UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

ESCUELA DE INGENIERÍA

Departamento de Ingeniería en Sistemas Computacionales

ADMINISTRACIÓN SEMÁNTICA DE CONOCIMIENTO E INFORMACIÓN EN EL WEB SEMÁNTICO

Tesis para obtener el grado de

Licenciatura en Ingeniería en Sistemas Computacionales

Héctor Manuel Pérez Urbina

Asesora:	**Dra. Genoveva Vargas Solar**
Co-Asesor:	**Dr. Gerardo Ayala San Martín**
Sinodal:	**Dr. José Luis Zechinelli Martini**

UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

ESCUELA DE INGENIERÍA

Departamento de Ingeniería en Sistemas Computacionales

SEMANTIC KNOWLEDGE AND INFORMATION MANAGEMENT IN THE SEMANTIC WEB

Tesis para obtener el grado de

Licenciatura en Ingeniería en Sistemas Computacionales

Héctor Manuel Pérez Urbina

Asesora:	**Dra. Genoveva Vargas Solar**
Co-Asesor:	**Dr. Gerardo Ayala San Martín**
Sinodal:	**Dr. José Luis Zechinelli Martini**

“Knowledge is power”

Edward Feigenbaum

1994 ACM Turing Award Winner

To Furpa, Murma, Gunter and Purme.

Acknowledgements

To my parents to whom I owe my education, my faith, my life. Thank you for teaching me that the best way to success is learning of your defeats. "Vive" lives in me because of you furpas. I shall never forget it.

To the five I could never forget: Inge, Dayana, Katy, Alma and Abi. We have lived so many things together and conquered so many battles. Thank you for all these years, thank you for being my friends no matter what.

To the friends I fortunately found here: Isaac, Ale, Lalo, Rochy, Erick and Rich. Thank you for all your help, your support, your company and your laughter. You all made my stay in the university the best time I have ever had.

To Ale Bandala, for her understanding and patience, for her support and love.

To Gennaro Bruno, for being more than an excellent guide, for being my friend. This work would not be possible without your help and advice. Je me rappellerai toujours du tigre mon ami, merci beaucoup.

To Dr. Genoveva Vargas, for her infinite patience and advice, for her compromise and dedication. Thank you for your encouragement, for showing me that the only way to do something is doing it well.

Index

1 Introduction

This work has as academic framework the Administración de Datos y del Conocimiento and the Interfaz Humano-Computadora groups of the Centro de Investigación en Tecnologías de Información y Automatización (CENTIA) at UDLAP and the Networked Open Database Services (NODS) group of the Logiciels Systèmes Réseaux (LSR-IMAG, UMR 5526) laboratory in Grenoble, France.

1.1 Context and motivation

Currently there is an amazing quantity of heterogeneous information distributed over a large number of sources. Retrieving information is becoming a difficult task in which regular users use their knowledge in terms of formats, query languages and data models to obtain acceptable results. Instead of having to use many tools to retrieve different kinds of information users prefer having a single tool that allows managing heterogeneous information that comes from different sources.

Interesting information may come from the Web. Although current Web pages structure allows navigation through hyperlinks and specifies some annotated information (e.g. keywords) it still cannot provide the means to exploit knowledge on a large scale. The problem with the majority of data in the Web is that it is difficult to use because there is no global system for publishing data in such way as it can be easily processed by anyone.

Current Web has significant weaknesses related to searching and extracting information. Current keyword based searches can retrieve irrelevant information. Besides, human browsing and reading is required to extract relevant information from information sources. As explained in [DFH03] the Web can reach its full potential only if it becomes an environment where data can be shared and processed by automated tools as well as by people.

The Semantic Web is an extension where human readable documents are annotated with information that will bring together an extremely large amount of human knowledge and will complement it with computers being able to process it. The Web is an extremely large

collection of information or resources; therefore the Semantic Web will become an extremely large collection of annotated resources. These resources could be used by applications by means of data mediation techniques.

1.2 Problem statement

A data mediation system based on semantics can be built on top on the Semantic Web in order to allow applications to have transparent access to Semantic Web resources. The specification and construction of such a mediation system implies describing the semantics of the resources and the semantics of the application requirements in terms of its specific domain. In order to provide a global view on the Semantic Web resources it is necessary to define data models that integrate the semantic representation of resources and application requirements.

1.3 Objective and methodology

Our objective is to build a data mediation system based on semantics that allows applications to have transparent access to distributed, autonomous and heterogeneous sources composed by Semantic Web resources.

1.3.1 Methodology

In order to accomplish our objective we used the following methodology:

- Study of semantic representation mechanisms of resources based on the Semantic Web approach.
- Study of data mediation technology.
- Specification of a mediation system based on semantics by defining its architecture and its functions.

- Specification of a mediator building framework capable of providing the means to create mediation systems from a set of requirements.
- Building of a data mediation system adapted to a specific application context. We consider Computer Assisted Instruction (CAI) as our application context.

1.3.2 Towards SKIMA

SKIMA (Semantic Knowledge and Information Management) is the mechanism we propose as a data mediation system for the Semantic Web that enables applications to have transparent access to sources. Figure 1.1 presents its general approach.

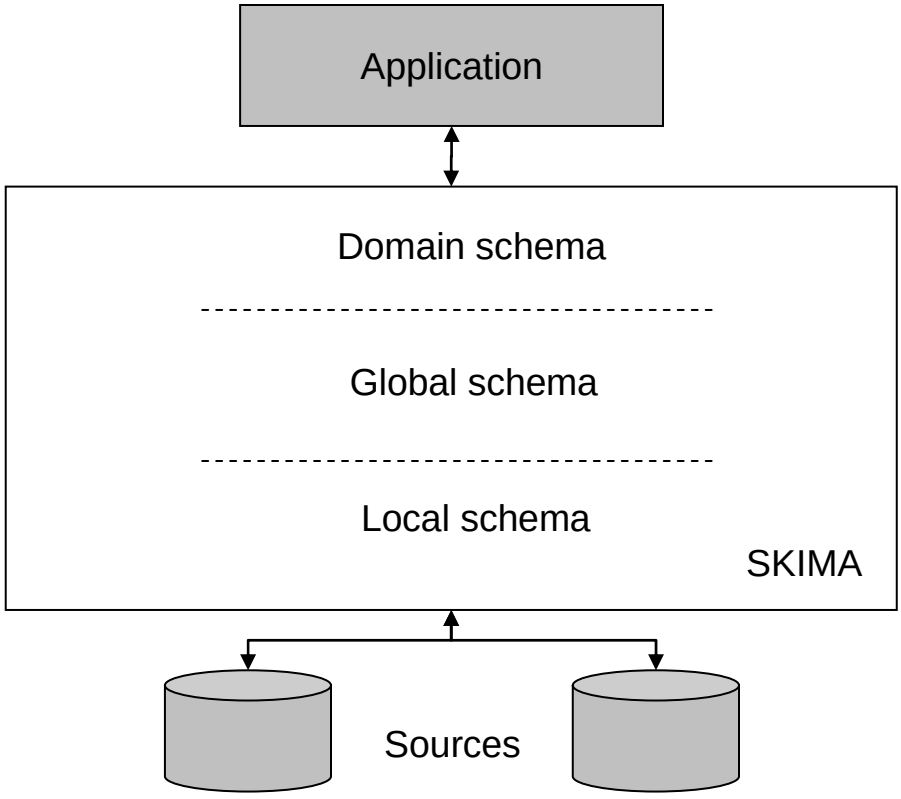


Figure 1.1 SKIMA general approach.

The system is composed by three components: the domain schema, the global schema and the local schema. The domain schema represents a specific application domain, while the local schema integrates semantic representation of resources. On the other hand the

mediation system provides an integrated and global view of sources content and couples it to the description of the application domain through the global schema.

Our approach also considers the creation of a mediator building framework (MBF). MBF is used to build ad hoc mediation systems given a schema specification that includes preferences about the domain and the sources. Figure 1.2 depicts MBF as a mechanism that builds a mediation system from a set of requirements.

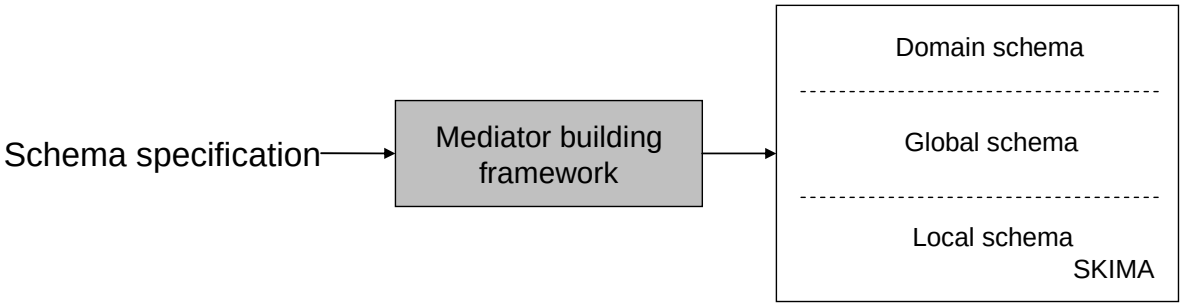


Figure 1.2 MBF general approach.

1.4 Organization of the document

Chapter 2 presents the basic concepts of the Semantic Web in terms of its basic components and the current technological efforts that are being made in this field. Chapter 3 describes SKIMA the mediation system we propose, its architecture, its functions, the data model it implements and the way applications can use it. Chapter 4 describes MBF a system used to build ad hoc mediation systems given a set of requirements. Chapter 5 describes our experimentation that consists in the development of a specific application related to CAI. Chapter 6 presents our results and our perspectives for future work.

2 Semantic Web

This chapter introduces the basic concepts for defining the problems to consider and possible solutions for building the Semantic Web. The remainder of this chapter is organized as follows. Section 2.1 presents Tim-Berners Lee's Semantic Web stack [Be04] to explain the Semantic Web architecture. Section 2.2 focuses on description logics (DL), DL systems and DL languages. Applications used to edit and visualize ontologies are described in section 2.3. Finally section 2.4 presents our conclusions.

2.1 The Semantic Web stack

"The Semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation" [BHL01]. It is an extension where human readable documents are annotated with metainformation (i.e. information of information) that will bring together an extremely large amount of human knowledge and will complement it with computers being able to process it. It is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners as the next step of the evolution of the current Web towards knowledge exploitation [W3C].

For the Semantic Web to function computers should have access to structured meta information collections and to a set of inference rules that can be used to develop automated reasoning [Pa01, BHL01]. Figure 2.1 shows the Semantic Web stack: a description of the Semantic Web in terms of its principal aspects and components.

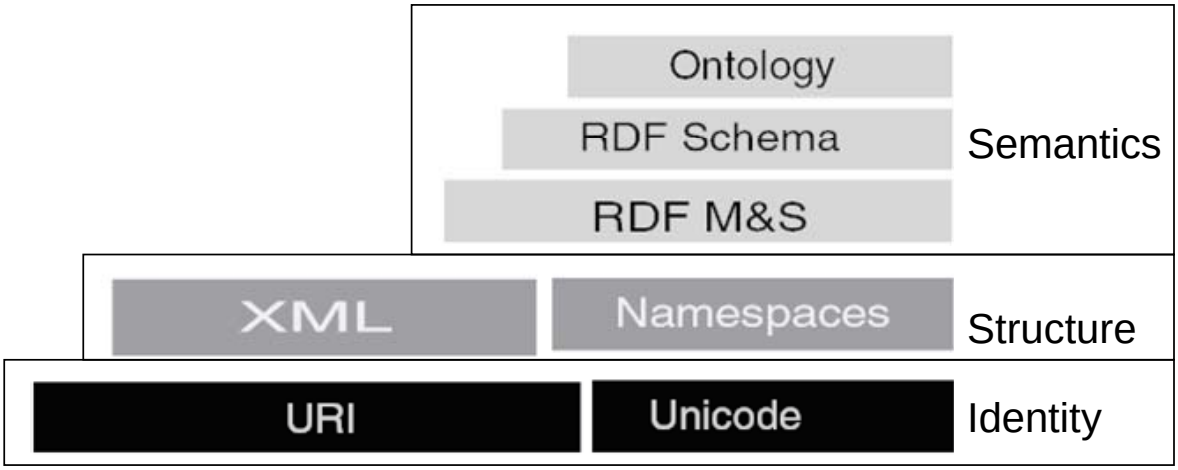


Figure 2.1 The Semantic Web stack [Be4].

There are three principal aspects to be considered in the Semantic Web stack: identity, structure and semantics. Identity refers to identifying resources in the Semantic Web. Structure is related to providing the means to represent resources and to organize them into categories. Semantics has to do with describing resources in terms of their relations with other resources and reasoning over this resource description.

1. Identity

Identity is a very important aspect to consider in order to build the Semantic Web. Given the fact the Semantic Web is an extremely large collection of annotated resources, it is necessary to provide mechanisms to identify each of these resources. Identity is considered in current Web in which Web pages are identified by a URL (Uniform Resource Locator). Efforts regarding identity are URI (Uniform Resource Identifier) and Unicode [Be4]. A URI is a Web identifier and anything that has a URI is considered to be on the Web [Pa01]. Unicode is a character-encoding standard that supports international characters.

2. Structure

Structure is important to be considered in order to describe Semantic Web resources and to organize sets of resources in categories. Hence mechanisms for description

of resources and organization of resources are needed. Current Web mechanism regarding structure, such as HTML (HyperText Markup Language), are not enough to capture all that is going to be expressible in the Semantic Web. XML (eXtendable Markup Language) is a superset of HTML that can be used as syntax for the description of Semantic Web resources.

Namespaces where added to XML to increase its modularization and the reuse of XML vocabularies [Ga03]. Namespaces are used in the Semantic Web for organizing huge amounts of Semantic Web resources in categories. For example there can be namespaces for resources related to education, business, biology, etc.

3. Semantics

Semantics is a crucial aspect for the Semantic Web. Semantics refers to the description of resources in terms of their semantic relations to other resources. Technological efforts regarding semantics include the development of RDF (Resource Description Framework) and the use of ontologies [Be4].

RDF

The Resource Description Framework is a framework for describing and interchanging metadata (i.e. data of data). According to [Brt1] it is built on the following rules:

- A Resource is anything that can have a URI (e.g. the document identified with the URI http://www.name.domain/document1.txt).
- A PropertyType is a Resource that is used as a property (e.g. hasAuthor or hasTitle).
- A Property is the combination of a Resource, a PropertyType and a value or another Resource (e.g. http://www.name.domain/document1.txt hasAuthor “Héctor Pérez” or http://www.name.domain/document1.txt isDiscussedIn http://www.name.domain/discussion1.txt).

A set of resources and their semantic relations (i.e. properties) can be represented using a graph where nodes represent resources and semantic relations are represented with edges. The structure of nodes and edges conform directed graphs that model the network of terms (i.e. resources) and relations between terms of the Semantic Web [Ga03]. Particular edges are identified by the triad composed by the origin node, the property and the destination node. A relation represented with an edge can be regarded as the substantive, the verb and the object of a simple sentence [BHL01]. Triads are called triples ore RDF statements and they are the RDF abstract syntax. Graphs can be serialized as a set of triples, one for each edge in the graph.

Triples can also be assigned an explicit identifier (i.e. an URI). A new node is created that represents the triple and it is associated to three nodes for the three triple components. Abstract triples are the common model to which diverse data structures can be mapped. XML syntax facilitates integrating Semantic Web documents in the current HTML/XML Web. The other possibility is N-Triples, the nearest to the abstract form, a series of triples with the substantive, the verb and the object identified by their URI. The latter uses many syntactic tricks to improve human readability and make serializations more compact for it is the more human aware syntax.

The following XML example was taken from [Pa01] and it represents a resource, a specific article. It basically says that the described article has the title “The Semantic Web: An Introduction” and was written by someone whose name is “Sean B. Palmer”.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:foaf="http://xmlns.com/0.1/foaf/" >
   <rdf:Description rdf:about="">
```

```
      <dc:creator rdf:parseType="Resource">
         <foaf:name>Sean B.  Palmer</foaf:name>
      </dc:creator>
      <dc:title>The Semantic Web: An Introduction</dc:title>
   </rdf:Description>
</rdf:RDF>
```

RDF Schema is an extension to RDF [Be4]. Simple RDF provides the tools to construct semantic networks of resources and relations; nonetheless, there is still a lack of many facilities for they are not available with RDF. RDF needs a way to define specific application classes and properties. Specific application classes and properties must be defined using extensions to RDF like RDF Schema. [Ga03] states that the more relevant schema primitives of the RDFS specification are the following:

- Type. It is a property that relates a resource to a Class to which it pertains. The resource is categorized as a member of this Class and thus it possesses its characteristics.
- Class. It is a set of things that share some characteristics; they have a common conceptual abstraction.
- subClassOf. This property holds the taxonomical relations between classes. If class B is a subclass of class A, then class B has all the typical characteristics of class A plus some specific ones that can distinguish it from A.
- subPropertyOf. This property creates the taxonomy of properties. If property B is a subproperty of property A, then whenever it is stated that the property B holds between two resources it can be deduced that A also holds.

- Domain and range. Both are properties that associate other properties to classes. They constraint the classes to which the associated properties can be connected.

Ontologies

Ontologies are formal, explicit specifications of a shared conceptualization [Gr93]. Conceptualization refers to an abstract model of phenomena in the world by having identified the relevant concepts of those phenomena. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that ontologies should be machine readable. Shared reflects that ontology should capture consensual knowledge accepted by the communities.

Ontologies are a key enabling technology for the Semantic Web for they interweave human understanding of symbols with their property to be processed by machines [DFH03]. Ontologies are used to represent knowledge and the reason they are becoming popular is due to what they promise: a shared and common understanding of a domain that can be shared between people and applications.

2.2 Description logics

Description logics (DL) is a family of knowledge representation (KR) formalisms that represent the knowledge of an application domain (the world) by first defining the relevant concepts of the domain (its terminology), and then using these concepts to specify properties of objects and individuals occurring in the domain (the world description) [BN02]. One of the characteristics of these languages is that they are based on formal, logic-based semantics. Another distinguished feature is the emphasis on reasoning: reasoning allows one to infer implicitly represented knowledge from the knowledge that is explicitly contained in the knowledge base.

DL supports inference patterns that occur in many applications of intelligent information processing systems, and which are also used by humans to structure and understand the world: classification of concepts and individuals [BN02]. Classification of concepts determines if a concept is a specialization of another concept. Classification of individuals determines whether a given individual is always an instance of a certain concept.

According to [BN02] DL is based on the following three ideas:

1. The basic syntactic building blocks are atomic concepts (unary predicates), atomic roles (binary predicates), and individuals (constants).
2. The expressive power of the language is restricted in that it uses a rather small set of constructors for building complex concepts and roles.
3. Implicit knowledge about concepts and individuals can be inferred automatically with the help of inference procedures.

2.2.1 DL system

A DL system also called inference engine provides facilities to reason about the knowledge base (KB) content and to manipulate it (see figure 2.2). Frequently the inference engine is part of a larger environment. Other components interact with the inference engine by querying the knowledge base and by modifying it, that is, by adding and deleting concepts, roles, and assertions.

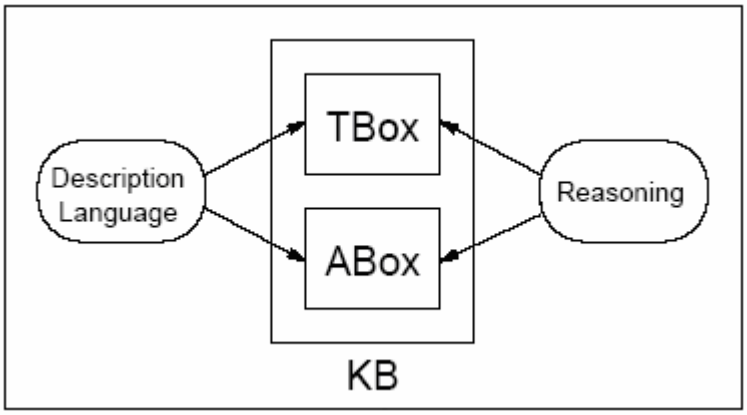


Figure 2.2 Architecture of a knowledge representation system [BN02].

A knowledge base (KB) comprises two components: the TBox and the ABox [BN02]. The TBox introduces the terminology or the vocabulary of an application domain; the vocabulary consists of concepts that represent sets of individuals and roles which represent relations between individuals. The ABox contains assertions about named individuals in terms of this vocabulary.

A DL system not only stores terminologies and assertions, but also offers services that reason about them. Typical reasoning tasks for a terminology are to determine whether a description is satisfiable (i.e. no contradictory), or whether one description is more general than another one. Important problems for an ABox are to find out whether its set of assertions is consistent, that is, whether it has a model, and whether the assertions in the ABox state that a particular individual is an instance of a given concept description.

There are technological efforts regarding the development of DL systems or inference engines such as Racer. Racer (Renamed ABox and Concept Expression Reasoner) can be considered as a core inference engine for the Semantic Web. Racer is used for managing and querying ontologies. It can handle TBoxes with generalized concept inclusions, ABoxes (based on the unique name assumption) and concrete domains [HM1]. Racer is freely available for research purposes and can be accessed by standard HTTP or TCP protocols. It can read knowledge bases either from local files or from remote Web servers. In turn, other client programs that need inference services can communicate with a Racer server via TCP-based protocols [HM2].

2.2.2 DL languages

DL languages are used to create ontologies. There are technological efforts regarding the development of DL languages. We present RDFS, OIL, DAML+OIL and OWL because we consider them good examples given their characteristics.

RDFS

RDFS can be regarded as an ontology language as it can be used to create ontologies. However [DFH03] states that many types of knowledge cannot be expressed in this simple language and presents few examples of useful things that cannot be said in RDFS:

- Stating that every book has exactly one price, but at least one author (and possibly more).
- Stating that titles of books are strings and prices of books are numbers.
- Stating that no book can be both hardcover and softcover.
- Stating that every book is either hardcover or softcover (i.e. there is no other option than these two).

Richer languages than RDFS are required to represent more than trivial ontologies on the Semantic Web.

OIL

OIL (Ontology Inference Layer or Ontology Interchange Language) is a standard for specifying and exchanging ontologies. It offers a Web based representation and inference layer for ontologies [DFH03]. OIL incorporates the essential modeling primitives of frame-based systems: it is based on the notion of a concept and the definition of its superclasses and attributes. Relations can also be defined not as an attribute of a class but as an independent entity having a certain domain and range. Like classes, relations can fall into a hierarchy. OIL has a well-defined syntax in XML and it is an extension to RDF and RDFS.

According to [DFH03] the main design goals for OIL are (1) maximizing compatibility with existing W3C standards, such as XML and RDF, (2) maximizing partial interpretability by less semantically aware processors, (3) providing modeling primitives that have proven useful for large user communities, (4) maximizing expressiveness to enable modeling of a wide variety of ontologies, (5) providing a formal semantics (a mathematically precise description of the meaning of every expression) in order to facilitate

machine interpretation of that semantics and (6) enabling sound, complete and efficient reasoning services, if necessary by limiting the expressiveness of the language. These design goals lead to the following three requirements [DFH03]:

- It must be highly intuitive to the human user.
- It must have a well-defined formal semantics with established reasoning properties to ensure completeness, correctness, and efficiency.
- It must have a proper link with existing Web languages such as XML and RDF to ensure interoperability.

DAML+OIL

DAML+OIL (DARPA Agent Markup Language + Ontology Inference Layer) is the successor of OIL, defined in collaboration with research groups from the DARPA (Defense Advanced Research Projects Agency) sponsored DAML program [DFH03]. From the point of view of language constructs, the differences between OIL and DAML+OIL are relatively trivial. Although there is some difference in keyword vocabulary, there is usually a one to one mapping of constructors and in the cases where the constructors are not completely equivalent, simple translations are possible [DFH03]. It includes the following characteristics:

- Integration with RDFS

  DAML+OIL is similar to OIL in many respects, but is more tightly integrated with RDFS. While the dependence on RDFS has some advantages in terms of the re-use of existing RDFS infrastructure and the portability of DAML+OIL ontologies, using RDFS to completely define the structure of DAML+OIL is quite difficult as, unlike XML, RDFS is not designed for the precise specification of syntactic structure.

- Treatment of individuals

  The treatment of individuals in DAML+OIL is very different from that in OIL. In the first place, DAML+OIL relies on RDF for assertions on the type (class) of an

individual or a relation between a pair of individuals. In the second place, DAML+OIL treats individuals occurring in the ontology as true individuals (i.e. interpreted as single elements in the domain of discourse) and not as primitive concepts as is the case in OIL. Moreover, there is no unique name assumption: in DAML+OIL it is possible to explicitly assert that two individuals are the same or different, or to leave their relation unspecified.

- Data Types

  The last DAML+OIL version was extended with arbitrary data types from the XML Schema type system which can be used in restrictions and range constraints.

OWL

OWL (Ontology Web Language) facilitates greater interpretability of Web content than that supported by XML, RDF and RDFS by providing additional vocabulary along with formal semantics. OWL is a revision of the DAML+OIL Web ontology language incorporating lessons learned from the design and application of DAML+OIL. OWL provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users:

- OWL Lite

  It supports those users primarily needing a classification hierarchy and simple constraints. While it supports cardinality constraints, it only permits cardinality values of 0 or 1. OWL lite provides a quick migration path for thesauri and other taxonomies. It also has a lower formal complexity than OWL DL [W3C04].

- OWL DL

  It supports those users who want the maximum expressiveness while retaining computational completeness (i.e. all conclusions are guaranteed to be computable) and decidability (i.e. all computations will finish in finite time). OWL DL is so

named due to its correspondence with DL, a field of research that is the formal foundation of OWL [W3C04].

- OWL Full

  It is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. OWL Full allows an ontology to augment the meaning of the pre-defined vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full [W3C04].

2.3 Ontology edition and visualization applications

Other technological efforts include the development of ontology editors and visualization applications such as RICE and OilED. RICE (Racer Interactive Client Environment) is a Racer tool for visualizing Aboxes and enables users to interactively define queries using these visualizations [MCH03]. RICE is started as an application program and can be configured to connect to a Racer server by giving a host name and a port. When RICE connects to a Racer server it retrieves all TBoxes and displays them in an unfoldable tree view and shows a graphical representation of relations in an Abox. RICE can add individual DL statements to Racer and can be used to pose queries on Racer. RICE can also deal with multiple TBoxes and associated ABoxes. In particular, it can show instances of a concept and concepts of instances [HM2].

OilEd (Ontology Inference Layer Editor) is an ontology editor that allows building ontologies using DAML+OIL. The initial intention behind OilEd was to provide a simple editor that demonstrated the use of the OIL language. The current version of OilEd does not provide a full ontology development environment. It will not actively support the development of large-scale ontologies, the migration and integration of ontologies, versioning, argumentation and many other activities involved in ontology construction.

Rather, it is the “NotePad” of ontology editors, offering enough functionality to allow users to build ontologies [BG].

2.4 Conclusions

The essential aim of the Semantic Web vision and the surrounding technological efforts is to make Web information practically processable by a computer. The Semantic Web will eventually become an environment where data can be shared and processed by automated tools as well as by people. Underlying this is the goal of making the Web more effective for its users. This increase in effectiveness is constituted by the automation or enabling of things that are currently difficult to do: locating content, relating content or drawing conclusions from information found in two or more separate sources. The rest of the document shows the considerations that have to be made in order to build a mediation system that allow applications to have transparent access to Semantic Web resources.

3 SKIMA

SKIMA (Semantic Knowledge and Information Management) is a mediation system that gives transparent access to heterogeneous sources (e.g. Semantic Web resources) by taking advantage of their semantics and application requirements. It is based on a pivot model that abstracts concepts and relations similar to ontology based approaches. Sources content is described as a set of concepts and their semantic relations. Similarly, application's requirements are described in terms of concepts and semantic relations concerning a specific domain (see figure 3.1).

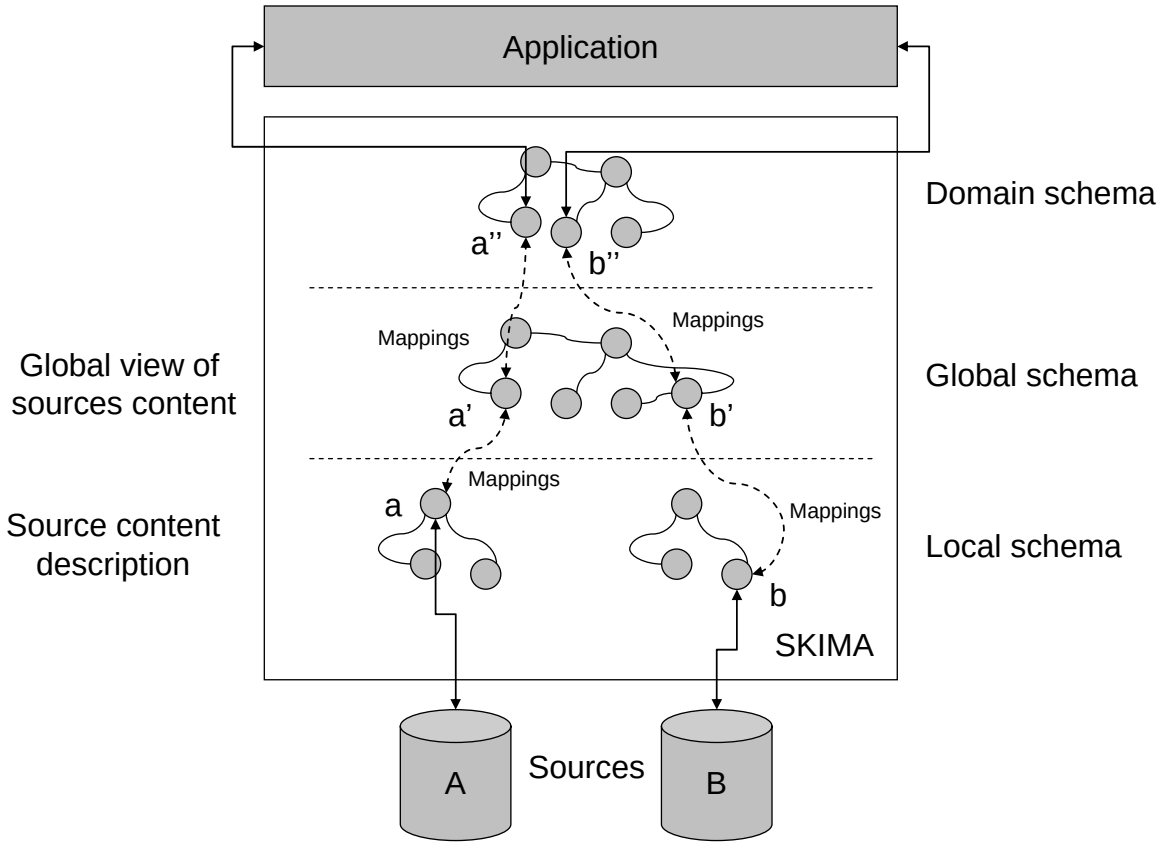


Figure 3.1 SKIMA general approach.

SKIMA provides an integrated and global view of sources content and couples it to the description of application requirements. There are relations between schemas that enable concepts to be translated from a schema to another; these relations are called mappings. Domain schema concepts are linked to global schema concepts. Similarly, global schema

concepts are linked to local schema concepts. SKIMA provides a global schema and translates domain expressions to local through the global schema using mappings.

The remainder of this chapter is organized as follows. Section 3.1 presents the general architecture of SKIMA and its functions. Section 3.2 presents SKIMA data structures. Section 3.3 describes the query, its representation and processing. Section 3.4 presents how to use SKIMA. Finally section 3.5 presents our conclusions.

3.1 General architecture

Figure 3.2 shows the general architecture of SKIMA that enables applications to query sources by using specific domain terms. SKIMA processes queries in order to have a representation that helps to determine where to retrieve information.

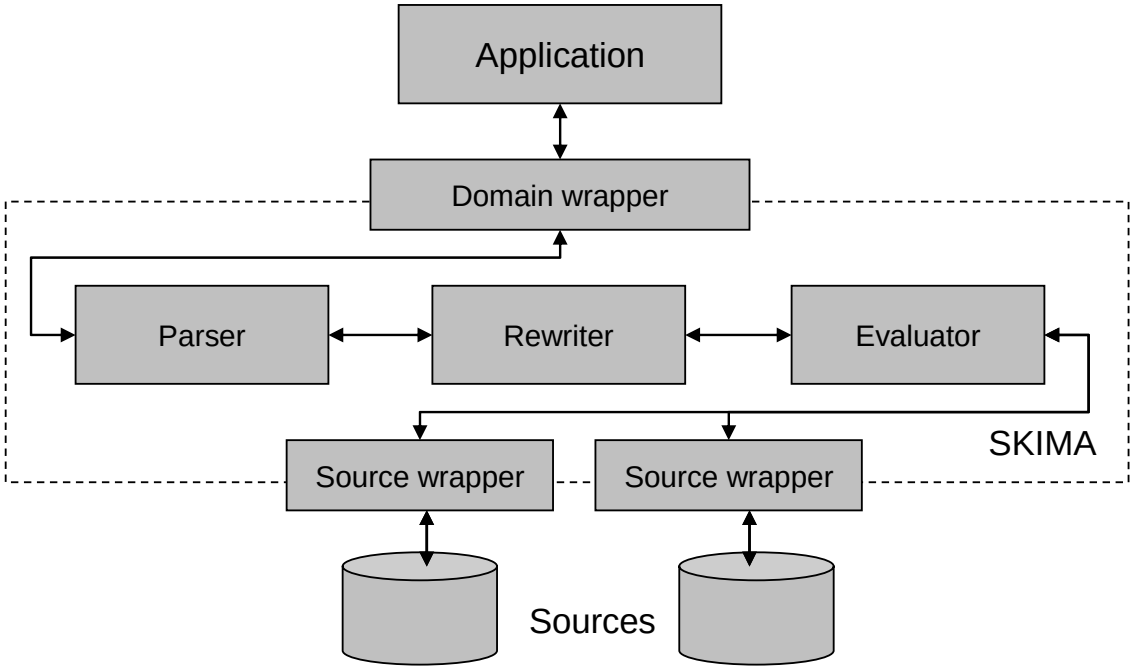


Figure 3.2 SKIMA general architecture.

SKIMA is composed by two types of wrappers and three internal modules. The domain wrapper represents the application requirements while the source wrapper is used to

represent source content descriptions. The parser verifies queries (i.e. checks if it is well expressed). The rewriter rebuilds queries expressed in terms of the domain model to the global schema and then to the local schema. Finally the evaluator is an inference engine that evaluates queries and translates results expressed in terms of the local schema to the domain schema through the global schema. The parser, the rewriter and the evaluator are used to manage queries made by applications over the domain schema.

3.1.1 Data model

As already said, SKIMA is based on a pivot model that abstracts concepts and relations as directed graphs similar to ontology based approaches (see figure 3.3).

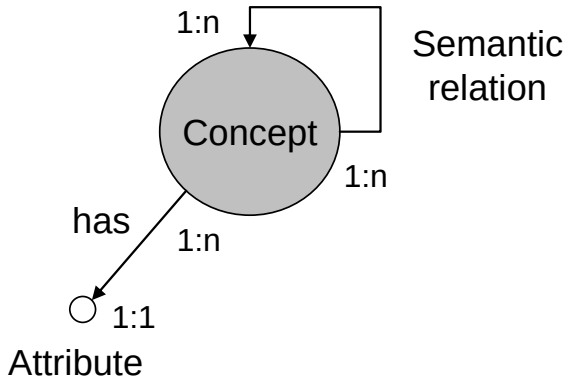


Figure 3.3 Data model.

A concept can be related to many concepts through semantic relations. A concept can be related to many attributes through a specific relation called ‘has’. An attribute is related to only one concept. A concept is identified with a URI (Uniform Resource Identifier). A semantic relation is also identified with a URI and it is composed by the origin node, the relation name and the destination node. A collection of concept and relation definitions (i.e. axioms) conform a schema.

3.1.2 Functions

SKIMA functions are related to query handling. A query made by an application over the domain schema has to be translated to a new query in terms of the global schema and the

local schema before being evaluated. Similarly, query results are translated from the local schema to the global schema and then from the global schema to the domain schema (see figure 3.4). Translations between schemas are possible thanks to mappings.

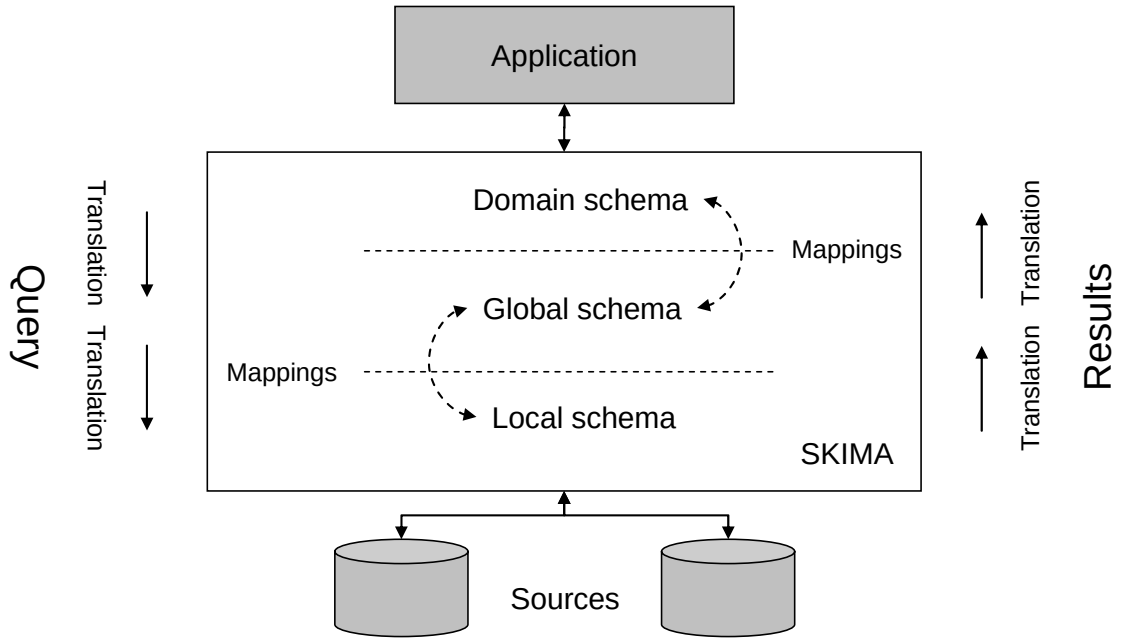


Figure 3.4 Query handling.

3.2 Data structures

SKIMA data structures are all based on the pivot model presented in section 3.1.1. Data structures include the local schema, the domain schema, the global schema and mappings.

3.2.1 Local schema

The local schema represents the content and the characteristics of a given source. Content is represented with a graph expressed according to the pivot data model. Source characteristics include its availability, its cost and its quality. Figure 3.5 shows the abstract representation of a source.

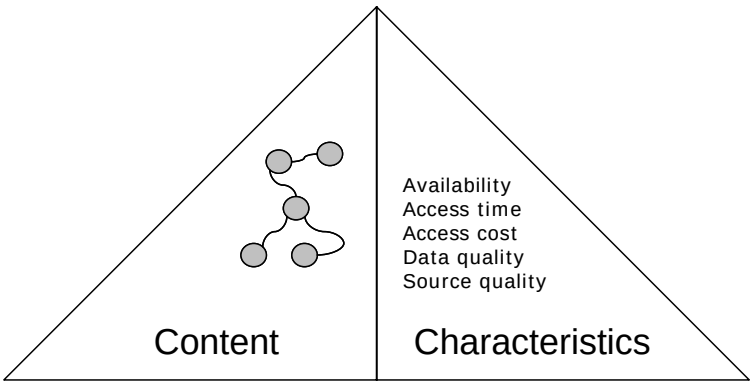


Figure 3.5 Abstract representation of a source.

3.2.2 Domain schema

The domain schema represents a set of concepts and their relations. The nature of these concepts depends on a certain application domain. If we were to build an application to manage family information we could use the domain shown in figure 3.6.

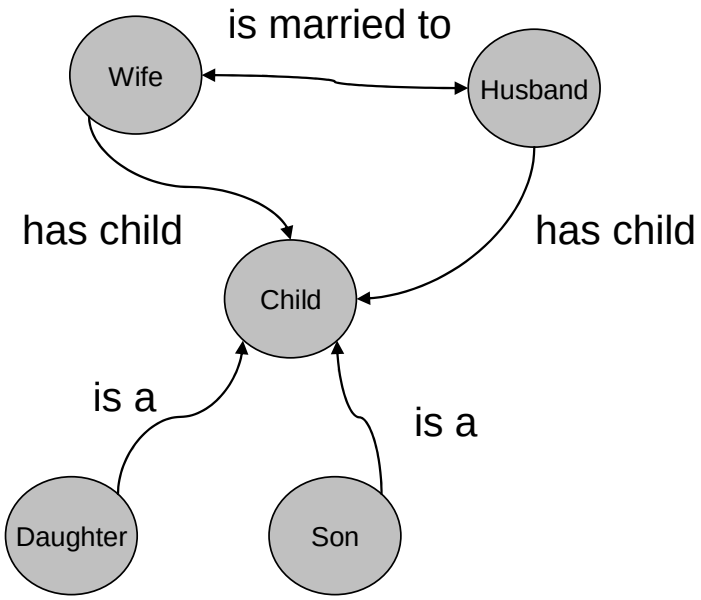


Figure 3.6 Domain example.

As it can be seen in figure 3.6, there are five concepts representing people. Some familiar relationships are shown. Wives are married to their husbands and husbands are married to their wives. Husbands and wives have children that can be sons or daughters.

3.2.3 Global schema

The global schema is a global view of sources content represented according our pivot model. Global schema concepts are related to local schema concepts and to domain schema concepts as shown in figure 3.7.

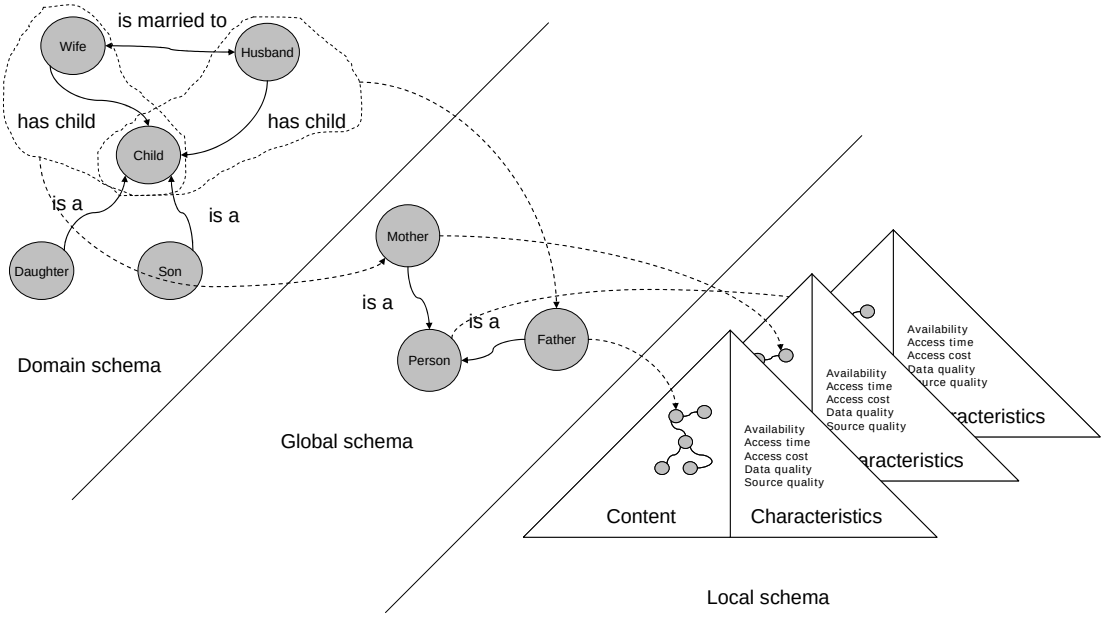


Figure 3.7 Translation between schemas.

3.2.4 Mappings

Relations between concepts from different schemas are called mappings. They are used to establish whether a concept is equivalent to, a subset of or a superset of another concept. There are three types of mappings: exact, sound and complete.

Exact mappings

An exact mapping can be established between two concepts from different schemas when they are semantically equivalent (i.e. they have the same meaning). This kind of mapping is used to manage synonymy. The exact mapping shown in figure 3.8 establishes that wives that have children are married mothers.

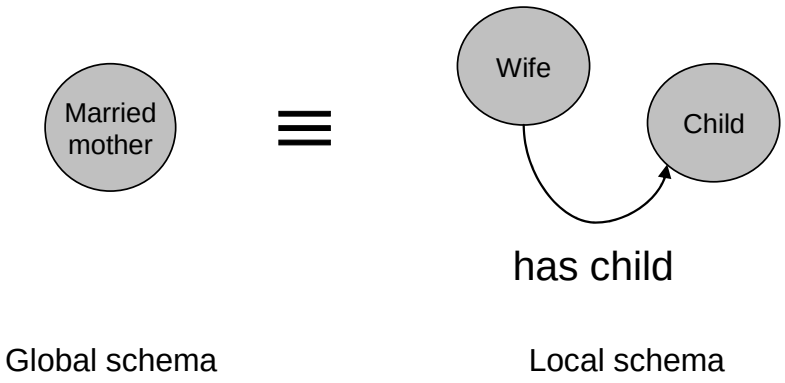


Figure 3.8 Exact mapping.

Sound mappings

A sound mapping can be established between two concepts when a global schema concept is a subset of a local schema concept. The sound mapping shown in figure 3.9 establishes that wives who have children are a subset of wives.

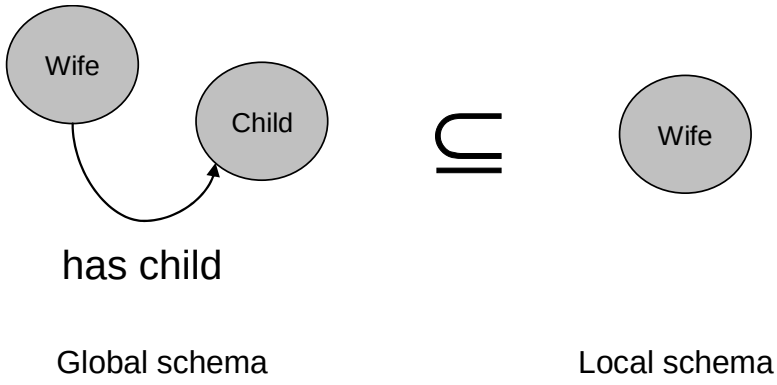


Figure 3.9 Sound mapping.

Complete mappings

A complete mapping can be established between two concepts when a global schema concept is a superset of a local schema concept. The complete mapping shown in figure 3.10 establishes that wives are a superset of wives who have children.

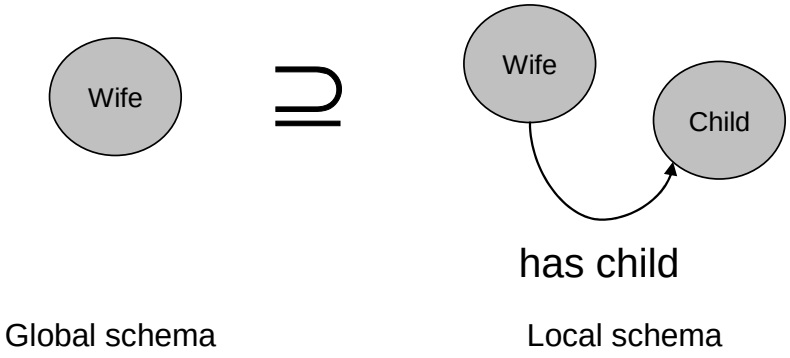


Figure 3.10 Complete mapping.

3.3 Querying SKIMA

The local schema represents the content and the characteristics of a given source. Content is represented with a graph expressed according to the pivot data model. Source characteristics include its availability, its cost and its quality. Figure 3.5 shows the abstract representation of a source.

Queries are expressed in terms of the domain schema. The abstract representation of a query includes a concept tree and a profile (see figure 3.11). A concept tree is the representation of a domain schema concept. The profile describes sources that can be considered in order to retrieve query results.

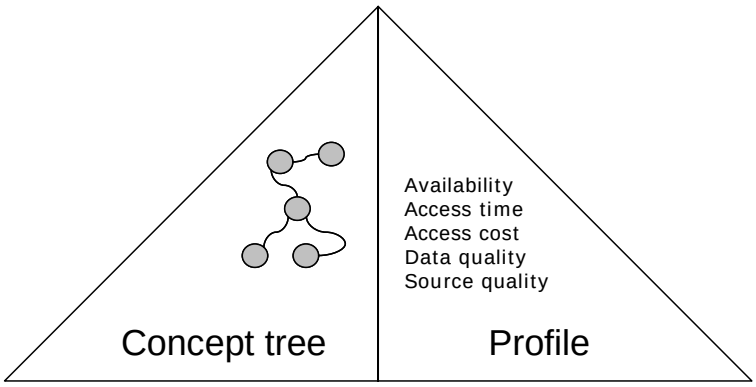


Figure 3.11 Abstract representation of a query.

In order to explain the notion of concept tree consider the domain shown in figure 3.6. One could be interested in getting the list of husbands who have children; a concept tree can be built from this specific query as shown in figure 3.12. A concept tree can be as complex as necessary in order to represent complex queries.

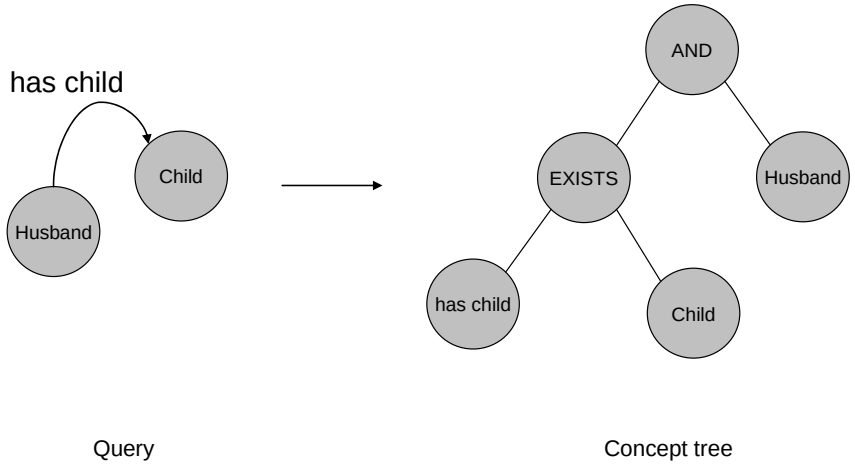


Figure 3.12 Concept tree example.

3.3.1 Query processing

As it is shown in figure 3.13 queries made by applications over the domain schema pass by three phases:

1. Parsing. The query concept tree is transformed into a query expression.
2. Rewriting. The query expression is translated to a set of global schema concepts, then each of these global schema concepts are translated to a set of local schema concepts.
3. Evaluation. The local schema concepts are populated with a set of individuals.

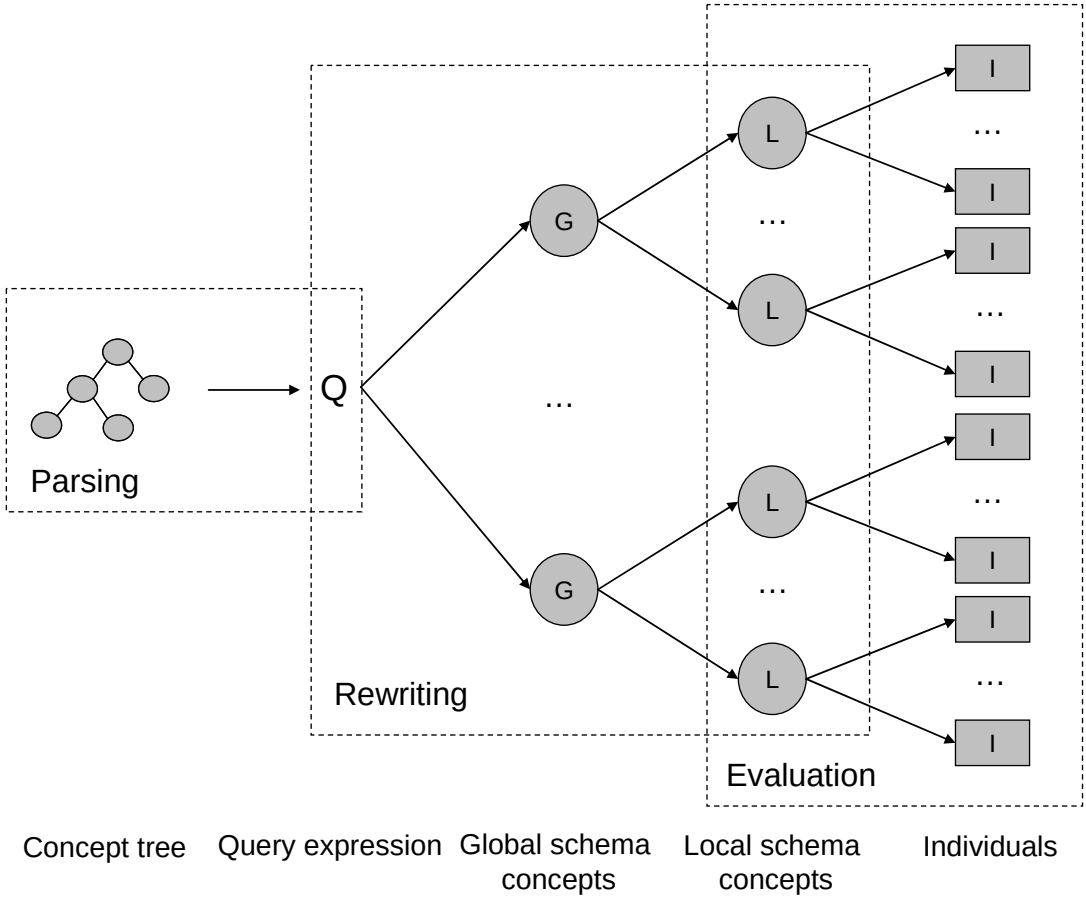


Figure 3.13 Query processing.

Query processing is based in the following algorithm:

Query processing (*cp :* Concept tree) : Populated concept list array

1. *qe* : Query expression = Parsing (*cp*)
2. If *qe* is not null then
3. *cla* : Concept list array = Rewriting (*qe*)
4. *pcla* : Populated concept list array = Evaluating (*cla*)
5. Return *pcla*
6. Else
7. Return null (*qe* is not valid)

Query processing manages a set of data types (see figure 3.14). Primitive types are concept, individual, concept tree and query expression. Complex data types are the following:

- Concept list. Array of concepts.
- Concept list array. Array of concept lists.
- Populated concept. Array of individuals.
- Populated concept list. Array of populated concepts.
- Populated concept list array. Array of populated concept lists.

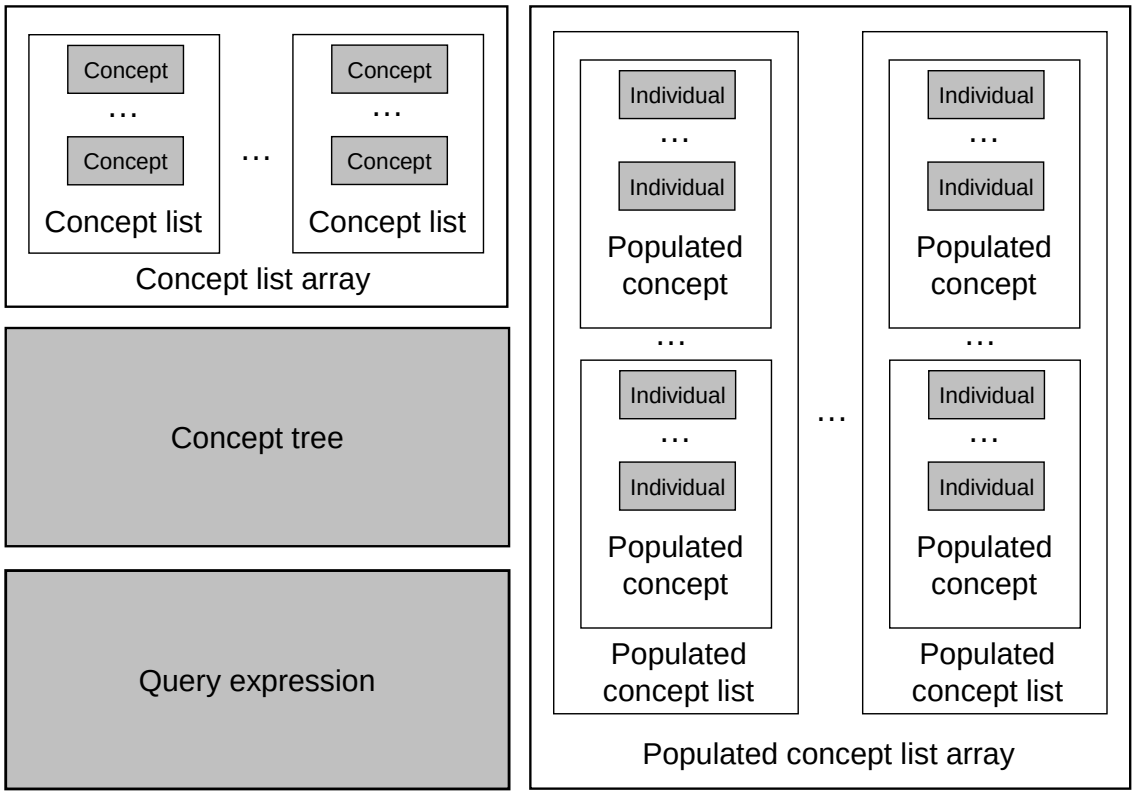


Figure 3.14 Data types.

Parsing

Parsing a query involves verifying if it is well constructed. The parser transforms a concept tree into a query expression and then checks for its validity. As already explained a query is represented with a concept tree. Any concept tree can be translated into a query expression (see figure 3.15).

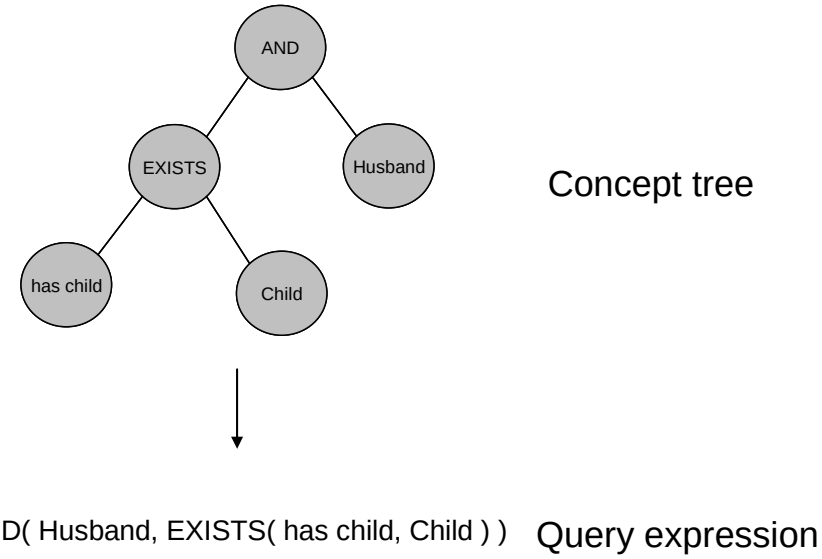


Figure 3.15 Query expression example.

Once the query expression is gotten from the concept tree, it is verified. In this phase the query is analyzed semantically. The parsing process has two possible outputs: a valid expression and a null expression. Parsing is based on the following algorithm:

Parsing (*cp* : Concept tree) : Query expression

1. Translate *cp* to a query expression (*qe* : Query expression)
2. If *qe* respects global schema constraints (i.e. axioms on concepts and relations) then
3. If *qe* can be classified under one or more global schema concepts then
4. Return *qe* (*qe* is valid)
5. Else
6. Return null (*qe* is invalid)
7. Else

8. Return null (*qe* is invalid)

Rewriting

Rewriting refers to obtaining a set of local schema concepts from a given query expression. Rewriting is based on the following algorithm:

Rewriting (*qe* : Query expression) : Concept list array

1. Get equivalent global schema concepts of *qe* using mappings between the domain schema and the global schema (*gcl* : Concept list)
2. For each concept of *gcl* do
3. Get equivalent local schema concepts of the given global schema concept using mappings between the global schema and the local schema (*lcl* : Concept list)
4. Add *lcl* to an array of concepts list (*cla* : Concept list array)
5. Return *cla*

Evaluation

Evaluation refers to populating local schema concepts with instances. The evaluator is an inference engine that provides a set of instances of a given concept or set of concepts. Evaluation is based on the following algorithm:

Evaluation (*cla* : Concept list array) :

1. For each concept list of *cla* do
2. For each concept of the given concept list do
3. Get individuals of the given concept by populating it using the inference engine (*pc* : Populated concept)
4. Add *pc* to a populated concept list (*pcl* : Populated concept list)
5. Add *pcl* to a populated concept list array (*pcla* : Populated concept list array)
6. Return *pcla*

3.4 Using SKIMA

In order for application to use SKIMA it is necessary to configure it and start it.

3.4.1 Configuring the system

In order to use SKIMA it must be configured. Configuring SKIMA means to provide the domain schema, the global schema, the local schema and the mappings between them (see figure 3.16).

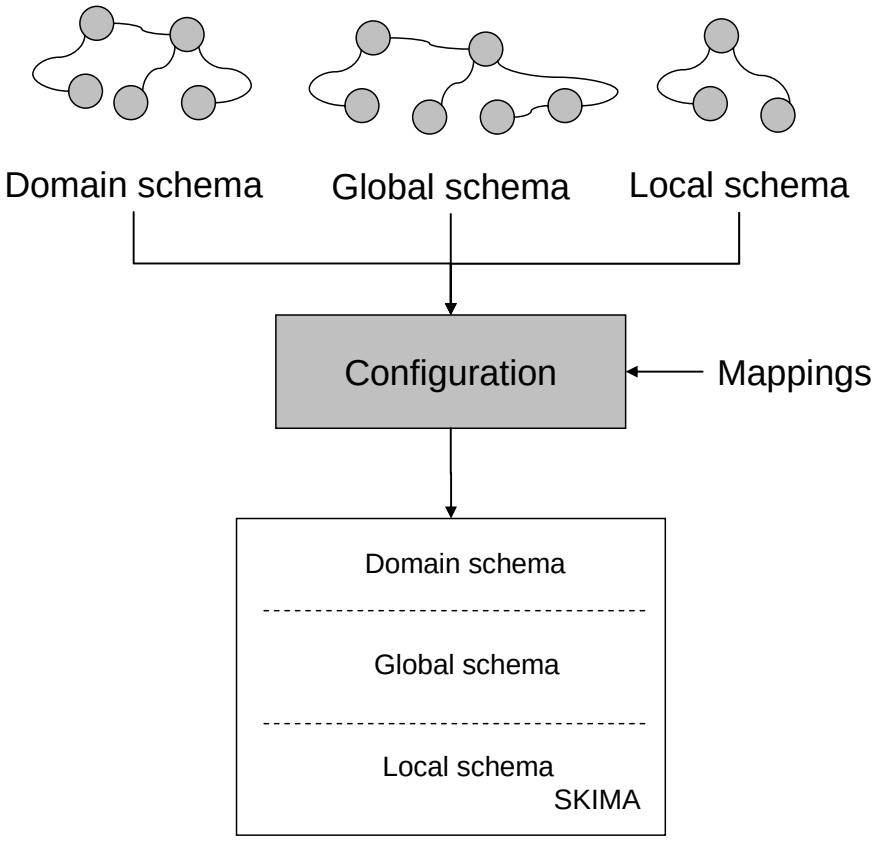


Figure 3.16 Configuring SKIMA.

3.4.2 Starting the system

Once schemas and mappings are provided, SKIMA can be started for applications to use it. An application imports interfaces provided by SKIMA into its code and uses SKIMA to query the domain schema based on a client/server architecture, where the application is the

client and SKIMA is the server (see figure 3.20). There can be of course many clients or applications using the same server, nevertheless all of them have to share schemas.

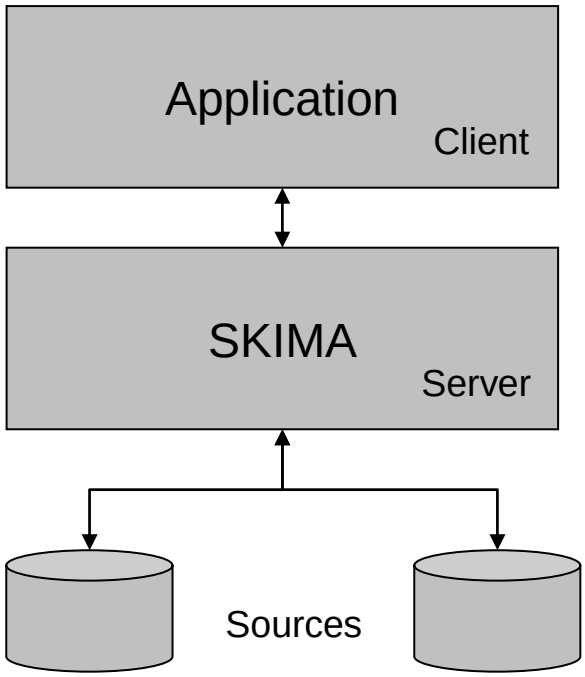


Figure 3.17 An application using SKIMA based on a client/server architecture.

3.5 Conclusions

SKIMA is a data mediation system that gives transparent access to heterogeneous sources (e.g. Semantic Web resources) by taking advantage of their semantics and application requirements. It implements three schemas: the domain schema, the global schema and the local schema. These schemas are related with mappings that allow translating concepts from one schema to another.

SKIMA processes queries using three components: the parser, the rewriter and the evaluator. The parser is a mechanism that verifies if queries are valid. The rewriter is a

module that translates concepts from the domain schema to the local schema using mappings. The evaluator is an inference engine that populates local schema concepts.

In order to use SKIMA it is necessary to configure it by providing the schemas (domain, global and local) and the related mappings. Once schemas and mappings are provided, applications can import SKIMA interfaces in order to use them based on a client/server architecture.

4 MBF

The mediator building framework (MBF) is a template for building mediators. It is a collection of several domain schemas, global schemas and local schemas with their related mappings. It allows users to choose a set of schemas in order to build their own SKIMA. MBF is based on the pivot data model presented in section 3.1.1.

The remainder of this chapter is organized as follows. Section 4.1 presents the general architecture of MBF. Section 4.2 describes the schema description as a set of characteristics used to choose a set of schemas in order to build a mediator. Section 4.3 presents MBF functions. Section 4.4 describes how to use MBF. Finally section 4.5 presents our conclusions.

4.1 General architecture

Figure 4.1 presents the general architecture of MBF that enables users to build ad hoc mediators from a chosen set of schemas and their related mappings.

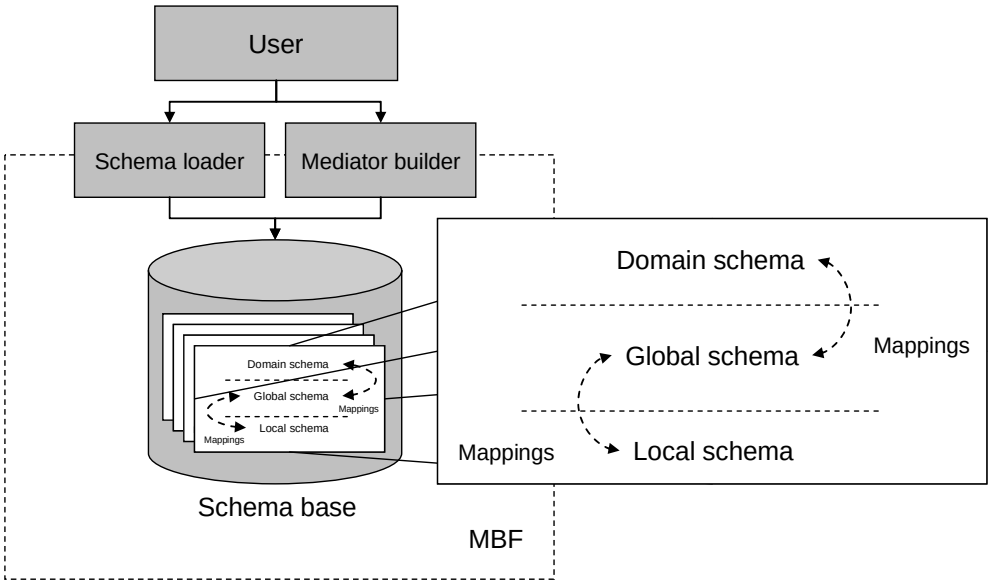


Figure 4.1 MBF general architecture.

The MBF is composed by three components: the schema loader, the mediator builder and the schema base which are used by MBF functions. The schema loader is used to add schemas and mappings to the schema base. The schema base is an inference engine that is used to store schemas and mappings. The mediator builder provides the means to choose schemas and mappings from the schema base given a set of preferences called schema description.

4.2 Schema specification

The schema specification is a set of characteristics that describe a set of schemas. The schema specification is used to choose a domain schema, a global schema and a local schema with their related mappings from the schema base. Figure 4.2 depicts the abstract representation of a schema specification.

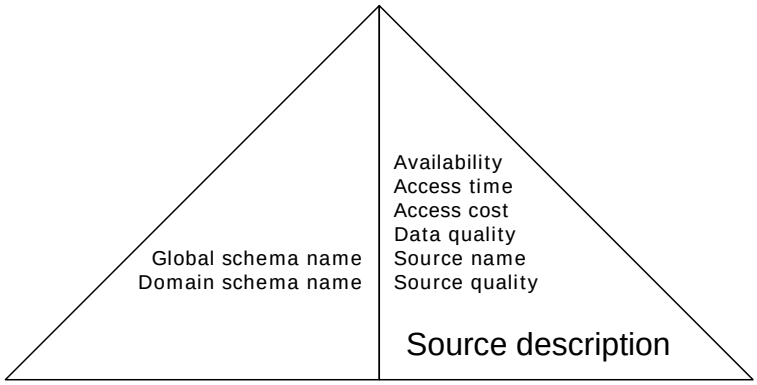


Figure 4.2 Abstract representation of a schema specification.

A schema specification is composed by two parts: the global schema and the domain schema description in terms of their names and the source description as a set of characteristics that describe the sources.

4.3 Functions

As already said MBF has two functions: (1) schema and mapping subscription and (2) mediator building.

4.3.1 Schema and mapping subscription

The schema loader is used to subscribe schemas and mappings to the schema base. Figure 4.3 shows the loading process.

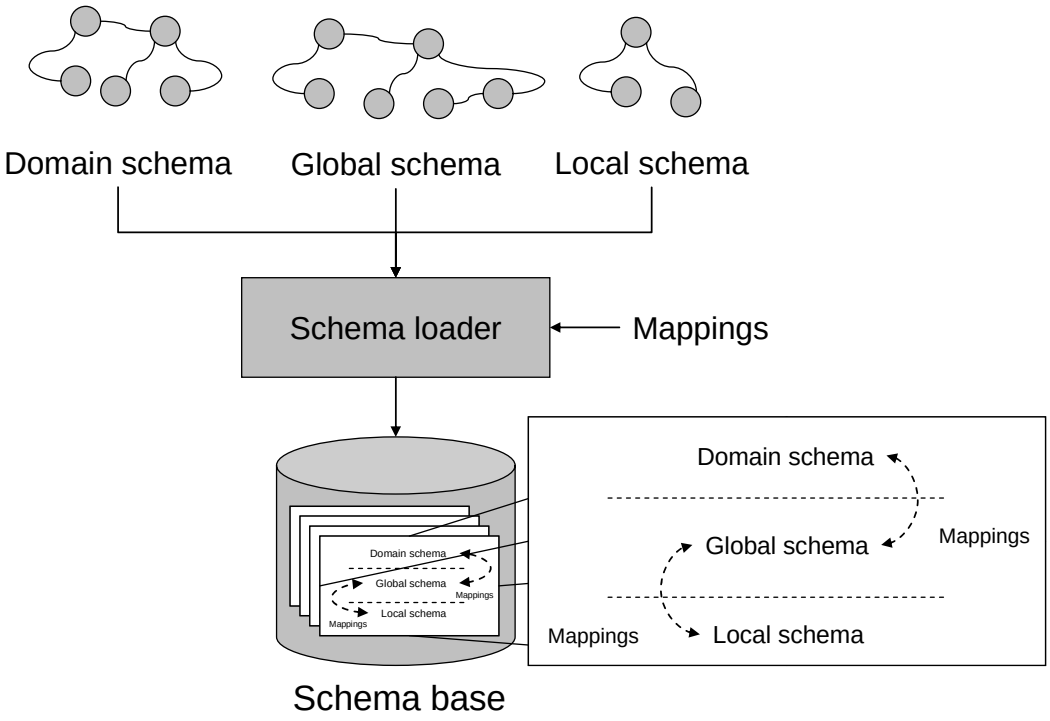


Figure 4.3 Loading schemas and mappings.

The schema loader receives a set of schemas and their related mappings and stores them in the schema base.

4.3.2 Mediator building

The mediator builder is used to build a specific SKIMA from a schema description that is used to choose a domain schema, a global schema and a local schema with their related mappings. Figure 4.4 shows the mediator building process.

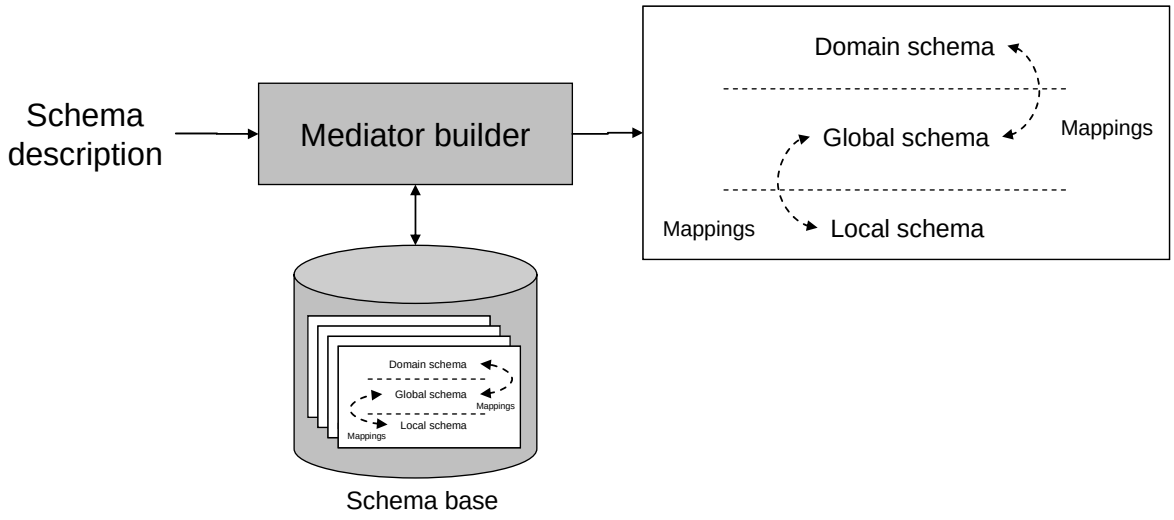


Figure 4.4 Building mediators.

The mediator builder is based on the following algorithm:

Selects schemas (*sd* : Schema description) : Schema set

1. Search the domain schema of *sd* in the schema base (*ds* : Domain schema)
2. If *ds* is not null then
3. Add *ds* to a schema set (*ss* : Schema set)
4. Search the global schema of *sd* in the schema base (*gs* : Global schema)
5. If *gs* is not null then
6. Add *gs* to *ss*
7. Get mappings between *ds* and *gs* (*dgm* : Mappings)
8. Add *dgm* to *ss*
9. Get the local schema based on the source description of *sd* (*ls* : Local schema)

10. If *ls* is not null then
11. Add *ls* to *ss*
12. Get mappings between *ls* and *gs* (*lgm* : Mappings)
13. Add *lgm* to *ss*
14. else
15. return null
16. else
17. return null
18. else
19. return null
20. return *ss*

4.4 Using MBF

Figures 4.5 and 4.6 shows MBF GUI (Graphical User Interface) with its two sections: Load schemas and Mediator building.

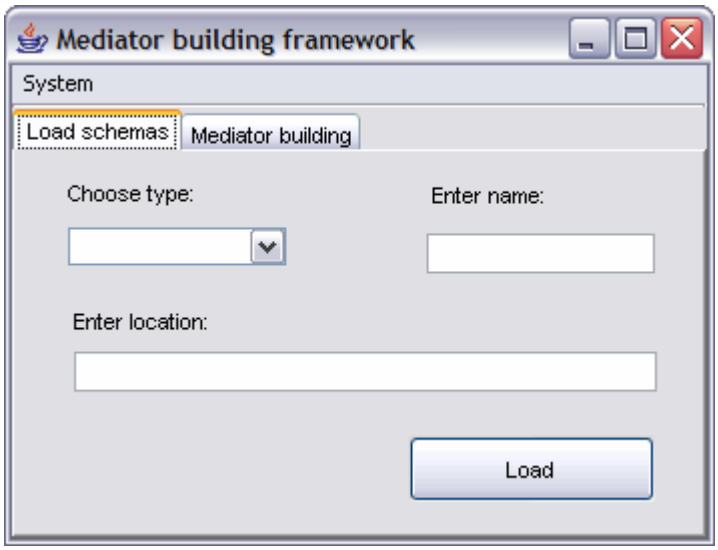


Figure 4.5 Loading schemas.

As it can be seen in figure 4.5 the GUI has one menu. The System menu is composed by only one item (Exit) which is used to stop MBF. Loading a schema means loading it into the schema base. Schemas are sets of concepts and their semantic relations (i.e. ontologies). In order to load a schema one must specify its type (e.g. domain, global, local), its name and its location.

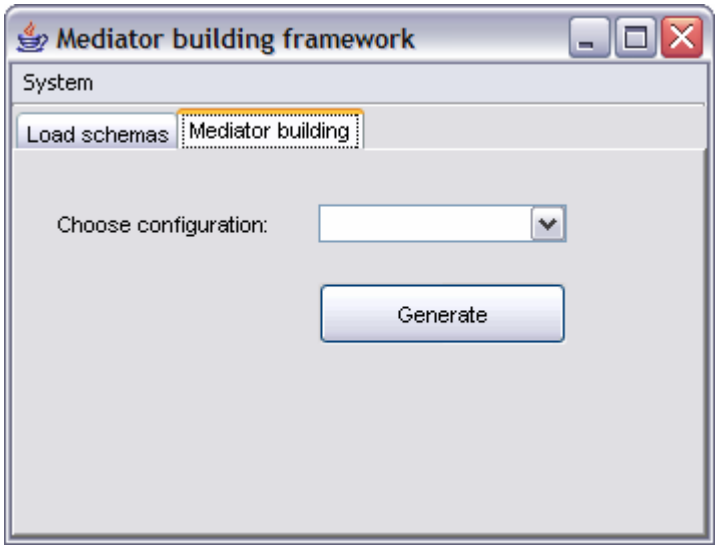


Figure 4.6 Building a mediator.

In order to build a mediator, MBF retrieves the list of schema descriptions (i.e. configurations). Once a specific configuration is selected, the mediator can be built based on the selected schema description. In general terms, building a mediator consists in choosing a set of schemas (i.e. a domain schema, a global schema and a local schema) from the schema base. Once a set of schemas is chosen an application can use SKIMA.

4.5 Conclusions

MBF is a template for building mediators as a collection of several domain, global and local schemas with their related mappings. MBF manages a schema base and it is used to load schemas and to build mediators based on specific schema descriptions. MBF and its functions are based on the pivot model shown in section 3.1.1.

5 Experimentation

This chapter is a description of our prototype a simple tutorial or programmed instruction (PI) system that uses SKIMA to access Semantic Web resources. The use of the Semantic Web facilitates the exploitation of many types of resources that can be easily selected in terms of their utility thanks to their annotated metadata. The idea is to obtain the needed information from the Semantic Web and to take advantage of the available metadata to query it.

The remainder of this chapter is organized as follows. Section 5.1 presents our validation context. Section 5.2 describes the architecture and the functions of the PI system. The domain schema that represents the application domain is presented is section 5.3. The local schema of our system is discussed in section 5.4. Section 5.5 presents how to use the PI system. Finally section 5.6 presents our conclusions.

5.1 CAI

CAI (Computer assisted instruction) in general refers to the use of a computer as a tool within the educational process. The following definitions are a synthesis taken from [Ba85, Gr77]:

- Computer Assisted Instruction (CAI) refers to practice activities, tutorials or simulations offered by computers as supplement of traditional education.
- Computer Based Education (CBE) and Computer Based Instruction (CBI) refer to any use given to computers in an educational environment.
- Computer Managed Instruction (CMI) refers to the use of computers by the school staff for organizing information and making decisions.
- Computer Enriched Instruction (CEI) has to do with the learning activities in which computers (1) generate data for the students, (2) execute programs developed by the students and (3) provide feedback to the students.

5.1.1 Programmed Instruction (PI)

PI is a method of presenting new subject matter to students in a graded sequence of controlled steps [Ay03a]. Students work through the programmed material by themselves at their own speed and after each step test their comprehension by answering an examination question or filling in a diagram. They are then immediately shown the correct answer or given additional information.

Computers and other types of teaching machines are often used to present the material. The use of computers brings positive aspects since computers individualize learning, allow to experiment with different possibilities and learning options, give immediate feedback, are more objective than conventional teachers, give a sense of control over learning and are excellent to practice with. The use of computers in a PI context implies the design of a PI system.

There are five concepts to take into consideration when designing a PI system [Ay03a]:

- Structural analysis. Determination of the content concepts and their relations. Creation of the concept net.
- Didactic structure. Organization of the concept net according to a logical and a psychological order. The logical order is defined based on the content structure. The psychological order is considered to make the content as easy to learn as possible and it is based on the content complexity.
- Presentation units. Definition of units that contain the knowledge students should obtain. These units are created based on the logical order of the concept net. Once the units have been defined they are organized based on the psychological order of the concept net.
- Examination tests. Definition of the questions correspondent to each presentation unit. Pressey (based on choosing the correct option) or Skinner (based on filling in the blanks) models may be followed.
- Feedback. Definition of the system response after examination. Proper multimedia is presented to explain the student success or failure.

Any system requires useful and understandable information to perform properly. In a PI system this information mainly concerns students, teachers and courses. Given the fact all this information could be obtained from the exploitation of Web resources, it could come in a great variety of content, format, location, quality and many other criteria. We have identified three main needs regarding knowledge management in a PI system: (1) the demand of useful and understandable information, (2) the exploitation of Web resources and (3) the use of a great variety of data.

5.2 General architecture

Figure 5.1 presents the PI system general architecture. As already said the PI system uses SKIMA to access Semantic Web resources.

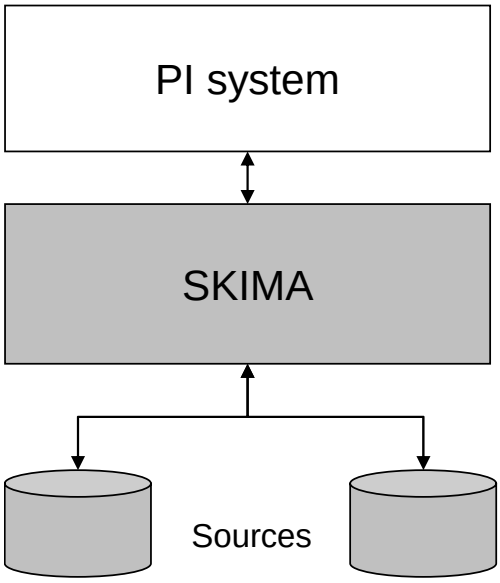


Figure 5.1 PI system general architecture.

The PI system uses SKIMA in order to access sources that are composed by resources based on the Semantic Web representation approach. As already presented a PI system implements a method of presenting new subject matter to students in a graded sequence of

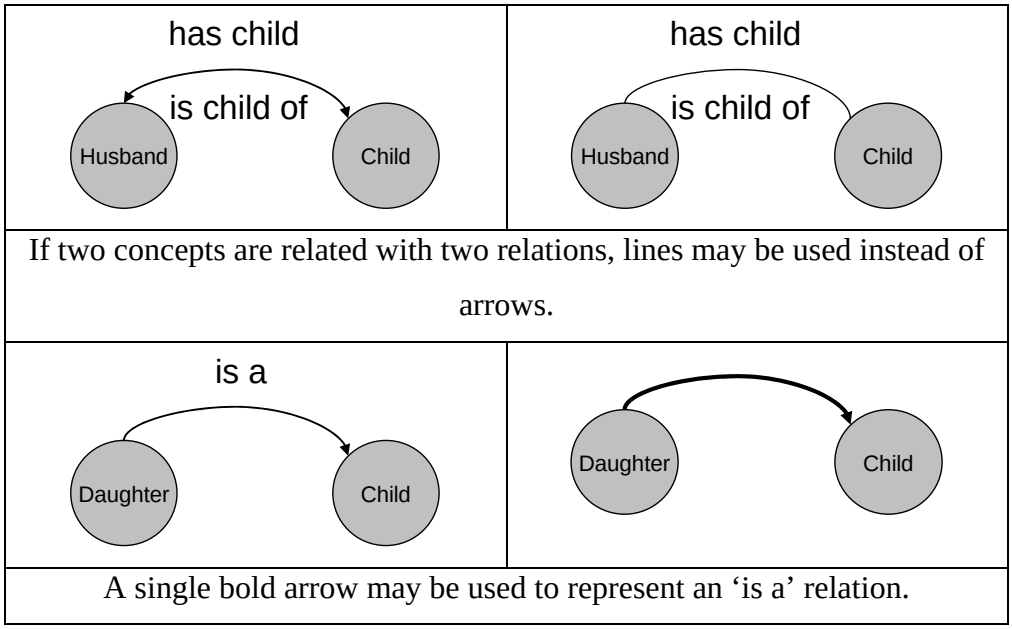
controlled steps. The strategy of a PI system can be reduced to the basic algorithm presented in [Ay03a]:

1. Present initial unit
2. Do
3. The student reads, assimilates and integrates the presented information
4. The system asks the student something related to the unit that has just been presented
5. If the unit is considered to be approved then
6. Next unit will be the following according to the psychological order of the concept net
7. Else
8. Next unit is again the one that has just been presented
9. Present next unit
10. While there are units to present

5.3 Domain schema

The domain schema represents concepts and relations regarding the PI context. Figure 5.2 shows the domain schema, the core of the PI system. We make the hypothesis that the global schema is a copy of the domain schema.

Although the domain schema is based on the pivot model presented in section 3.1.1 we use the following translations in order to make graphical notation easier to understand:



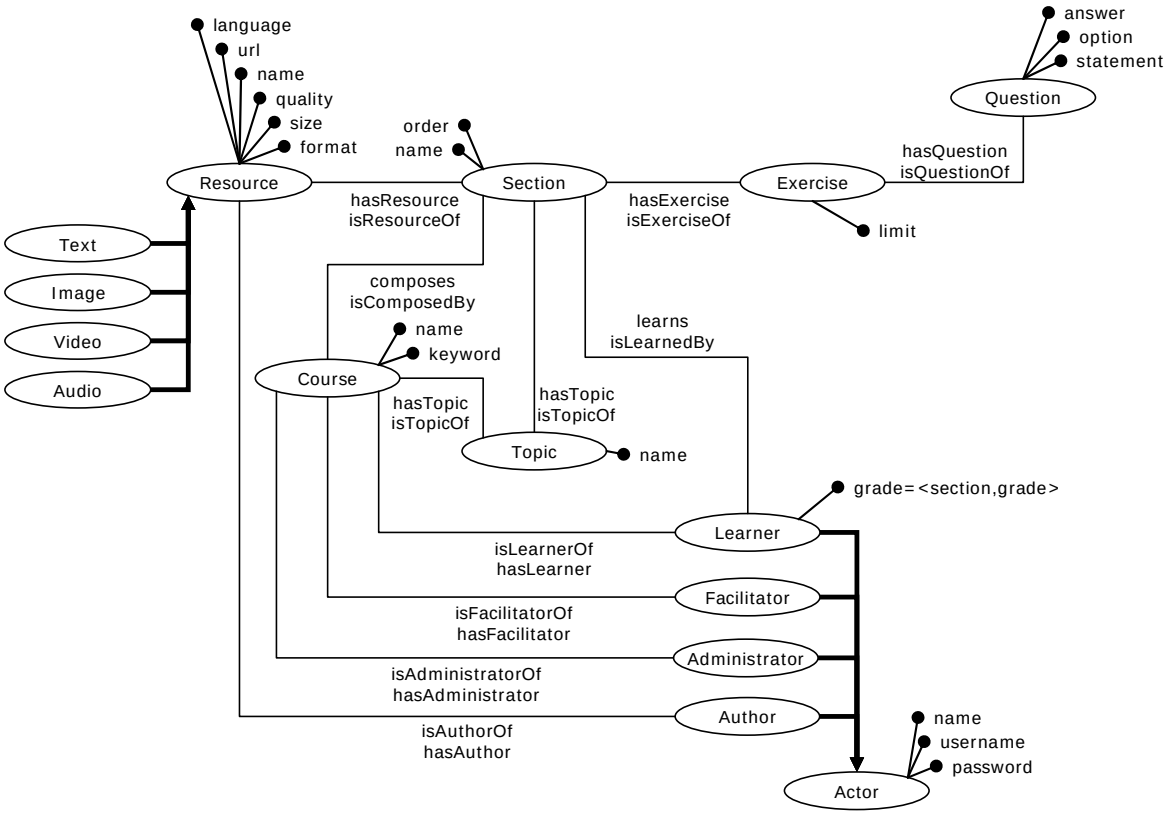


Figure 5.2 Domain schema.

A course is taken by learners, lead by a facilitator, managed by an administrator and composed by one or more sections. Each section is related to a set of resources and to an exercise composed by a set of questions.

5.3.1 Course

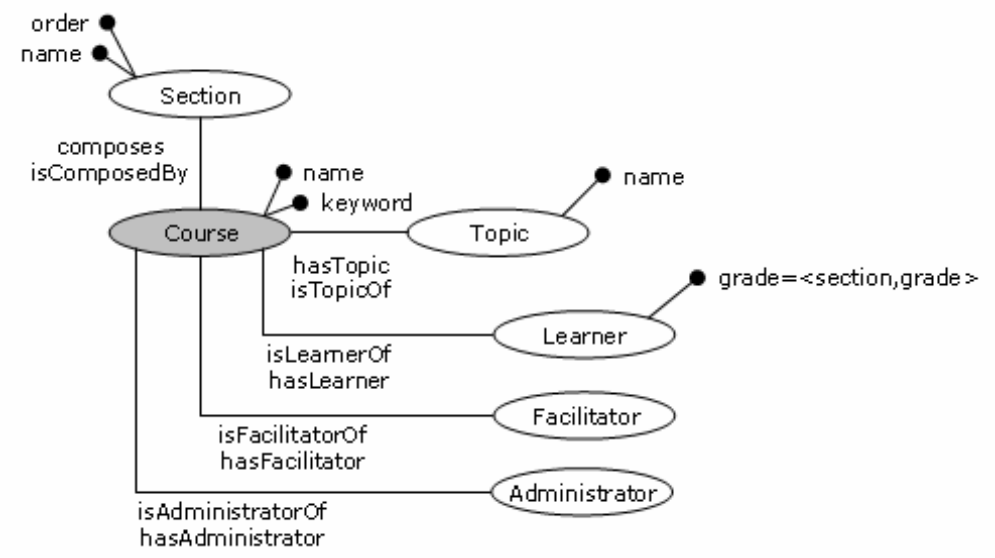


Figure 5.3 Course.

Course and their correspondent topics represent the courses offered by the system. A course is composed by a set of sections or unities, it is identified by a name and it is related to a set of keywords. Every course is managed by an administrator and it is related to the set of learners (students) that take the course and to the facilitator (teacher) that is responsible for the course.

5.3.2 Section

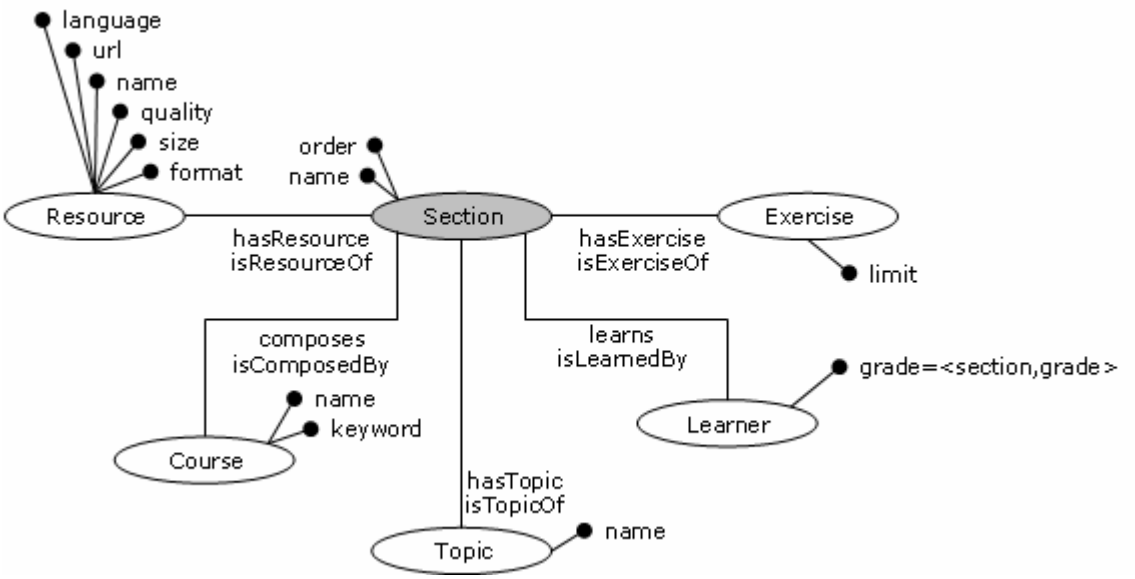


Figure 5.4 Section.

Section represents unities that compose courses. A section is identified by a name and can be part of one or more courses in a certain order. Each section has a topic and it is related to a set of resources and to an exercise.

5.3.3 Topic

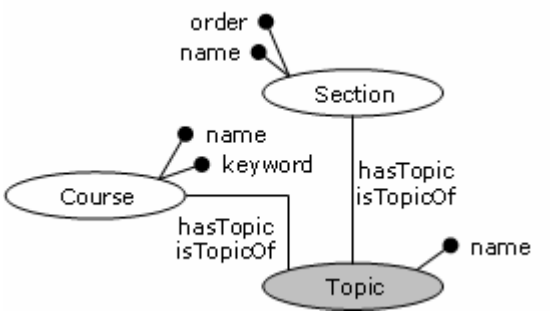


Figure 5.5 Topic.

This class is used to represent a list of topics. These topics correspond to the topics of courses and sections of the system. More than one course or section may share the same topic.

5.3.4 Exercise

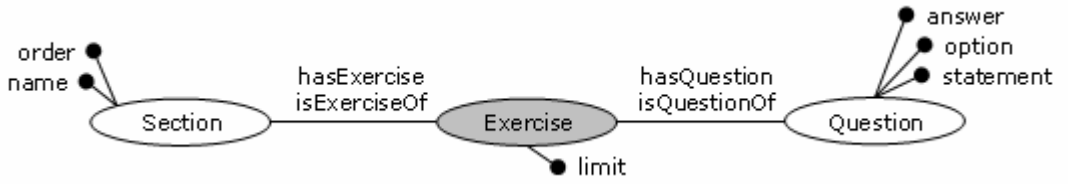


Figure 5.6 Exercise.

At the end of every section learners are given an evaluation to determine whether they continue with the following section. These evaluations consist of a set of questions and are represented by Exercise. Each exercise has a limit that determines the minimum needed correct answers for the learner to approve.

5.3.5 Question

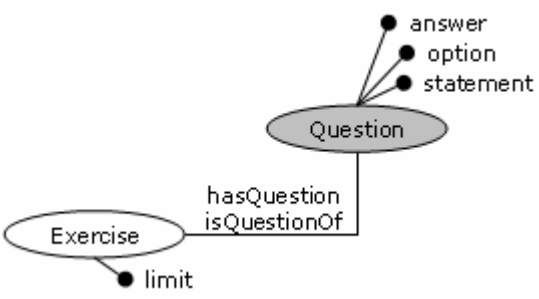


Figure 5.7 Question.

This class represents the questions we use to compose section exercises. Each question has a statement, a set of options and a correct answer. Normally a question belongs to only one exercise but there is not a restriction on the number of exercises they could be related to.

5.3.6 Resource

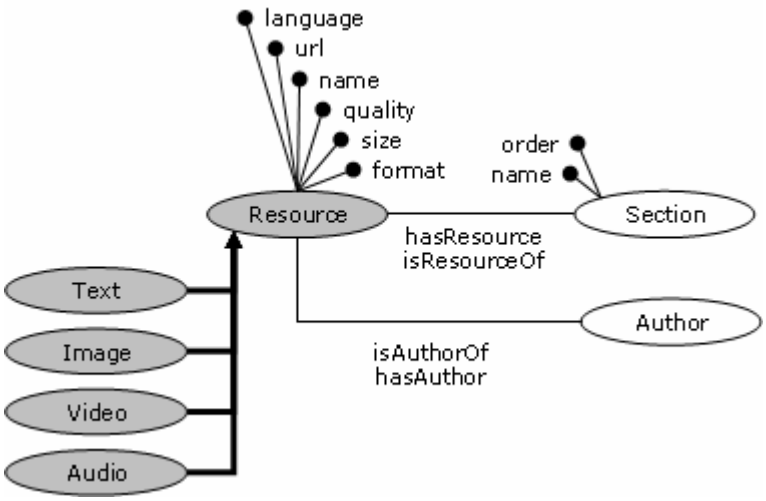


Figure 5.8 Resource.

This class represents a set of Semantic Web resources that are used by sections. Every resource is related to the actor who created it (i.e. its author). We consider four kinds of resources: text, image, video and audio. Text represents plain and rich text documents, presentations, worksheets and Web pages. Image, Video and Audio represent images, videos and audios of any format. Resources are described in terms of their name, location (i.e. url), format, language, quality and size. Except for location there is not a unique convention regarding the nature of the possible values of these attributes.

5.3.7 Actor

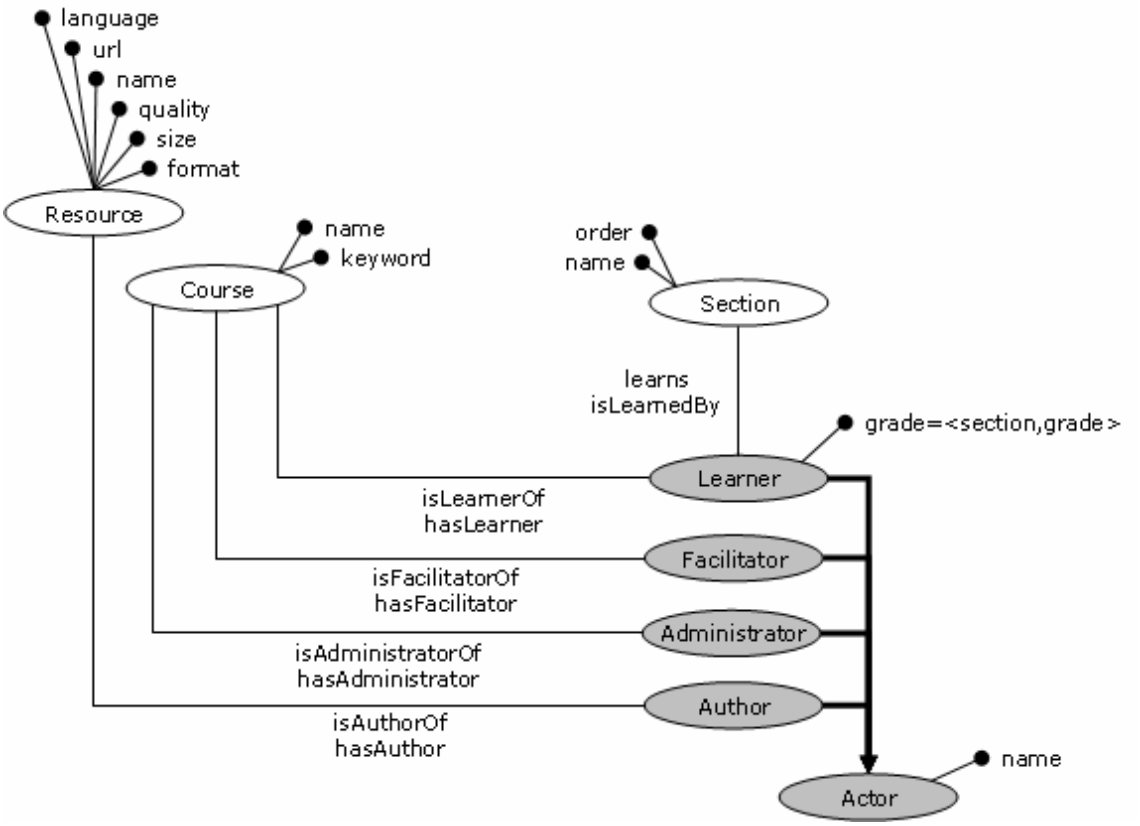


Figure 5.9 Actor.

Actor and its subclasses represent important actors of the system; every actor is identified by a name, an username and a password. Learners or students are related to the course they are taking, the section they are currently studying and the grades they got in previous sections. Facilitators or teachers are related to the course(s) they teach. Administrators are related to the course(s) they are responsible for. Authors are related to the resources they created.

5.4 Local schema

We propose nine sources as local schemas: resource, section, course, exercise, question, learner, facilitator, administrator and author. Mappings between the domain schema and

the local schemas are considered to be exact given the fact sources are subschemas of the domain schema. Although the domain schema is based on the pivot model presented in section 3.1.1 we use the translations presented in section 5.3 in order to make graphical notation easier to understand.

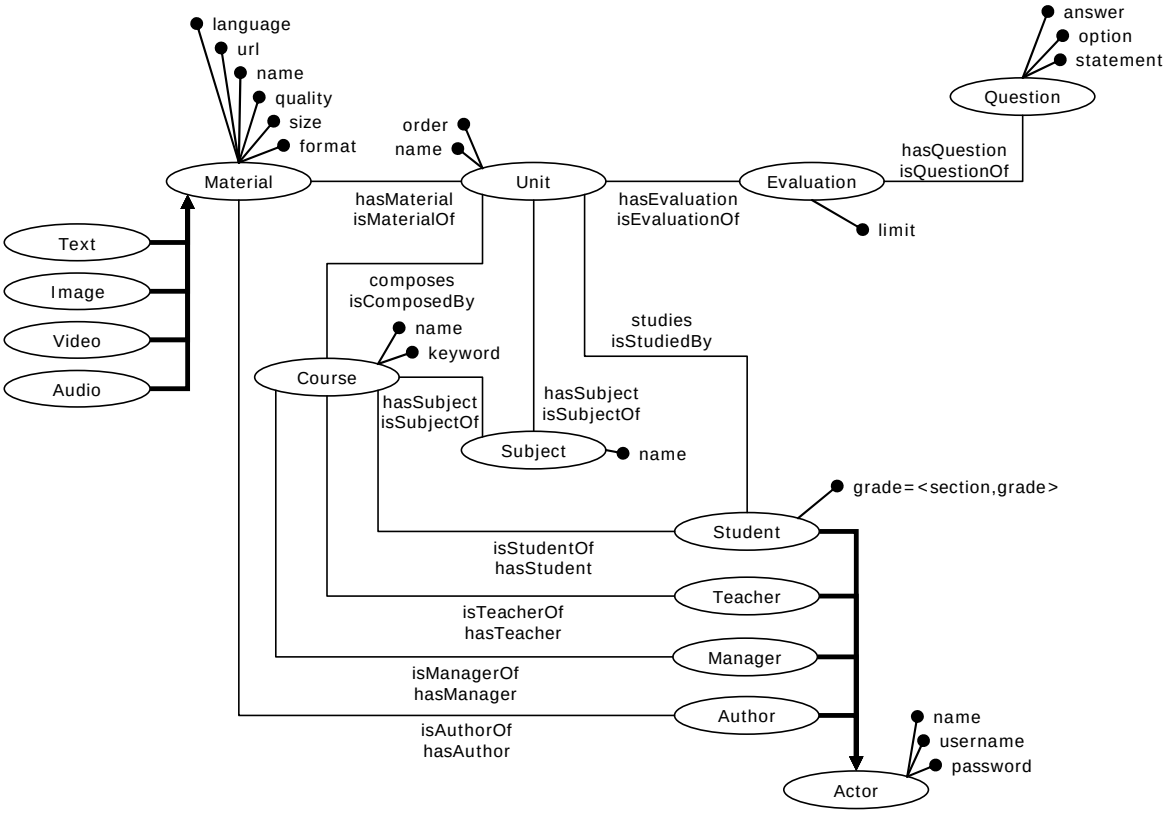


Figure 5.10 Local schema

A course is taken by students, lead by a teacher, managed by a manager and composed by one or more units. Each unit is related to a set of materials and to an evaluation composed by a set of questions.

5.5 Using the PI system

This section focuses on the use of the PI system. Before starting the PI system, MBF should be used to build a SKIMA. Once SKIMA is ready the PI system can be started. Figure 5.11 shows the PI system GUI.

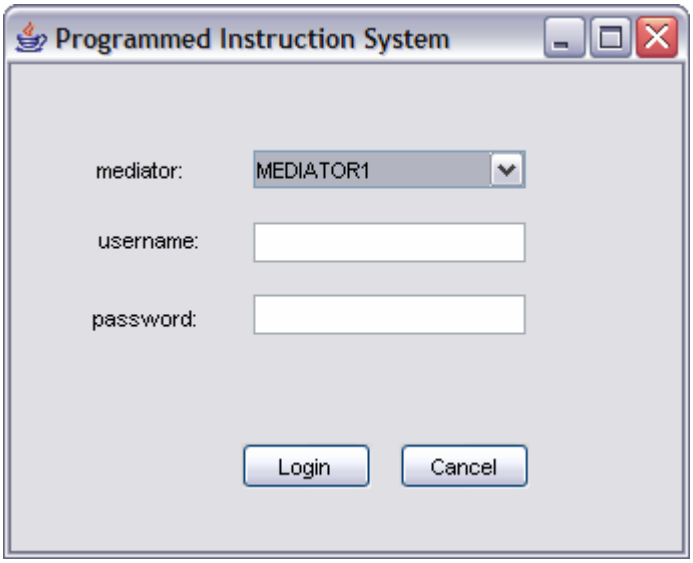


Figure 5.11 Login into the PI system.

In order to login into the PI system the user (i.e. a learner or student) has to choose one of the source descriptions MBF offers and to provide his/her username and password. If this information is valid the user enters the system where is able to work with a specific section of a given course.

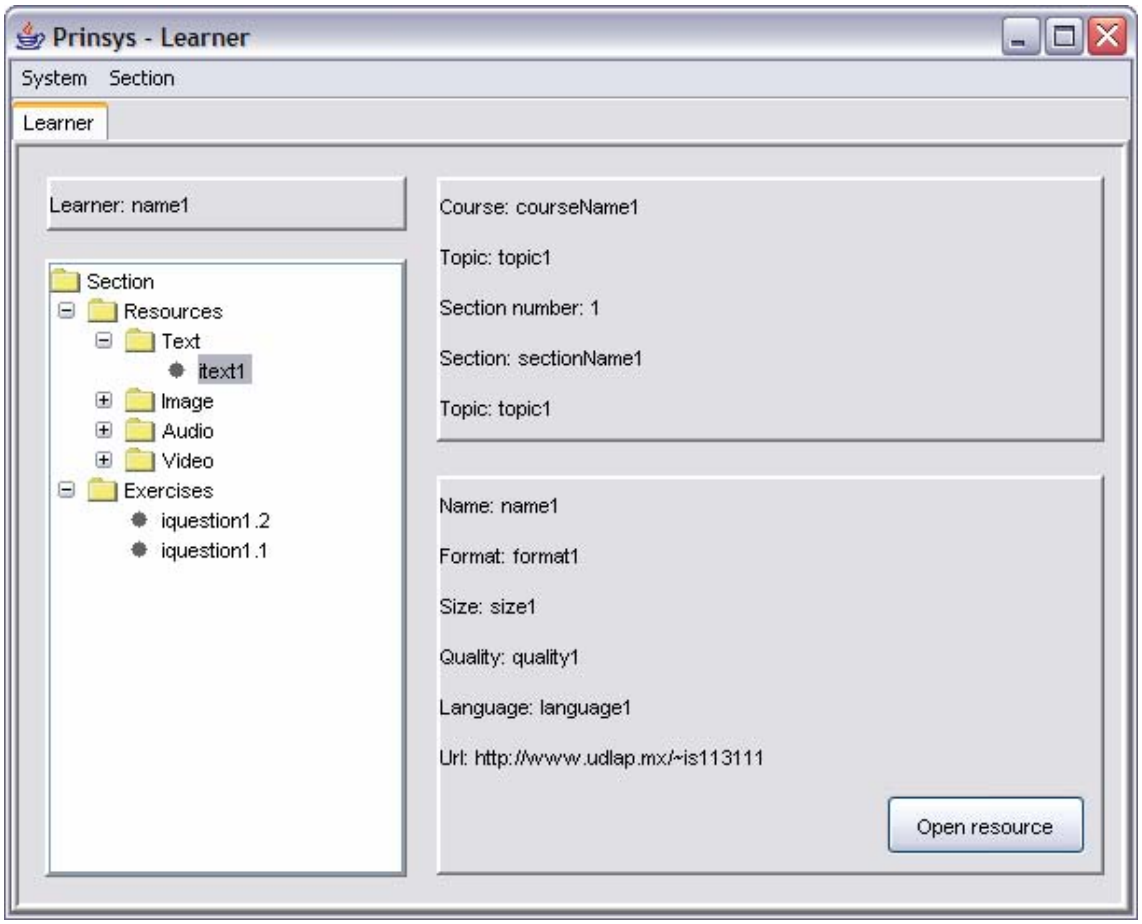


Figure 5.12 Working with resources in the PI system.

As it can be seen in figure 5.12 the PI system has two menus. Similarly the System menu has two items: Logout and Exit: Logout is used to logout from the system and return to the login screen while Exit is used to close the system. The Section menu allows users to reset the section (i.e. make all exercises available again) and evaluate the section.

The system shows the name of the user and information about the section and the correspondent course. It also shows the list of resources and exercises associated with the section. When the user clicks on any resource, the system retrieves its metadata and allows the user to open it in a Microsoft Internet Explorer window. In order to continue with the following section, any learner must take the section evaluation. An evaluation is composed by exercises or questions.

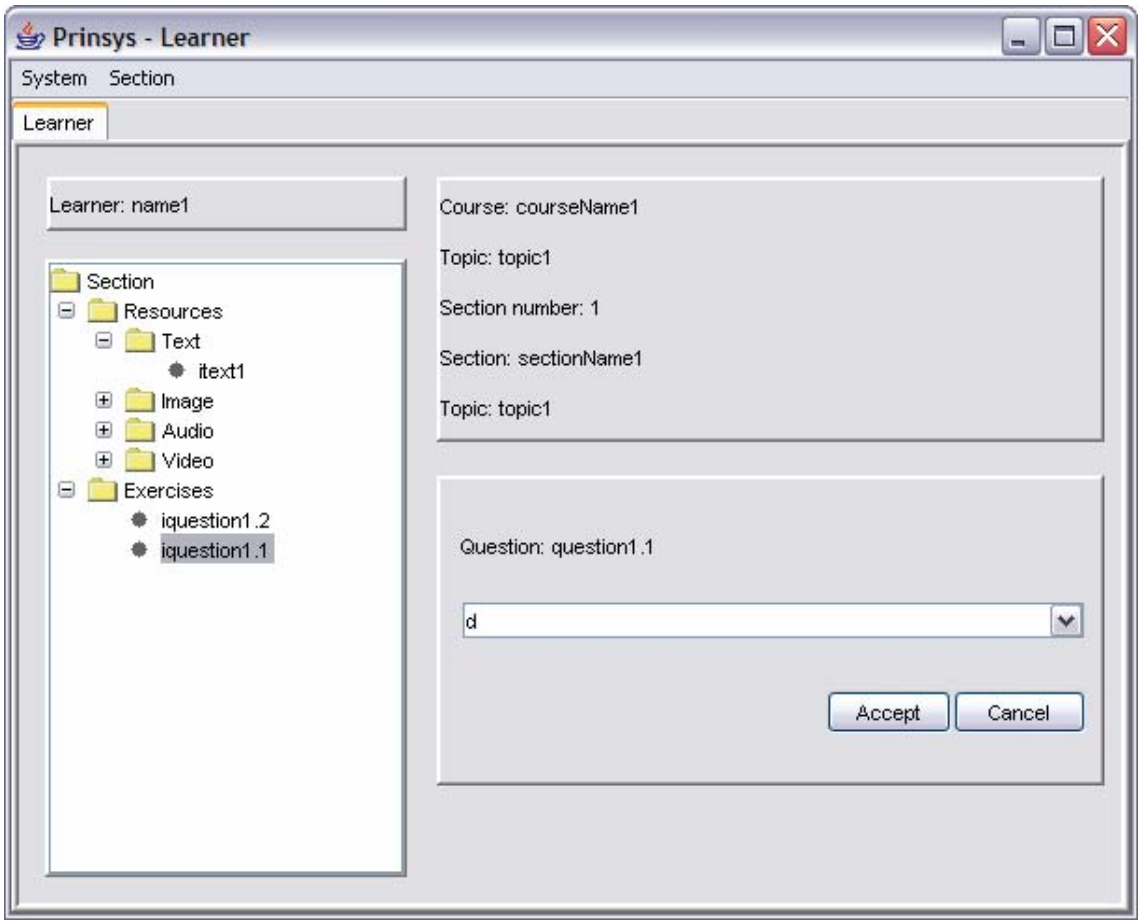


Figure 5.13 Taking the evaluation.

As it can be seen in figure 5.17 when the user clicks on an exercise the system shows a question and a list of possible answers (following the Pressey model). If the user chooses as many correct answers as defined in the domain schema he/she may continue with the following section (until of course there are not more sections to be taken). If the user fails he/she will have to recheck the resources and take the evaluation again. Once an exercise is answered it cannot be answered again (unless the user resets the evaluation).

5.6 Conclusions

The PI system uses SKIMA to allow learners or students to access Semantic Web resources that compose sections of courses. We provide the domain schema and the local schema.

The domain schema represents concepts of the PI context: courses, sections, exercises, questions, resources and actors. Local schema is composed by five different sources: all resources; courses and sections; sections and resources; learners and courses; and facilitators and courses. The system operates on top of SKIMA that was built with MBF based on a specific schema description.

6 Conclusions

The objective of this work was building a data mediation system based on semantics that allows applications to have transparent access to distributed, autonomous and heterogeneous sources composed by Semantic Web resources. The remainder of this chapter is organized as follows. Section 6.1 presents the results we obtained after the development of this work. Section 6.2 describes the perspectives we propose for future work.

6.1 Results

We think the objective of this work was accomplished with the development of SKIMA (Semantic Knowledge and Information Management) as a data mediation system based on semantics that allows application to have transparent access to Semantic Web resources. SKIMA is an effort for the development of the Semantic Web for it presents a scenario where data mediation techniques are used to exploit Semantic Web resources and semantics based on a description of application requirements.

The use of the Semantic Web facilitates the exploitation of many types of resources that can be easily selected in terms of their utility thanks to their annotated metadata. The idea is to obtain the needed information from the Semantic Web and to take advantage of the semantics to query it using SKIMA. Our approach allows users to reason over SKIMA using the inference engine giving them the possibility to both retrieve explicit knowledge and automatically discover implicit knowledge.

On the other hand we presented MBF (Mediator Building Framework) as a template for building mediators. MBF allows exploiting large collections of diverse Semantic Web resources for it is able to manage several schemas. Any user can choose a set of schemas and build a new mediator (i.e. SKIMA). Finally we presented a simple PI system as validation of our work. The PI system uses SKIMA to access Semantic Web resources.

6.2 Perspectives

As it was already said, we proposed a PI system as validation for our system. We provided a description of the domain and the sources. Nevertheless it could be interesting for SKIMA to be validated by experts in a certain domain. Experts would define a specific domain and a set of sources based on our pivot data model in order to decide whether SKIMA behaves as expected.

Including application requirements and source descriptions in a data mediation system gives the possibility of building ad hoc mediators that manage certain type of sources. Nevertheless reasoning with application and source semantics can be computational expensive specially where handling large amounts of information. This situation also motivates the interest of studying the impact that large amounts of information have on SKIMA performance. Besides, we consider it is important studying and implementing query optimization techniques compatible with our data model in order to make query processing as inexpensive as possible.

On the other hand perspectives related to MBF include the development of a mediator generator. MBF as a mediator building framework is a collection of schemas where building a mediator implies specializing the framework by choosing a set of schemas from its schema base. A mediator generator unlike a mediator builder allows the generation of autonomous mediation systems that implements independent architectures.

References

[ALF+01] Anido, L., Llamas, M., Fernández, M. J., Caeiro, M., Rodríguez, J., Santos, J., A Component Model for Standarized Web-Based Education. WWW10, ACM 1-58113-348-0/01/0005, 2001.

[Ay03a] Ayala, G., Educación Asistida por Computadora, http://mail.udlap.mx/~ayalasan/ambientesDeAprendizaje/cap01.html, 2003.

[Ay03b] Ayala, G., Modelado del Usuario y Personalización, http://mail.udlap.mx/~ayalasan/ambientesDeAprendizaje/cap02.html, 2003.

[AY96] Ayala, G., Yano, Y., Learner Models for Supporting Awareness and Collaboration in a CSCL Environment. Intelligent Tutoring Systems, Proceedings of the Third International Conference ITS 96, Montreal Canada, June 1996.

[Ba85] Bangert-Drowns, R. L., Meta-Analysis of Findings on Computer-Based Education with Precollege Students. Annual Meeting of the American Educational Research Association, Chicago, IL, March April 1985. (ED 263 905).

[Bat86] Batey, A., Building a Case for Computers in Elementary Classrooms: A Summary of What the Researchers and the Practitioners Are Saying. Second Leadership in Computer Education Seminar, Seattle, WA, December 1986.

[Be03] Bechhofer, S., FaCT and OilEd clients, University of Manchester, 2003.

[Be1] Berners-Lee, T., Semantic Web Road map, http://www.w3.org/DesignIssues/Semantic.html

[Be2] Berners-Lee, T., What the Semantic Web Can Represent, http://www.w3.org/DesignIssues/RDFnot.html

[Be3] Berners-Lee, T., Conceptual Graphs and the semantic Web, http://www.w3.org/DesignIssues/CG.html

[Be4] Berners-Lee, T., Web services – Semantic Web, http://www.w3.org/2003/Talks/0521-www-keynote-tbl/Overview.html

[BG02] Berlanga, A., García, F. J., Propuesta para la Creación de Espacios de Aprendizaje Adaptativos. 2002.

[BGL+90] Breitbart, Y., García-Molina, H., Litwin, W., Roussopoulos, N., Rusinkiewicz, M., Thompson, G., Wiederhold, G., Proceedings on the final report of the workshop on multidatabase and semantic interoperability, First workshop on multidatabase and semantic interoperability, November, 1990.

[BHL01] Berners-Lee, T., Hendler, J., Lassila, O., The Semantic Web, http://www.scientificamerican.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2, 2001.

[BN] Bechhofer, S., Ng, G., OilED, http://oiled.man.ac.uk/

[BN02] Baader, F., Nutt, W., Basic Description Logics, in the Description Logic Handbook, edited by F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider, Cambridge University Press, 2002, pages 47-100.

[Br1] Bremen University - semantic translation project, Project Description, http://www.tzi.de/buster/description.html

[Br2] Bremen University - semantic translation project, Enabling Technologies for Information Integration, http://www.tzi.de/buster/technologies.html

[Brt1] Bray, T., RDF and Metadata, http://www.xml.com/pub/a/98/06/rdf.html

[Ch1] Chang, W., A Discussion of the Relation Between RDF-Schema and UML, http://www.w3.org/TR/1998/NOTE-rdf-uml-19980804/

[Co01] Collet, C., A mediation framework for a transparent access to biological data source, the MEDIAGRID project, 2001.

[Co91] Cotton, K., Computer-Assisted Instruction: What the research shows. May 1991.

[CVZ03] Cerquitelli, T., Vargas, G., Zechinelli, J., Building multimedia data warehouses from distributed data, Avances en Ciencias de la Computación, IV Encuentro Internacional de Ciencias de la Computación, Tlaxcala, México, Noviembre, 1993.

[DAT+02] Dimitriadis, Y., Asensio, J. I., Toquero, J., Estébanez, L., Martín, T. A., Martínez, A., Hacia un Sistema de Componentes Software para el Dominio del Aprendizaje Colaborativo Apoyado por Ordenador (CSCL). Simposio de Informática y Telecomunicaciones SIT ’02, 2002.

[De+00] Decker, S. et. al., The Semantic Web: The Role of XML and RDF, IEEE Internet Computing, Sep. 2000, pp.63-74.

[DFH03] Davies, J., Fensel, D., Harmelen, F., Towards the Semantic Web: Ontology-driven Knowledge Management, copyright 2003 John Wiley & Sons, Ltd. ISBN: 0-470-84867-7

[DLNET03] DLNET, A Digital Library Network Engineering and Technology, http://aloha.netera.ca:16080/uploads/gregoir/IntroToLO_InDLNET.pdf

[DMM00] Decker, S., Mitra, P., Melnik, S., Framework for the Semantic Web: An RDF Tutorial, IEEE Internet Computing, Nov. 2000, pp. 68-73.

[FC00] Favela, J., Contreras J. J., Supporting Causal Interaction and Collaborative Information Exploration in Distributed Software Development Projects, 2000.

[Ga03] García, R., Exploring the Semantic Web, PhD Thesis status report, Doctorate Computer Science and Digital Communication, Universitat Pompeu Fabra, Barcelona, September, 2003

[Gr77] Grimes, D. M., Computers for Learning: The Uses of Computer Assisted Instruction (CAI) in California Public Schools. Sacramento, CA: California State Department of Education, 1977.

[Gr93] Gruber, T. R., A Translation Approach to Portable Ontology Specifications, 1993.

[Gu97] Guarino, N., Understanding, Building and Using Ontologies, In international journal of human and computer study, 1997.

[HH00] Heflin, J., Hendler, J., Semantic Interoperability on the Web, Proceedings of Extreme Markup Languages 2000, pp. 111-120.

[HM1] Haarslev, V., Möller, R., Racer, Semantic Middleware for Industrial Projects Based on RDF/OWL, http://www.sts.tu-harburg.de/~r.f.moeller/racer/

[HM2] Haarslev, V., Möller, R., Racer: A Core Inference Engine for the Semantic Web.

[IEEE02] IEEE Learning Technology Standards Committee (LTSC), IEEE 1484.12.1-2002, Draft Final for Learning Object Metadata, http://ltsc.ieee.org/wg12/files/LOM_1484_12_1_v1_Final_Draft.pdf

[La1] Lassila, O., Introduction to RDF Metadata, http://www.w3.org/TR/NOTE-rdf-simple-intro

[Lan03] Langan, T., Center for Instructional Development & Delivery, Wisconsin Online Resource Center Interactive Learning Objects, http://www.wisc-online.com

[LEL02] López, A., Escalera, S., Ledesma, R., Ambientes virtuales de aprendizaje. Presimposio Virtual SOMECE 2002.

[MCH03] Möller, R., Cornet, R., Haarslev, V., Graphical interfaces for Racer: querying DAML+OIL and RDF documents. In proceedings of the International Workshop on Description Logics – DL '03, 2003.

[MD00] Melnik, S., Decker, S., A Layered Approach to Information Modeling and Interoperability on the Web, Proc. of the ECDL'00 Workshop on the Semantic Web, 2000.

[NS] Nuñez, G., Shemeretov, L., Ambiente de enseñanza – aprendizaje cooperativo personalizado para la educación superior, http://www.anuies.mx/anuies/revsup/res111/txt3.htm

[Pa01] Palmer, S., The Semantic Web: An Introduction, http://infomesh.net/2001/swintro/, 2001.

[Pol00] Polsani, P., Use and Abuse of Reusable Learning Objects. University of Arizona, USA, Learning Technology Center, 2000.

[PS00] Perini, T., Schürman, P., The learning environment OLAT (On line Learning and Testing). DETECH, 2000.

[RMI] An Overview on RMI Applications, http://java.sun.com/docs/books/tutorial/rmi/overview.html

[SA02] Saito, N., Ayala, G., Design an Implementation of a Life Long Learning Environment using the JSP Model 2 Architecture. In Proceedings of the XII Congreso Internacional de Ingeniería Electrónica, Comunicaciones y Computadoras, IEEE Puebla y UDLAP, 2002.

[SSU91] Silberschatz, A., Stonebraker, M., Ullman, J., Database systems: Achievements and opportunities, Communications of the ACM, 34(10), October, 1991.

[SYE+90] Scheuermann, P., Yu, C., Elmagarmind, A., García-Molina, H., Manola, F., McLeod, D., Rosentahl, A., Templeton, M., Report on the workshop on heterogeneous database systems, SIGMOD record, 19, December, 1990.

[VDD02] Vargas-Solar, G., Dobre, A., Dittrich, K., Towards a content management infrastructure for learning environments. In proceedings of the 14th International Conference on New Education Environments, 2002.

[VZ99] Vargas, G., Zechinelli, J., Data bases: current trends and perspectives in the information age, Laboratory Logiciels Systèmes Réseaux and UDLAP, 1999.

[W3C] World Wide Web Consortium, Semantic Web, http://www.w3.org/2001/sw/

[W3C04] World Wide Web Consortium, OWL Web Ontology Language Overview, http://www.w3.org/TR/owl-features/

[Wi94] Wiederhold, G., Interoperation, mediation, and ontologies, Published in the Proceedings International Symposium on Fifth Generation Computer Systems (FGCS), Workshop on Heterogeneous Cooperative Knowledge-Bases, Vol. W3, pages 33-48, ICOT, Tokyo, Japan, Dec., 1994.

[Wil00] Wiley, D., Connecting Learning Objects to Instructional Design Theory: A Definition, Metaphor and a Taxonomy. Digital Learning Environments Research Group, The Edumetrics Institute, p. 4, 2000.

Appendix A Interfaces

A.1 SKIMA

SKIMA is composed by several modules that perform specific tasks in interaction with an inference engine. These modules provide a set of functions that can be used by applications according to their specific permissions. Functions are based on an abstract representation of SKIMA (see figure A.1). Although this representation is based on the pivot model presented in section 3.1.1 we use the translations presented in section 5.3 in order to make graphical notation easier to understand.

A.1.1 Abstract representation

As it can be seen in figure A.1 SKIMA manages a domain which has users. Users have specific permissions over certain functions. Functions belong to interfaces which are implemented by components. The mediator itself is a component that generates internal components. Components are represented by schemas. The mediator preferences about the domain are contained in its view definition while the sources to be used are described by a source description.

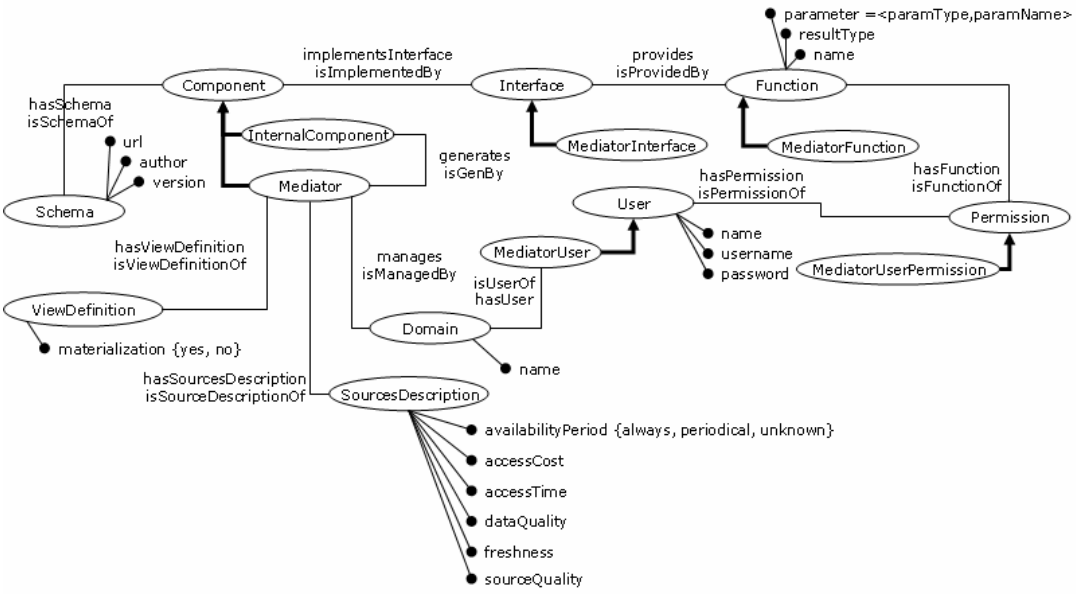


Figure A.1 Abstract representation of SKIMA.

Component

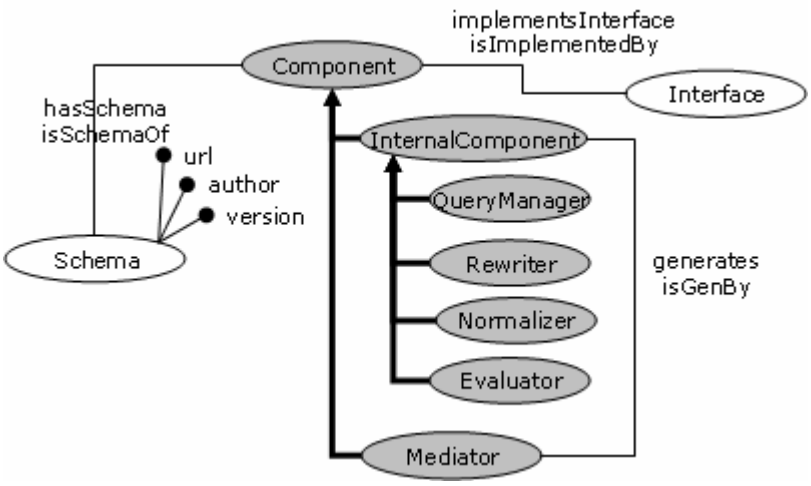


Figure A.2 Component.

Component represents the mediator and its components. Although the mediator is a component itself, it is not the only one, for it generates internal components. Component has two subclasses: Mediator and InternalComponent. In a similar way InternalComponent has four subclasses corresponding to the internal components of the mediator: the query manager, the rewriter, the normalizer and the evaluator. Details on each of these internal components will be given later on.

ViewDefinition

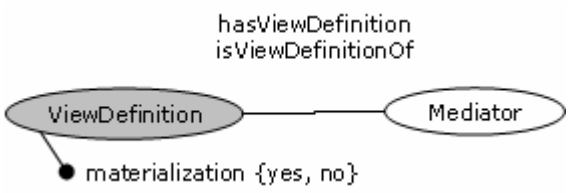


Figure A.3 ViewDefinition.

ViewDefinition defines if the mediator handles a materialized or a not materialized domain schema. A materialized schema has data already stored while a not materialized schema does not.

SourcesDescription

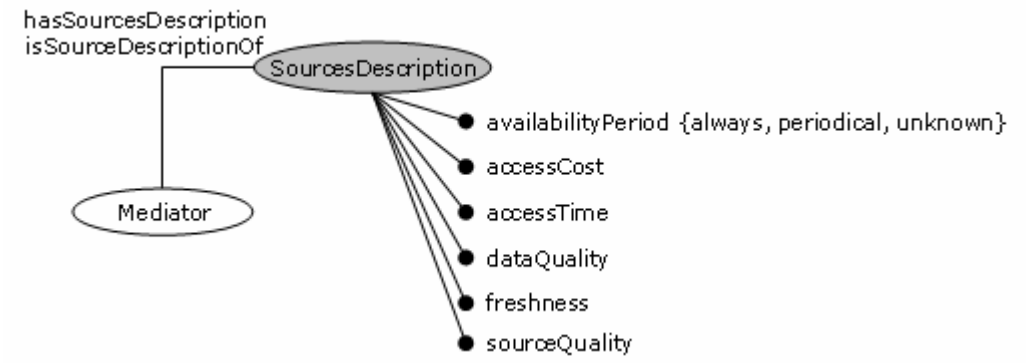


Figure A.4 SourcesDescription.

This class represents a set of sources that match a given description. These sources are managed by the mediator. The description is made in terms of availability, access cost, access time, data quality, source quality and freshness. There is not a unique convention regarding the nature of the possible values of these attributes. Access time can be given in seconds or milliseconds, access cost can be given in pesos or dollars, etc. it all depends on hypothesis made a priori.

Schema

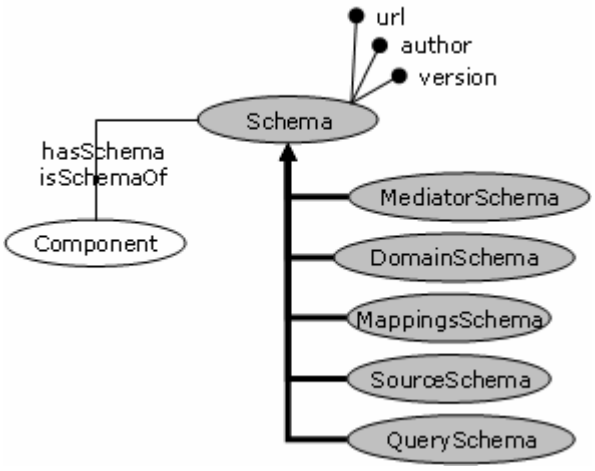


Figure A.5 Schema.

Schema has subclasses that contain metadata about the schemas we use throughout the mediation process. Schema's subclasses have metadata about the mediator, the sources, the domain, the ontologies used to handle mappings between the domain and the sources, and the ontologies used to handle queries on the mediator. Details on sources, queries and mappings will be discussed later on.

Domain

Figure A.6 Domain.

Domain represents the domain that is handled by the mediator. Our mediator manages an ontology apart dedicated to the domain (the domain schema) that we will discuss later.

User

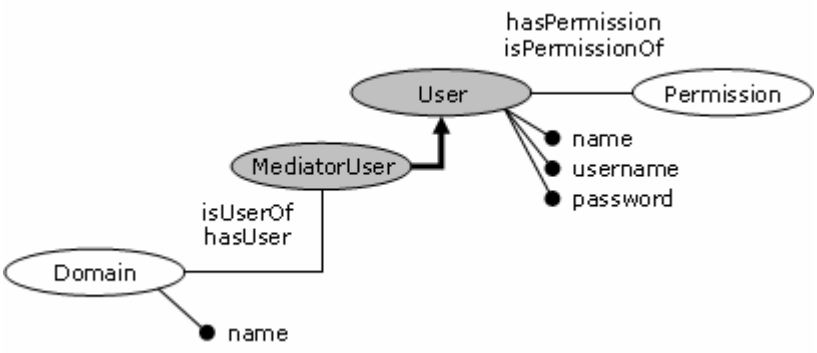


Figure A.7 User.

User and its subclass MediatorUser represent the set of people that use the mediator. Users are described in terms of their name, their username and their password. These users have permissions over certain functions and are related to a specific domain as the mediator.

Permission

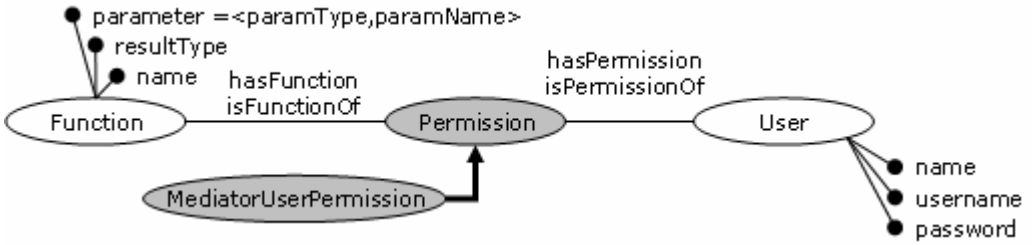


Figure A.8 Permission.

Permission and its subclass MediatorUserPermission are used to establish different types of users in terms of the functions they can use. The idea is that each user has a specific permission which is related to a set of functions (the functions the user is allowed to use). Many types of permissions can be defined and with them many kinds of users.

Function

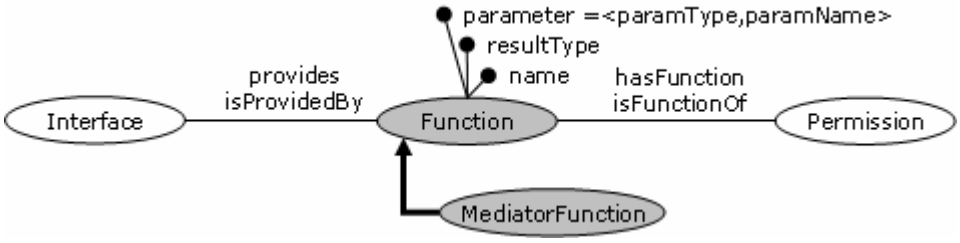


Figure A.9 Function.

Function and its subclass MediatorFunction represent the functions the mediator offers. Functions are described in terms of their name, their result type and their parameters. Each function is provided by a specific interface.

Interface

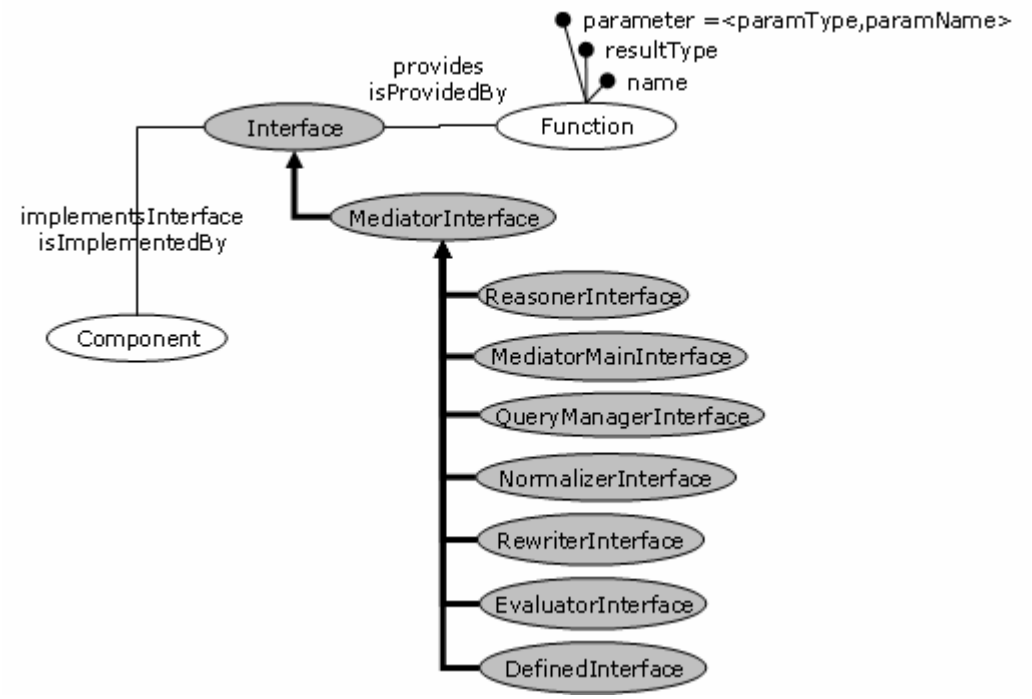


Figure A.10 Interface.

Interface and its subclass MediatorInterface represent the set of interfaces implemented by the mediator components (including the mediator itself). Each instance of Interface is related to the set functions that belong to this interface. We will present each of these interfaces and their functions later on.

A.1.2 SKIMA modules

SKIMA is composed by several modules that perform specific tasks in interaction with an inference engine. These modules provide a set of functions that can be used by mediator users according to their specific permissions.

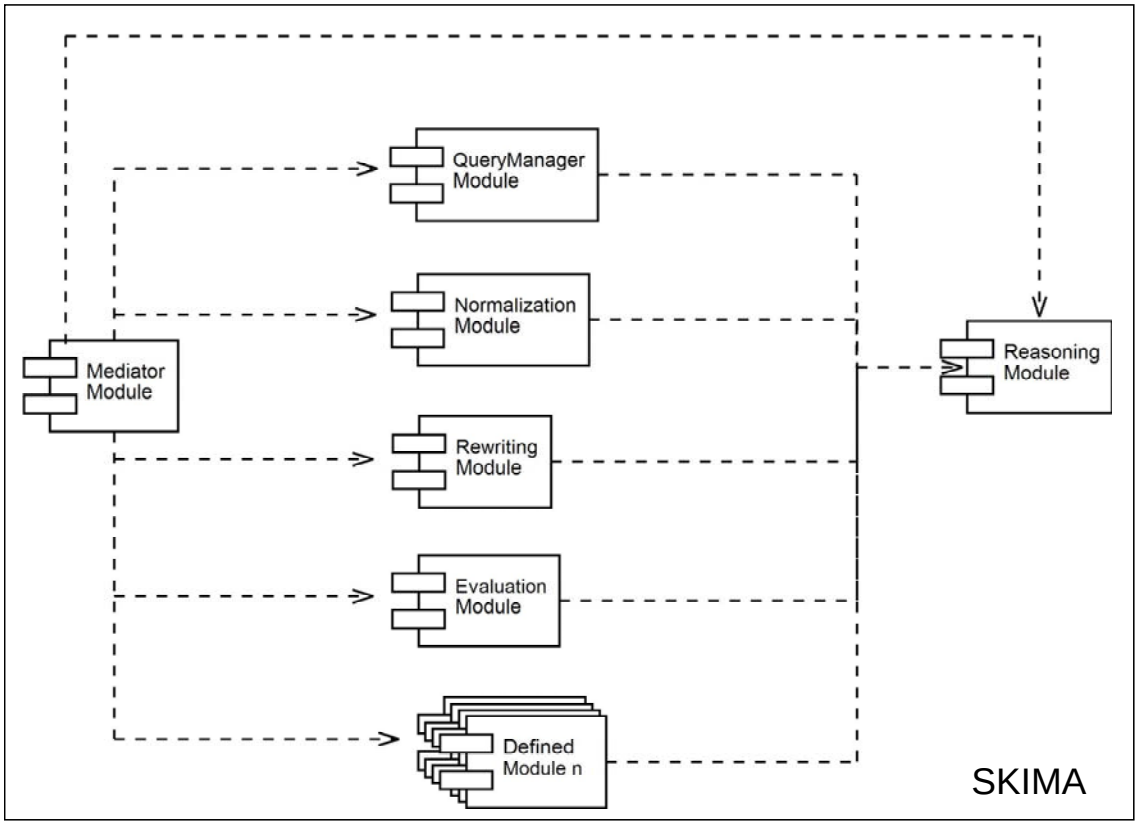


Figure A.11 SKIMA modules.

Mediator module

The mediator module provides the mediator main interface. It allows creating queries and to get other interfaces in order to execute their functions.

- **Mediator main interface**

  String createQuery(Query query)

  Interface getInterface(String username, String password, String interface)

Query manager module

The query manager module provides the query manager interface. It allows verifying queries after creation and to get query metadata. In this phase the query is analyzed semantically, the query is valid if (1) it respects global constraints expressed in terms of

axioms on concepts and roles and (2) it can be classified under one or more existing concepts.

- **Query manager interface**

  String createQuery(Query query)

  Query getQuery(String queryId)

  QueryProjection getQueryProjection(String queryId)

  QueryConceptTree getQueryConceptTree(String queryId)

  QueryFilter getQueryFilter(String queryId)

  String[] getIdsByQuery(Query query)

  String[] getAllQueryIds()

  String[] getQueryVariableIds(String queryId)

  String[] getQueryVariableDescs(String queryId)

  String getQueryVariableDescById(String variableId)

  String getQueryVariableRootId(String queryId)

  String getQueryVariableParentId(String variableId)

  String[] getQueryVariableChildrenIds(String variableId)

  String[] getQueryVariableMaterializedInstances(String variableId)

  String[] getQueryVariableMaterializedInstances(String variableId, String[] childrenIds)

  String[] getQueryVariableLocalInstances(String variableId, String rewriting)

String[] getQueryVariableLocalInstances(String variableId, String[] childrenIds)

Normalization module

The normalization module provides the normalization interface. In this phase we get semantic global mappings for the query, these mappings can be (1) equivalent concepts, (2) approximate concepts or (3) incomplete concepts.

- **Normalization interface**

  boolean isNormalizedQuery(String queryId)

  boolean isNormalizedQueryNode(String variableId)

  String[] getEquivalentQueryNodes(String variableId)

  String[] getCompleteQueryNodes(String variableId)

  String[] getSoundQueryNodes(String variableId)

Rewriting module

The rewriting module provides the rewriting interface. It allows rewriting the mappings we got in the normalization phase. There are three types of rewritings as there are of mappings: (1) equivalent rewritings (i.e. synonyms), (2) approximate rewritings and (3) incomplete rewritings.

- **Rewriting interface**

  Query[] getRewritings(String queryId)

  String[] getQueryNodeRewritings(String variableId)

  String[] getQueryNodeExactRewritings(String variableId)

  String[] getQueryNodeCompleteRewritings(String variableId)

String[] getQueryNodeSoundRewritings(String variableId)

Evaluation module

The evaluation module provides the evaluation interface. It allows evaluating queries after rewriting to get the results. Evaluation implies to retrieve a set of instances from a given schema using the inference engine.

- **Evaluation interface**

  boolean evaluates(String queryId)

  boolean evaluatesQueryNode(String variableId)

  String[] getQueryNodeResults(String variableId)

  void discardQueryNodeResults(String variableId)

  boolean materializesQueryNodeResults(String variableId)

Defined modules

Defined modules are defined by users and can interact with other modules. Any user can define a new module. Doing so implies to define and implement a specific interface that contains functions according to a set of needs. These functions are completely dependent on the user and can be a set of calls to functions from other modules or totally independent.

We have one defined module called generic functions module. This module provides the generic functions interface. It allows verifying, normalizing, rewriting and evaluating queries over local sources in a single function.

- **Generic functions interface**

  String executeQuery(String queryId)

  String executeQueryOnMaterializedData(String queryId)

Reasoning module

The reasoning module provides the reasoner interface. It is used by other modules to interact with the inference engine that contains metadata of the mediator and its components. It can also be used directly by certain authorized users.

- **Reasoner interface**

  String loadSchema(String url)

  String[] conceptDescendants(String boxname, String c)

  String[] conceptInstances(String boxname, String c)

  String[] individualFillers(String boxname, String i, String r)

  void cloneKB(String boxname, String newname)

  String[] individualDirectTypes(String boxname, String i)

  String getConceptDefinition(String boxname, String c)

  void removeCompleteIndividual(String boxname, String i, String c)

  Vector getNeeds()

  void forgetRoleAssertion(String boxname, String id1, String id2, String role)

  void addRoleAssertion(String boxname, String id1, String id2, String role)

A.2 MBF

MBF is composed by several modules that perform specific tasks in interaction with an inference engine. These modules provide a set of functions that can be used by users according to their specific permissions. Functions are based on an abstract representation of MBF (see figure A.12). Although this representation is based on the pivot model

presented in section 3.1.1 we use the translations presented in section 5.3 in order to make graphical notation easier to understand.

A.2.1 Abstract representation

MBF, as the mediator, is composed by a set of components, users, permissions, functions, interfaces and schemas; it can handle one or more domains and could manage many sources.

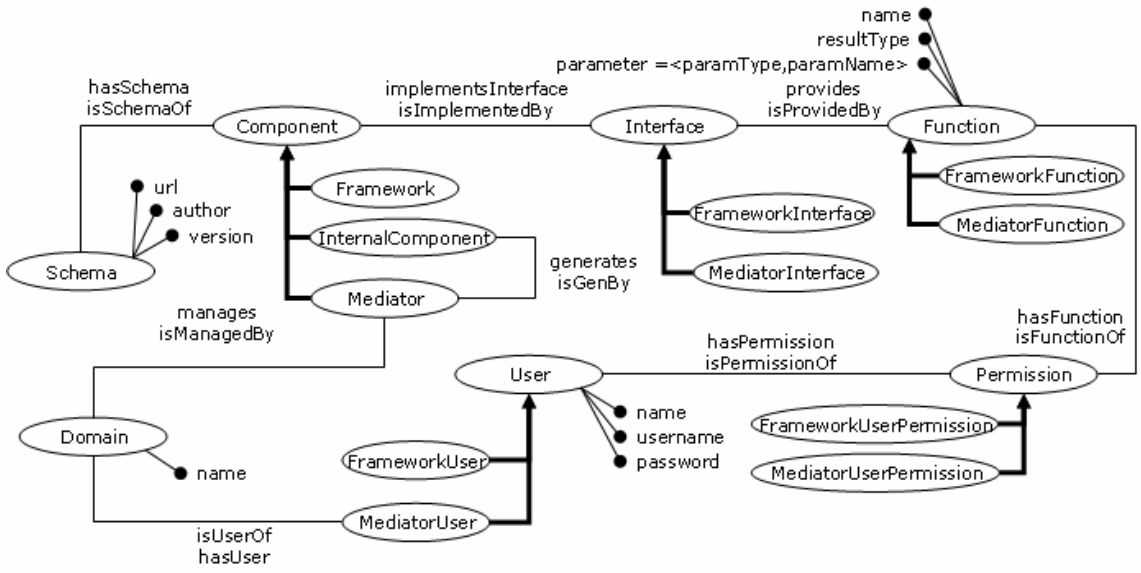


Figure A.12 Abstract representation of MBF.

MBF has users who have specific permissions over certain functions. The framework has a set of mediators each of such manages a domain which has users. Mediator users have specific permissions over certain functions. Each mediator generates internal components. Mediator and Framework functions belong to interfaces which are implemented by components, including the mediator and the framework themselves, represented by schemas.

MBF representation is very similar to that of SKIMA, as it can be seen we included new classes and removed others. The new classes are Framework, FrameworkUser, FrameworkPermission, FrameworkFunction, FrameworkInterface with its subclasses,

FrameworkSchema, NeedsSchema and SourcesMetadataSchema. We removed ViewDefinition and SourcesDescription from this schema to place them in the needs schema that will be discussed later on. We placed Framework as subclass of Component, FrameworkUser as subclass of User, FrameworkPermission as subclass of Permission, FrameworkFunction as subclass of Function, FrameworkInterface as subclass of Interface and FrameworkSchema, NeedsSchema and SourcesMetadataSchema as subclasses of Schema.

Component

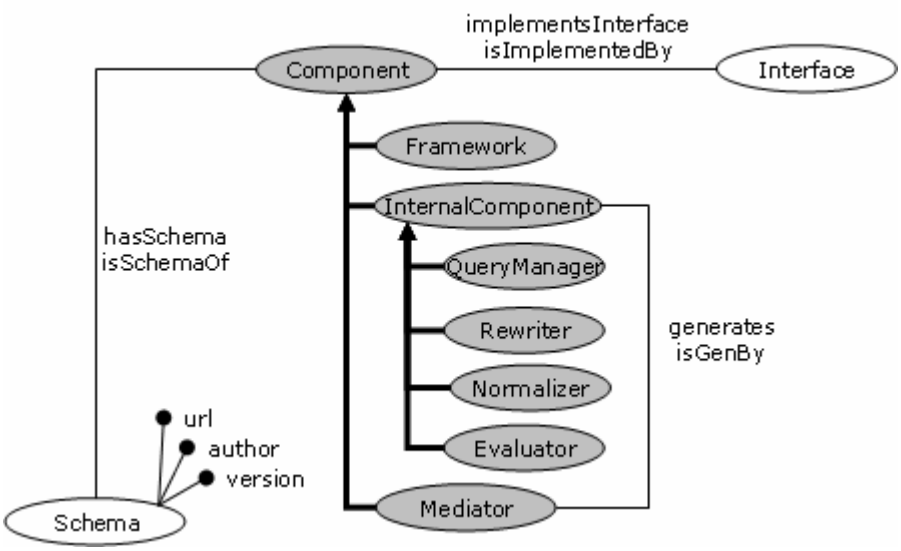


Figure A.13 Component.

Component represents the framework, its mediators and their components. Component has three subclasses: Framework, Mediator and InternalComponent.

Schema

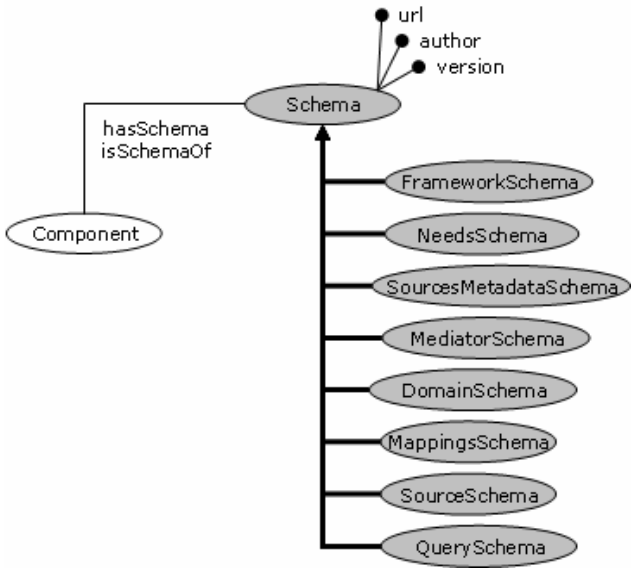


Figure A.14 Schema.

Schema has subclasses that contain metadata about the schemas we use throughout the mediator building process. Schema's subclasses have metadata about the framework, the needs to build mediators, the mediators, the used sources, the sources description, the domains, the ontologies used to handle mappings between the domains and the sources, and the ontologies used to handle queries on the mediators.

Domain

Figure A.15 Domain.

Domain represents the domain that is handled by a mediator. Our framework can manage many mediators; hence it can manage many domains.

User

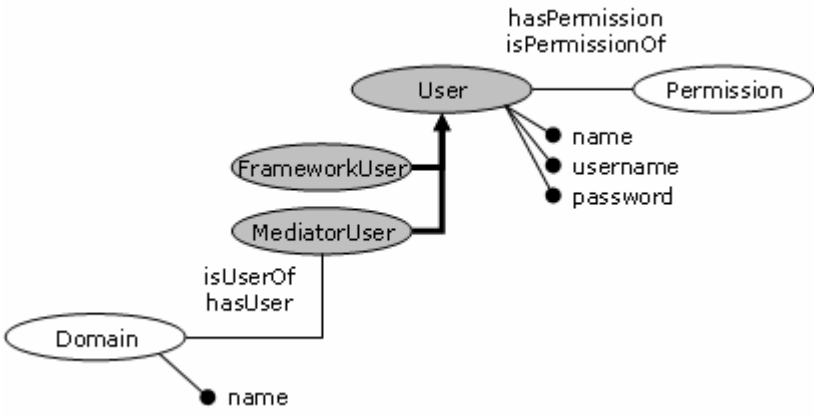


Figure A.16 User.

User and its subclasses FrameworkUser and MediatorUser represent the set of people that use the framework and that will eventually use a mediator. Users are described in terms of their name, their username and their password. These users have permissions over certain functions.

Permission

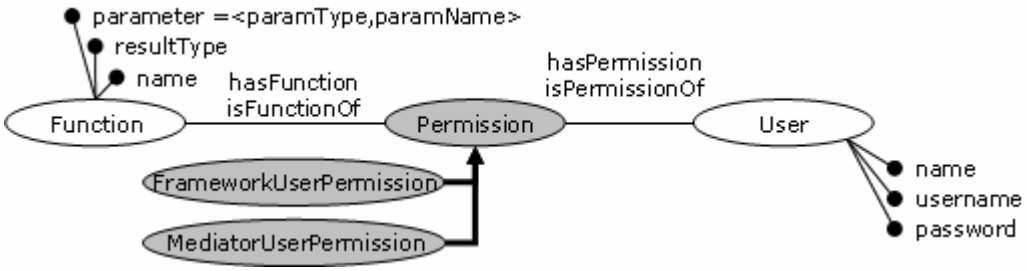


Figure A.17 Permission.

Permission and its subclasses FrameworkUserPermission and MediatorUserPermission are used to establish different types of users in terms of the functions they can use. The idea is that each user has a specific permission which is related to a set of functions (the functions the user is allowed to use). Many types of permissions can be defined and with them many kinds of users.

Function

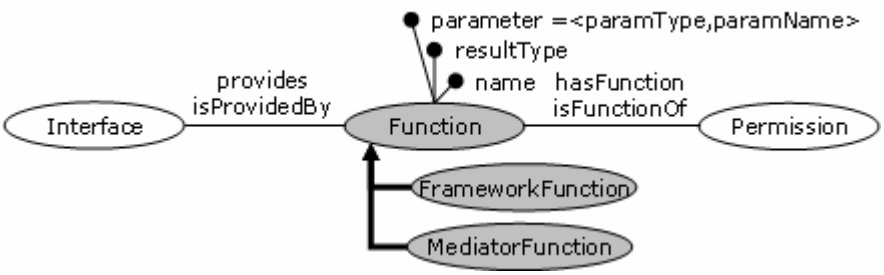


Figure A.18 Function.

Function and its subclasses FrameworkFunction and MediatorFunction represent the functions the framework and its mediators offer. Functions are described in terms of their name, their result type and their parameters. Each function is provided by a specific interface.

Interface

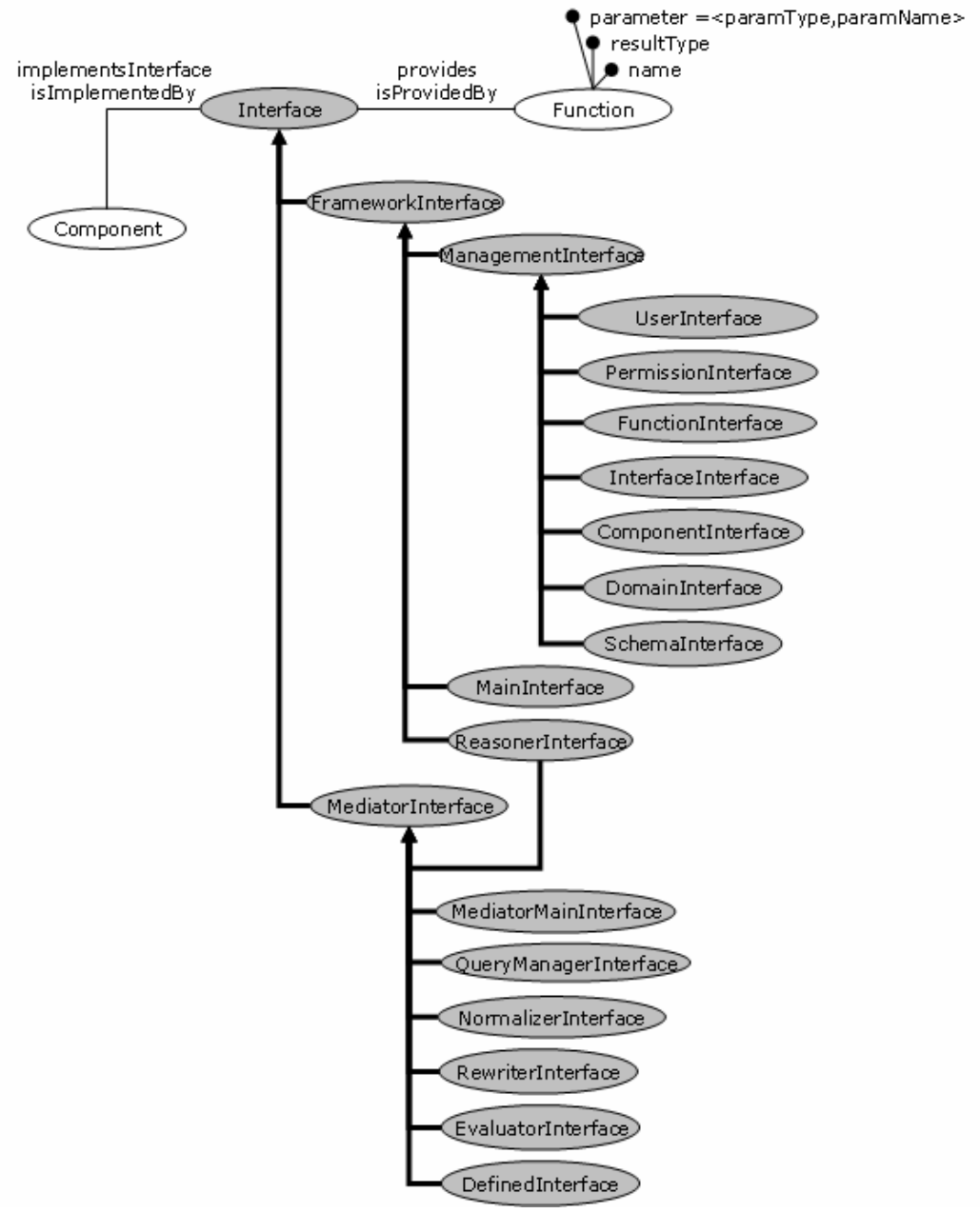


Figure A.19 Interface.

Interface and its subclasses FrameworkInterface and MediatorInterface represent the set of interfaces implemented by the framework components (including the framework itself, its mediators and their internal components). Each instance of Interface is related to the set

functions that belong to this interface. We will present each of these interfaces and their functions later on.

A.2.2 MBF modules

MBF is composed by SKIMA modules and by two new modules (see figure A.20). These new modules provide a set of functions that can be used by framework users according to their specific permissions. SKIMA modules were explained in section A.1.2, therefore we will focus on the other two modules: framework and framework management.

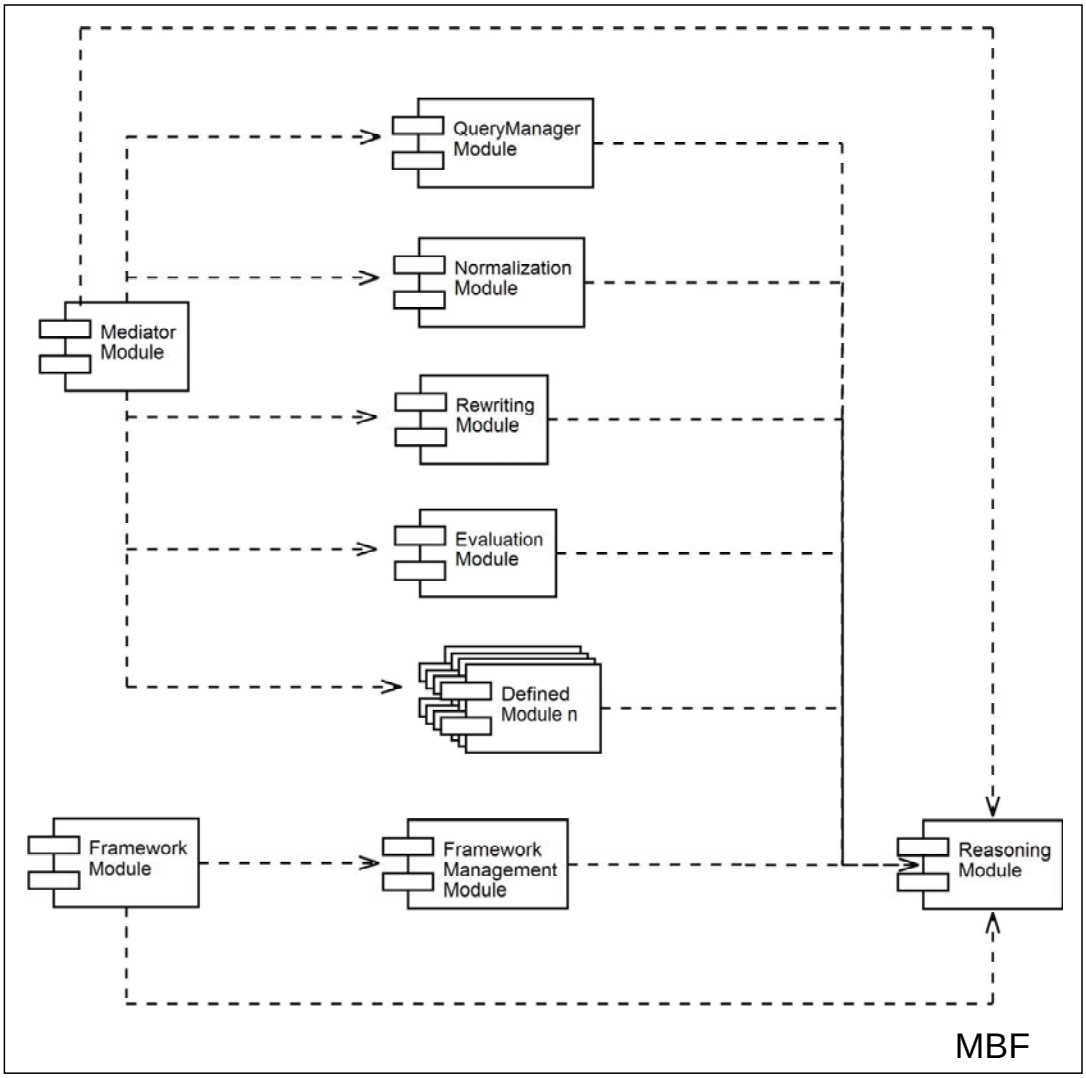


Figure A.20 MBF modules.

Framework module

The framework module provides the framework main interface. It allows subscribing domains, sources and mappings; defining a set of requirements describing a specific mediator and building a mediator based on a given set of requirements.

- **Framework main interface**

  void subscribeDomain(String name, String url)

  void subscribeSource(String availability, String accessCost, String accessTime, String dataQuality, String sourceQuality, String freshness, String url)

  void subscribeMapping(String url)

  void defineNeed(String domain, String[] mappings, String[] conceptSet, String[] functionSet, boolean materialization, String availabilityPeriod, String accessCost, String accessTime, String dataQuality, String sourceQuality, String freshness)

  void generateMediator(String name, String need)

Framework management module

The framework management module provides several interfaces that allow managing metadata of the framework and its components. These interfaces are used to query, add and remove users, permissions, functions, interfaces, components, domains and schemas.

- **User interface**

  void addUser(String type, String id, String name)

  void removeUser(String type, String user)

  void hasPermission(String user, String permission)

  void forgetHasPermission(String user, String permission)

void isUserOf(String domainUser, String domain)

void forgetIsUserOf(String domainUser, String domain)

String getUserName(String user)

String[] getAllUsers()

String[] getUsersByType(String type)

String[] getDomainUserDomains(String domainUser)

String[] getUserPermissions(String user)

- **Permission interface**

void addPermission(String type, String id)

void removePermission(String type, String permission)

void hasFunction(String permission, String function)

void forgetHasFunction(String permission, String function)

void isPermissionOf(String permission, String user)

void forgetIsPermissionOf(String permission, String user)

String[] getAllPermissions()

String[] getPermissionsByType(String type)

String[] getPermissionUsers(String permission)

String[] getPermissionFunctions(String permission)

- **Function interface**

  void addFunction(String type, String id, String name, String resultType, String[] aramNames, String[] paramTypes)

  void removeFunction(String type, String function)

  void isFunctionOf(String function, String permission)

  void forgetIsFunctionOf(String function, String permission)

  void isProvidedBy(String function, String inter)

  void forgetIsProvidedBy(String function, String inter)

  String getFunctionName(String function)

  String getFunctionResultType(String function)

  String[] getFunctionParameters(String function)

  String[] getAllFunctions()

  String[] getFunctionsByType(String type)

  String[] getFunctionPermissions(String function)

  String[] getFunctionInterfaces(String function)

- **Interface interface**

  void addInterface(String type, String id)

  void removeInterface(String type, String interface)

  void provides(String interface, String function)

void forgetProvides(String interface, String function)

void isImplementedBy(String interface, String component)

void forgetIsImplementedBy(String interface, String component)

String[] getAllInterfaces()

String[] getInterfacesByType(String type)

String[] getInterfaceFunctions(String interface)

String[] getInterfaceComponents(String interface)

- **Component interface**

void addComponent(String type, String id)

void removeComponent(String type, String component)

void implementsInterface(String component, String interface)

void forgetImplementsInterface(String component, String interface)

void manages(String mediator, String domain)

void forgetManages(String mediator, String domain)

void generates(String mediator, String internalComponent)

void forgetGenerates(String mediator, String internalComponent)

void isGeneratedBy(String mediator, String internalComponent)

void forgetIsGeneratedBy(String mediator, String internalComponent)

String[] getAllComponents()

String[] getComponentsByType(String type)

String[] getComponentInterfaces(String component)

String[] getFrameworSKIMAediators(String framework)

String getMediatorDomain(String mediator)

- **Domain interface**

  void addDomain(String id, String name)

  void removeDomain(String domain)

  void isManagedBy(String domain, String mediator)

  void forgetIsManagedBy(String domain, String mediator)

  void hasUser(String domain, String domainUser)

  void forgetHasUser(String domain, String domainUser)

  String[] getAllDomains()

  String getDomainName(String domain)

  String[] getDomainMediators(String domain)

  String[] getDomainUsers(String domain)

- **Schema interface**

  void addSchema(String type, String id, String author, String version, String url)

  void removeSchema(String type, String schema)

  String getSchemaAuthor(String schema)

String getSchemaVersion(String schema)

String[] getAllSchemas()

String[] getSchemasByType(String type)

A.3 PI system

The PI system is composed by several modules that perform specific tasks in interaction with an inference engine. These modules provide a set of functions that can be used by users according to their specific permissions. Functions are based on the domain schema presented in section 5.3.

A.3.1 PI system modules

Figure A.21 shows the PI system modules.

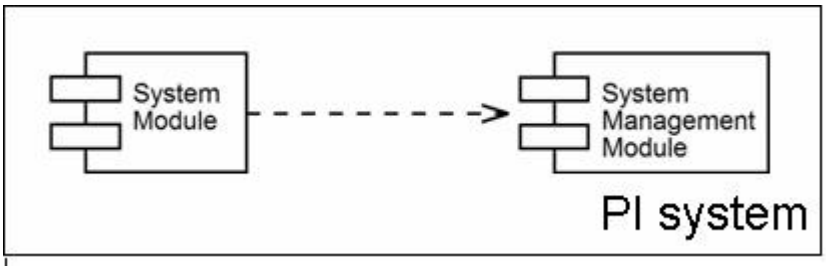


Figure A.21 PI system modules.

System module

The system module provides the system main interface. It allows system users to login and to logout, and allows the system to evaluate learners and to display resources.

- **System main interface**

  void enterSystem(String login, String password, String type)

void exitSystem()

void displayResource(String resource)

void evaluateLearner(Vector questions)

System management module

The system management module provides several interfaces that allow managing metadata of the system and its components. These interfaces are used to query, add and remove actors (i.e. learners, facilitators, administrators and authors), courses, sections, topics, exercises, questions and resources.

- **Actor interface**

  void addActor(String type, String id, String name)

  void removeActor(String type, String actor)

  String getActor(String uri)

  String getActorName(String user)

- **Learner interface**

  void isLearnerOf(String learner, String course)

  void forgetIsLearnerOf(String learner, String course)

  void learns(String learner, String section)

  void forgetLearns(String learner, String section)

  void addGrade(String learner, String section, String grade)

  String[] getLearnerGrades(String learner)

String getLearnerSection(String learner)

String[] getLearnerCourses(String learner)

- **Facilitator interface**

  void isFacilitatorOf(String facilitator, String course)

  void forgetIsFacilitatorOf(String facilitator, String course)

  String[] getFacilitatorCourses(String facilitator)

- **Administrator interface**

  void isAdministratorOf(String administrator, String course)

  void forgetIsAdministratorOf(String administrator, String course)

  String[] getAdministratorCourses(String administrator)

- **Author interface**

  void isAuthorOf(String author, String resource)

  void forgetIsAuthorOf(String author, String resource)

  String[] getAuthorResources(String author)

- **Course interface**

  void addCourse(String id, String name, String[] keywords)

  void removeCourse(String course)

  void hasLearner(String course, String learner)

  void forgethasLearner(String course, String learner)

void hasFacilitator(String course, String facilitator)

void forgetHasFacilitator(String course, String facilitator)

void hasAdministrator(String course, String administrator)

void forgetHasAdministrator(String course, String administrator)

void isComposedBy(String course, String section)

void forgetIsComposedBy(String course, String section)

String getCourse(String uri)

String getCourseName(String course)

String[] getCourseAdministrators(String course)

String[] getCourseFacilitators(String course)

String[] getCourseLearners(String course)

String[] getCourseSections(String course)

- **Section interface**

  void addSection(String id, String name, String order)

  void removeSection(String section)

  void composes(String section, String course)

  void forgetComposes(String section, String course)

  void isLearnedBy(String section, String learner)

  void forgetIsLearnedBy(String section, String learner)

void hasExercise(String section, String exercise)

void forgetHasExercise(String section, String exercise)

void hasTopic(String section, String topic)

void forgetHasTopic(String section, String topic)

void hasResource(String section, String resource)

void forgetHasResource(String section, String resource)

String getSection(String uri)

String getSectionName(String section)

String getSectionOrder(String section)

String getSectionCourse(String section)

String[] getSectionLearners(String section)

String getSectionExercise(String section)

String getSectionTopic(String section)

String[] getSectionResources(String section)

- **Topic interface**

void addTopic(String id, String name)

void removeTopic(String topic)

void isTopicOf(String topic, String sectionOrCourse)

void forgetIsTopicOf(String topic, String sectionOrCourse)

String getTopic(String uri)

String getTopicName(String topic)

String[] getTopicKeywords(String topic)

String[] getTopicSections(String topic)

- **Exercise interface**

  void addExercise(String id, String percentage)

  void removeExercise(String exercise)

  void isExerciseOf(String exercise, String section)

  void forgetIsExerciseOf(String exercise, String section)

  void hasQuestion(String exercise, String question)

  void forgetHasQuestion(String exercise, String question)

  String getExercise(String uri)

  String getExercisePercentage(String exercise)

  String getExerciseSection(String exercise)

  String[] getExerciseQuestions(String exercise)

- **Question interface**

  void addQuestion(String id, String question, String[] options, String answer)

  void removeQuestion(String question)

  void isQuestionOf(String question, String exercise)

void forgetIsQuestionOf(String question, String exercise)

String getQuestion(String uri)

String getQuestionStatement(String question)

String[] getQuestionOptions(String question)

String getQuestionAnswer(String question)

String getQuestionExercise(String question)

- **Resource interface**

  void addResource(String type, String id, String name, String format, String quality, String size, String url, String language)

  void removeResource(String type, String resource)

  void isResourceOf(String resource, String section)

  void forgetIsResourceOf(String resource, String section)

  void isOpenedWith(String resource, String application)

  void forgetIsOpenedWith(String resource, String application)

  void hasAuthor(String resource, String author)

  void forgetHasAuthor(String resource, String author)

  String getResource(String uri)

  String getResourceName(String resource)

  String getResourceFormat(String resource)

String getResourceQuality(String resource)

String getResourceSize(String resource)

String getResourceUrl(String resource)

String getResourceLanguage(String resource)

String[] getResourceSections(String resource)

String[] getResourceApplications(String resource)

String[] getResourceAuthors(String resource)

Appendix B Ontology graphical representation

In order to make the representation of our ontologies easier for the lector to interpret, we propose a mapping from UML to an easy graphical representation (see figure B.1).

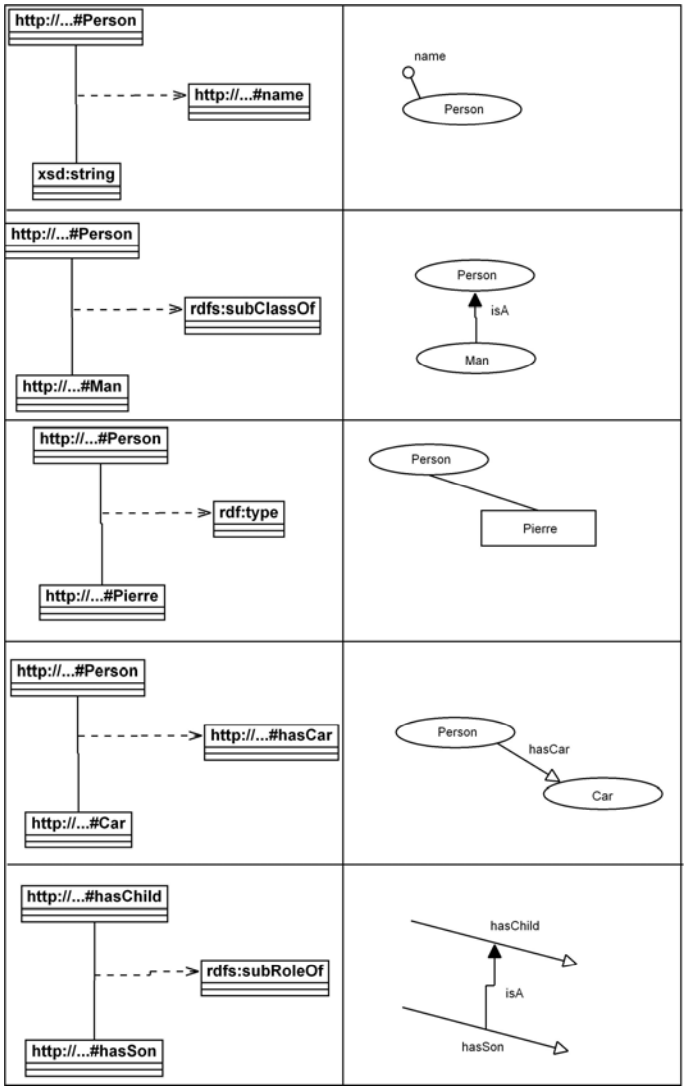


Figure B.1 Mapping from UML to a graphical ontology representation.

Along this work schemas will be represented with this graphical ontology representation.

Appendix C Installation

C.1 Required software

SKIMA must be used on a Windows platform that supports Internet Explorer for the PI system uses it to display resources. Besides it is necessary to install the following software:

- The Java 2 SDK 1.4.2 standard edition. (http://java.sun.com/j2se/1.4.2/download.html).
- The Renamed ABox and Concept Expression Reasoner (Racer). (http://www.cs.concordia.ca/~haarslev/racer/download.html).

C.2 Download

SKIMA is available in http://www.udlap.mx/~is113111/TESIS/Prototype.zip. This file includes the code and schemas for MBF, SKIMA and the PI system. Prototype.zip includes five executable files: racer.bat, rmiregistry.bat, MBFServer.jar, MBFClient.jar and Prinsys.jar.

C.3 Execution

SKIMA is executed as follows:

1. Start the inference engine.
2. Start MBF.
3. Load schemas.
4. Build a SKIMA.
5. Start the PI system.

To start the inference engine execute racer.bat. In order to start MBF execute rmiregistry.bat and MBFServer.jar. Then load the schemas (domainschema.daml, localschema.daml, sourcescharacteristics.daml, schemaspecification.daml) and build a SKIMA executing MBFClient.jar. Finally to start the PI system execute Prinsys.jar.