"), belongTo(E,O), naturalSubstrate(E,S), hasVariant(E,V)*

– *Enzyme(E), ecNumber(E,"ecNumber"), belongTo(E,O), naturalSubstrate(E,S), naturalProducts(E,S), hasVariant(E,V)*

– ...

However, the planning algorithm filters those groups that are not present in the SD-Data mappings, by querying the repository that will provide the data elements related with the given concepts. Thus, in this example we obtain the following groups:

– C1: *Enzyme(E), ecNumber(E,"ecNumber")*

– C2: *CellularOrganism(O), organismName(O,"organism")*

– C3: *Substrate(S)*

– C4: *Product(P)*

– C5: *EnzymeVariant(V)*

– C6: *naturalSubstrate(E,S)*

– C7: *naturalProducts(E,P)*

– C8: *hasVariant(E,V)*

– C9: *belongTo(V,O)*

– C10: *belongTo(E,O)*

From these (C1 to C10) groups, the planner selects the group that has some variable instanced (in the example, the group C1 would be selected for having the *ecNumber* instanced with the value "ecNumber").

The group selected in the previous step will be considered as the root of the possible planning tree. The order of execution of the groups will depend on the instanced variable: first, it executes the groups which contain instanced variables, then the groups that relate the instanced variable to other groups and so on.

For this example, the plan will execute the group C1 from which we will determine all the *Enzymes* that have *ecNumber* equal to "ecNumber". Then the groups C10, C8, C7 and C6 will be executed in parallel to be linked to the instanced variable in the previous step (the variable E in that case). From these simultaneous executions, the algorithm determines all objects that are respectively related to Enzyme by means of the relationships *belong_to*, *hasVariant*, *naturalSubstrate* and *naturalProducts*.

Once those objects are obtained, they will be checked to see if they satisfy the groups C2, C5, C4, and C3 respectively; in our case, the algorithm will verify whether the objects obtained from C10, C8, C7 and C6 are of the type *CellularOrganism*, *EnzymeVariant*, *naturalSubstrate* and *naturalProducts* respectively.

A possible plan of execution for our query could be that represented in Figure 8. The arcs represent object properties: *belong_to*, *hasVariant*, *naturalSustrate* and *naturalProducts* in this case.

Each node and arc (Figure 8) contains the following information: the XQuery query (produced from the mappings) corresponding to the sub-query of the data sources node or arc, name and URL.

6.1.3 The Query solver/Integrator

A query plan describes the order of execution of the local queries. The plan contains all the information needed by the Evaluator/Integrator to execute a local query; this information includes the resource URI and the local query expressed in XQuery. To answer a user query, this component first executes the data services in the order specified by the query plan. Then, it generates the instances from the data service results. To minimize the query execution time, we can run the local queries concurrently. This is possible when the local queries are independent of each other. We say that a query Q_j depends on another query Q_i if Q_j is a parameterized query whose parameters take values from the result of executing Q_i.

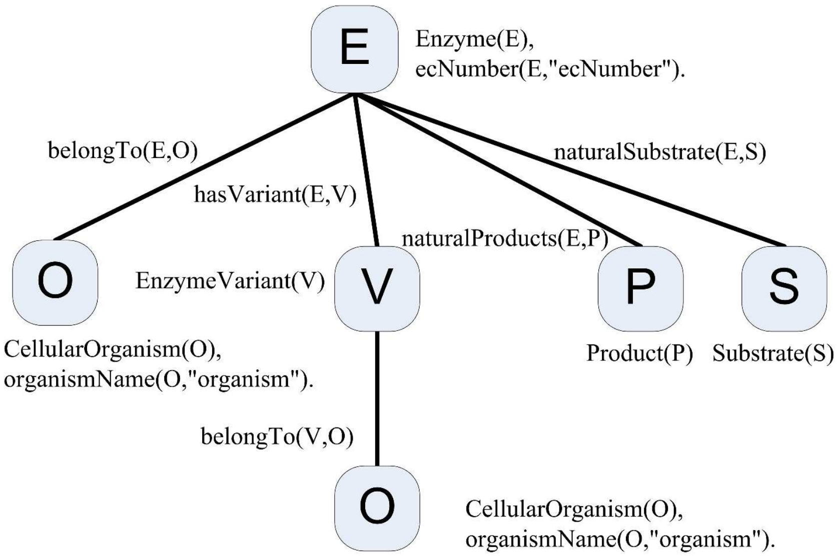

Figure 8: Query plan for the query "*Ans(E,O,P,S,V) :- Enzyme(E), ecNumber(E,"ecNumber"), CellularOrganism(O), organismName(O, "organism"), belongTo(E,O), Substrate(S), naturalSubstrate(E,S), Product(P), naturalProducts(E,P), EnzymeVariant(V), hasVariant(E,V), belongTo(V,O);*". Each node and arc contains information to access the corresponding data service.

For example, to execute the first local query (nodes E and O) the evaluator needs to know the resource URI (http://khaos.uma.es/brenda/services/brenda) and the XQuery query.

In the sample query plan (Figure 6), the query of node V depends on the query of node *E*, the query of node *P* depends on the query of node *E* and the query of node *S* depends on the query of node *E*. So, the queries of nodes *V*, *P* and *S* must be executed after the query of node *E*. However, queries *V*, *P* and *S* are independent of each other and can be executed concurrently improving the system performance.

The sample query has values for datatype properties of *Enzyme* and *CellularOrganism* classes. As shown in the query plan, the first data service to be executed is the one that corresponds to nodes *E* and *O* (it involves two nodes because it needs the values of two concepts to be executed); then the other data services are executed concurrently. The Object Properties *hasVariant*, *naturalProduct* and *naturalSubstrate* are used to retrieve concurrently values for variables *V*, *P* and *S* from the first data service result. Each variable will have a list of values. Obviously, queries parameterized with different values are independent, so the execution of a data service for different values can be done concurrently. As shown in the query plan, the data services that correspond to variables V, P and S are independent of each other. So the three data services for all value lists are executed concurrently.

The mediated schema used in our mediator is an ontology. Therefore the results must be returned to the user as ontology instances. Since in our architecture the component responsible for executing local queries is a Data Service, the results of executing the queries included in the plan are XML documents. Once the result is retrieved, the XML document is translated to ontology instances (RDF model), using the mappings between the XML elements and the OWL ontology. The translation process is based on mappings. The system supports two types of associations in the mappings: one-to-one and one-to-many. The one-to-one association means that a resource in the ontology is mapped with an XPath in the data service schema. On the other hand, the one-to-many association means that a resource in the ontology is mapped to several XPaths in the data service schema.

Based on related work in Ontology Mapping (such as [Ehrig et al. 2004]), we have defined three kinds of mappings: class mapping, datatype property mapping and object property mapping:

– Class Mapping: relates a class to an XML element. This means that for each XML element, in a data service result, mapped to a class an instance of this class is created. The use of a reasoner enables the extension of these mappings with the inferred class hierarchy.

– Datatype Property Mapping: relates a datatype property to an XML element. As properties can have more than one domain class, in this kind of mapping we need to specify the domain class to use. For each XML element, in a data service result, mapped to a datatype property an instance of this datatype property is created. The datatype property value is the text value of the XML element and the domain is an instance of the class declared in the mapping as domain class. The use of a reasoner allows the algorithm to detect when a property can be applied to a class even when it has not been explicitly defined in it.

– Object Property Mapping: this kind of mapping relates two classes involved in class mappings (already declared) to an object property name. For example, consider another class mapping that relates "enzyme" class to "/Result/ENZYME" element, and an object property mapping that relates "enzyme" and "NSP" to the object property named "has". This component generates an "enzyme" instance E and two "NSP" instances N1 and N2, and also it generates two "has" object property instances h1 and h2. h1 has E as domain and N1 as range, and h2 has E as domain and N2 as range. To determine which instance is the domain of a property instance, this component

```
<?xml version=1.0 encoding=UTF-8?>
<Result>
  <ENZYME>

    <ID>
      <EcNumber>2.3.1.57</EcNumber>
    </ID>

    <NSP>
       <NameSt>acetyl-CoA</NameSt>
       <NameSt>spermidine</NameSt>
       <NamePr>CoA</NamePr>
       <NamePr>N1-acetylspermidine</NamePr>
       <Organism>Homo sapiens</Organism>

    </NSP>
    <NSP>
       <NameSt>acetyl-CoA</NameSt>
       <NameSt>spermine</NameSt>
       <NamePr>CoA</NamePr>
       <NamePr>N1-acetylspermine</NamePr>
       <Organism>Homo sapiens</Organism>

    </NSP>

</ENZYME>
```


Figure 9: Part of the result returned by the data service corresponding to nodes E and O

uses relative paths as described above for the datatype property mapping. In this case, the use of a reasoner also allows the algorithm to detect when a property can be applied to a class that does not contain this property.

For the example shown in Figure 9, consider that we have a class mapping that maps a class "NSP" to "/Result/ENZYME/NSP" element and a datatype mapping that maps a datatype property "name" to "/Result/ENZYME/NSP/NameSt" element. Using these mappings, the component creates two "NSP" class instances N1 and N2, and four "name" datatype property instances n1 and n2 with N1 instance as domain and "acetyl-CoA" and "spermidine" as values, and n3 and n4 with N2 instance as domain and "acetyl-CoA" and "spermine" as values. As we have seen in this example, there is a need to determine which instance (N1 or N2) is the domain of each datatype instance (n1, n2, n3 and n4), the component uses relative paths to achieve this purpose. In this example the relative path between "/Result/ENZYME/NSP/NameSt" and "/Result/ENZYME/NSP/" is "../".

In an XML document it is usual that a value appears more than once for the same or different XPaths. But in an instance graph, the value of a property for an instance must appear only once. For both cases, when the value appears more than once for the same XPath with a one-to-one association or when the value appears more than once for different XPaths with a one-to-many association, the library deals with the situation and generates just one instance for this value.

The last, but not least important, characteristic of the library is related to inheritance. For both datatype properties and object properties specified for a class, if a mapping is specified for one of its subclasses, all the subclass instances generated must have values for the superclass properties (if possible).

6.1.4 Data Services

In our proposal, the sources are made available by publishing them as Web Services (named Data Services). Our primary goal here is to integrate databases accessible via internet pages. In this context, wrappers are an important part of the internal elements of data services. Data services, independently of the development process, are distributed software applications that receive queries in XQuery and return XML documents.

In the context of mediator development, the process of registering resources in a SD-Core implies finding a set of mappings between one or several ontologies and the data service schema (usually expressed as an XMLSchema document). These mappings will be the key elements to integrate all the data sources, and these mappings will be the way in which the resource semantics are made explicit.

7 Discussion

The Knowledge Web European project (`http://knowledgeweb.semanticweb.org`) has designed a Semantic Web Framework, which describes the main elements that Semantic Web applications will require and the type of applications that can be developed by using this framework. Thus, the main aim of this project is to identify and classify components (developed in Universities and Businesses) that could be useful for developing Semantic Web applications. The implementation of the identified components would solve the software developer problem of finding Semantic Web components and their interconnections.

This framework classifies the components in dimensions, chosen as a result of the developers' experience. The following dimensions are considered: *Data and metadata management*, *Querying and reasoning*, *Ontology development and management*, *Ontology customization*, *Ontology evolution*, *Ontology alignment*, *Ontology instance generation* and *Semantic web services*.

Each component is described by defining its dependencies with other components, and then a list of use cases is presented. Each use case describes how several components of the framework can be composed to solve a specific problem. From these use cases and the component dependencies, we can deduce that some blocks of components can be grouped, because they usually act together. In addition, their capabilities are almost the same as the ones provided by our infrastructure. These common blocks are the Ontology and Data repositories, whose definitions are [Leger et al. 2007]:

- 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.

The Ontology repository component is supposed to provide a defined protocol to access ontologies, enabling query support though standard languages. In addition, the repository should have some quality characteristics: provide fault tolerance mechanisms, implement caching mechanisms, manage change propagation, scale and allow management access.

The functionalities of the Ontology repository component are provided in our proposal by the *Ontology Metadata Repository* Interface. This interface allows users to locate registered ontologies. As the system stores the URLs to publicly available ontologies, the changes are directly updated when the source ontology is changed. In addition, URLs are tested before being sent to users to prevent providing ontologies that are temporarily inaccessible. Besides, the proposed interface provides methods for ontology providers to manage the registration of their ontologies.

On the other hand, the Data repository component is covered by the *Semantic Register Interface* and the *Resource Metadata Repository* Interface, which provide a defined protocol to access the resources, offering fault tolerance mechanisms, information propagation when resources are changed and management mechanisms for resource providers.

Another component that usually acts together with the previous ones is, the Metadata repository, defined as:

- Metadata registry component. This component provides functionalities to locally store and access metadata information.

These three components have also the interesting feature that they do not depend on any other component and appear as necessary elements for most of the simple components described. Therefore, they can be considered the core of any Semantic Web Application.

The Semantic Web Framework [Leger et al. 2007] is the most similar work to our proposal. However, whereas it is in its initial stage (framework design), we have successfully applied our infrastructure to create Semantic Web applications in real scenarios. The Semantic Web Framework has a structure in which applications are described using simple components. Our infrastructure overcomes this

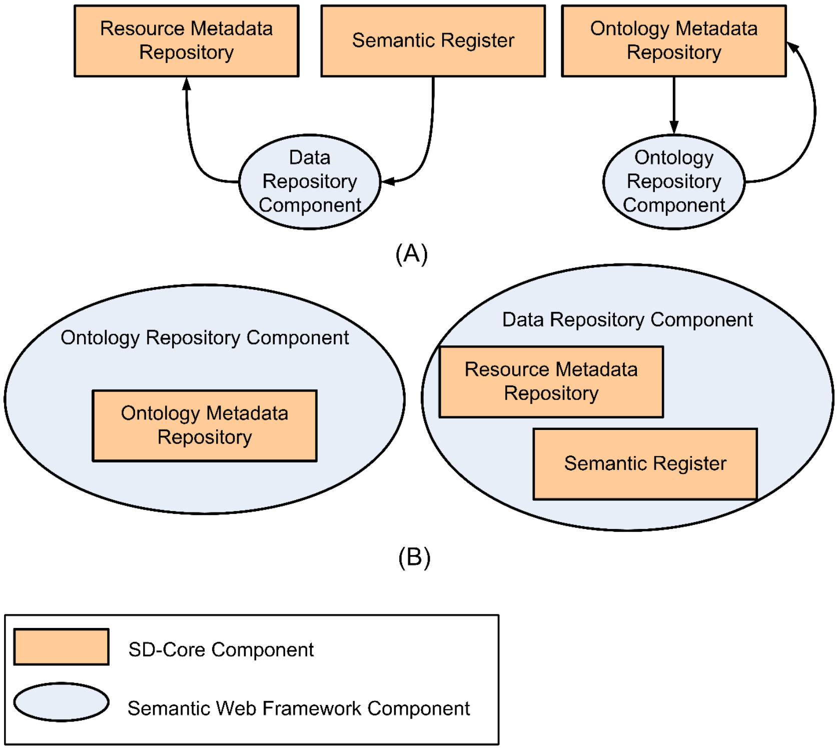

Figure 10: Relationship between SD-Core and the Semantic Web Framework. (A) The interfaces of the SD-Core could be implemented using components from the Semantic Web Framework. (B) Some of the components of the Semantic Web Framework could be implemented using the SD-Core implementation.

goal by describing bigger components because the analysis done by the Knowledge Web Project of the possible Semantic Web applications indicates that some combinations of simple components are shared in all of these applications.

Figure 10A shows how the SD-Core could be implemented using components from the Semantic Web Framework. Thus, methods of the SD-Core will be implemented as calls to these components. On the other hand Figure 10B shows the way in which some components of the Semantic Web Framework could be developed using the SD-Core implementation.

8 Conclusions

The main goal of this article has been to describe the design of the different elements that comprise our infrastructure and how it can be used to solve real problems. Thus, we have described the SD-Core and the design of the Semantic Directory concept. This core defines the minimum set of components required to implement any Semantic Web application. Its architecture is the result of the experience derived from (our) previous work[2].

This article describes an extension of SD-Core to include new capabilities by adding new methods (without adding new interfaces), producing SD-Data, for dealing with data providing resources. This extension has been used to design a framework that takes advantage of SD-Data to build ontology-based mediators. It will allow applications to take advantage of data by providing resources to reuse a common structure.

Using SD-Data we have designed a generic framework for developing ontology-based mediators, KOMF. The main advantage of using SD-Data is that the mediators developed do not have to deal with the problems of managing relationships between data providing resources and domain ontologies. Moreover, the architecture of KOMF allows developers to reuse components to develop ontology-based mediators.

The development of the mediation solution as a framework will enable the improvement of future versions, producing progress in this area of research by taking advantage of a working system. Thus, we aim to incorporate the use of further reasoning to optimize query plans, and the integration of other kinds of data sources (i.e. knowledge-bases with reasoning capabilities).

The usability of the interfaces described in this chapter has been tested by developing several tools. Thus, we describe both the instantiation of KOMF for Systems Biology, and the development of two different tools (the ASP Model Finder and the Systems Biology Metabolic Modeling), both of which take advantage of this mediator.

The ASP Model Finder has tackled the resolution of a well known bioinformatics problem by integrating a limited but increasingly growing number of databases. The initial problem to be solved is summarized as follows, and the use case developed to solve it is named AMMO-Prot: "A common and useful strategy to determine the 3D structure of a protein, which cannot be obtained by its crystallization, is to apply comparative modelling techniques. These techniques start working with the primary sequence of the target protein to finally predict its 3D structure by comparing the target polypeptide to those of solved homologous proteins [Baker et al. 2001]."

The System Biology Metabolic Modeling assistant (Figure **??**) is a tool developed to search, visualize, manipulate and annotate identity data and to assist in annotating the kinetic data.

[2] this software will be made available as Open Source

Acknowledgment

Supported by Grants TIN2008-04844 (Spanish Ministry of Education and Science), and P07-TIC-02978 (Innovation, Science and Enterprise Ministry of the regional government of the Junta de Andalucía).

References

[Baader et al. 2003] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2003.

[Baker et al. 2001] David Baker and Andrej Sali. Protein structure prediction and struc- tural genomics. Science, 294(5540):93-96, October 2001.

[Bancilhon et al. 1986] Francois Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D Ullman. Magic sets and other strange ways to implement logic programs. In Proceedings of the fifth ACM SIGACT-SIGMOD symposium on Principles of database systems, pages 1-15, New York, NY, USA, 1986. ACM.

[Beneventano et al. 2000] Domenico Beneventano, Sonia Bergamaschi, Silviana Castano, Al- berto Corni, R. Guidetti, G. Malvezzi, Michele Melchiori, and Mau- rizio Vincini. Information integration: The momis project demon- stration. In Proceedings of the 26th Intrnational Conference on Very Large Data Bases, pages 611-614, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[Bornhovd et al. 1999] Christof Bornhovd and Alejandro P. Buchmann. A prototype for metadata-based integration of internet sources. In Proceedings of the Conference on Advanced Information Systems Engineering, pages 439-445, 1999.

[Ding et al. 2004] Li Ding, Rong Pan, Tim Finin, Anupam Joshi, Yun Peng, and Pranam Kolari. Finding and Ranking Knowledge on the Semantic Web. In the Proceedings of the 4th International Semantic Web Conference, November 2005.

[Ehrig et al. 2004] M. Ehrig and Y. Sure. Ontology mapping - an integrated approach. Proceedings of the European Semantic Web Conference, Heraklion, Greece, 2004; 76-91.

[Garcia-Molina et al. 21997] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The tsimmis aproach to mediation: Data models and languages. Journal of Intelligent Information Systems, 8(2):117-132, 1997.

[Haas et al. 1996] L. Haas, D. Kossmann, E. Wimmers, and J. Yang. An optimizer for heterogeneous systems with nonstandard data and search capabilities. Data Engineering Bulletin, 19:37-44, 1996.

[Halevy et al. 2001] Alon Y. Halevy. Answering queries using views: A survey. VLDB Journal: Very Large Data Bases, 10(4):270-294, 2001.

[Hartmann et al. 2005] Jens Hartmann, York Sure, Peter Haase, Raul Palma, and Maria del Carmen Suarez-Figueroa. Omv - ontology metadata vocabulary. In ISWC 2005 - In Ontology Patterns for the Semantic Web, 2005.

[In Silico Discovery 2010] In Silico Discovery. Semantic discovery system (2010) (last visit January, 2010). http://www.insilicodiscovery.com.

[Kellenberger et al. 2006] Esther Kellenberger, Pascal Muller, Claire Schalon, Guillaume Bret, Nicolas Foata, and Didier Rognan. sc-pdb: an annotated database of druggable binding sites from the protein data bank. Journal of Chemical Information and Modeling, 46(2):717-727, 2006.

[Kitano et al. 2002] Hiroaki Kitano. Systems biology: a brief overview. Science, 295(5560):16621664, March 2002.

[Ksiezyk et al. 2001] Tomasz Ksiezyk, Gale Martin, and Qing Jia. Infosleuth: Agent-based system for data integration and analysis. In Proceedings of the 25th International Computer Software and Applicartions Conference on Invigorating Software Development, page 474, 2001.

[Langegger et al. 2008] Andreas Langegger and Wolfram W and Martin Blchl. A Semantic Web Middleware for Virtual Data Integration on the Web. In the Semantic Web: Research and Applications 2008. Tenerife, Canary Islands, Spain, June 1-5, 2008.

[Leger et al. 2007] Alain Leger, Asuncion Gomez-Perez, Diana Maynard, Dominik Zyskowski, Jens Hartmann, Jerome Euzenat, Martin Dzbor, Michal Zaremba, Maria del Carmen Suarez-Figueroa, Raul Garcia-Castro, Raul Palma, Stamatia Dasiopoulou, Stefania Costache, and Tomas Vitvar. Architecture of the semantic web framework. Technical report, February 2007.

[Mena et al. 1996] Eduardo Mena, Arantza Illarramendi, Vipul Kashyap, Amit P. Sheth: OBSERVER: An Approach for Query Processing in Global Information Systems Based on Interoperation Across Pre-Existing Ontologies. Distributed and Parallel Databases 8(2): 223-271 (2000).

[Navas-Delgado et al. 2005] Ismael Navas-Delgado, Maria del Mar Roldan-Garcia, Daniel Dianes-Mazorra, and Jose Francisco Aldana-Montes. Developing data services. In Proceeding of the 17th Conference on Advanced Information Systems Engimeering. Data Integration and the Semantic Web, DISWeb, pages 287-301, Oporto, Portugal, March 2005. ISBN 972-752-077-4.

[Navas-Delgado et al. 2008] Ismael Navas Delgado and José Francisco Aldana Montes. SD-Core: Generic Semantic Middleware Components for the Semantic Web. In Proceedings of KES 2008.

[Navas-Delgado et al. 2008B] Ismael Navas Delgado, Amine Kerzazi, Othmane Chniber and José Francisco Aldana Montes. SD-Core: A Semantic Middleware Applied to Molecular Biology. In Proceedings of OTM Workshops 2008.

[Navas-Delgado et al. 2008C] Ismael Navas-Delgado, Ral Montaez, Almudena Pinongeles, Aurelio A Moya-Garca, Jos Luis Urdiales, Francisca Snchez-Jimnez, and Jos F Aldana-Montes. AMMO-Prot: amine system project 3D-model finder. BMC Bioinformatics. 2008; 9(Suppl 4): S5.

[OpenLink Software 2010] OpenLink Software. OpenLink Virtuoso (last visit January 2010). http://www.openlinksw.com/virtuoso/

[Palma et al. 2006] Raul Palma, Peter Haase, and Asunción Gómez-Pérez.Oyster: sharing and re-using ontologies in a peer-to-peer community. In Proceedings of the 15th international conference on World Wide Web. Edinburgh, Scotland, 2006.

[Prudhommeaux 2007] E. Prudhommeaux: Federated SPARQL (May 2007), http://www.w3.org/2007/05/SPARQLfed/.

[Risch et al. 2001] Tore Risch and Vanja Josifovski. Distributed data integration by object-oriented mediator servers. Concurrency and Comoutation: Practice and Experience, 14:1-21, 2001.

[Roldan-Garcia et al. 2008] Maria M. Roldan-Garcia and Jose F. Aldana-Montes. Dbowl: Towards a scalable and persistent owl reasoner. In Proceeding of the Internet and Web Applications and Services, 2008. ICIW '08. Third International Conference on, pages 174-179, June 2008.

[Sattler et al. 2005] K.U. Sattler and I. Geist and E. Schallehn: Concept-based querying in mediator systems. The VLDB Journal 14(1), 97111 (2005).

[Tomasic et al. 1997] Anthony Tomasic, Remy Amouroux, Philippe Bonnet, Olga Kapit- skaia, Hubert Naacke, and Louiqa Raschid. The distributed information search component (disco) and the world wide web. In Proceeding of the 1997 ACM SIGMOD International Conference on Management of Data, pages 546-548, 1997.