UDC:
Original scientific paper

A DISTRIBUTED SEMANTIC MEDIATION ARCHITECTURE[1]

Ismael Navas-Delgado, José F. Aldana-Montes
Dpto. Lenguajes y Ciencias de la Computación, E.T.S. de Ingenería Informática
Universidad de Málaga, SPAIN
{ismael, ifam}@lcc.uma.es

Abstract: *Traditional mediators are usually developed as monolithic systems which envelope the data source's semantics as well as its location. Furthermore, their architecture based on wrappers involves a high degree of coupling among the mediators' components. This coupling does not allow sharing services with other organizations or the dynamic integration of new data sources. Therefore, wrappers must be re-designed and manually added for each mediation system. We propose an architecture based on P2P philosophy for semantic mediation in which the sources' query capabilities are published as web services. These services can be registered in one or more resource directories (Semantic Directories), which are the core of this architecture because they provide the needed flexibility and scalability for dynamic integration.*

Keywords: *integrating heterogeneous data sources, dynamic mediation, semantics, ontology, Web services.*

1. INTRODUCTION

Over the past years the Web has become a great information repository that is manually accessed in the majority of cases. The amount of information available and the complexity of a reasonable treatment to take advantage of all this information have led to a lot of research concerning database integration. The main goal of these systems is to allow users to make complex queries over heterogeneous databases, as if they were a single one, using an integration schema. Mediators offer user interfaces for querying the system, based on the integration schema. These mediators transform user queries into a set of sub-queries that other software components (the wrappers), which encapsulate data sources' capabilities, will send to data sources. Usually, sub-query results are unstructured documents that are translated into structured documents by wrappers following a standard format (XML, for example). XML technology makes it possible both to structure the information and to explicit the schema by means of an XML Schema document. However

[1]A preliminary version of this paper appeared in *Proceedings of the 6th International Conference on Enterprise Information Systems*, ICEIS2004, Porto, Portugal, April 4 -17, 2004.

this information is insufficient for agents searching the Web, due to the fact that they cannot interpret these XML documents because they do not know their semantics.

These mediation systems have evolved through several improvements of traditional mediation architecture (of systems such as Manifold [1] and TSIMMIS [2]). Nonetheless, there are still several unsolved problems in heterogeneous data integration, including the following:

1. Design problems:
   a. mediators and wrappers are strongly coupled;
   b. wrappers are manually designed for each new mediator by software developers; in other words, there is no reusability;
   c. there is no dynamic integration, so mediators must be changed for each new data source;
   d. it is not possible to use loose integration: that is, integration based on semi-automatic mappings between mediator schema and sources' schemas.
   e. software agents can not find wrappers, as they are "hidden" behind mediators.
2. Semantic problems:
   a. wrapper query capabilities and semantics are not published;
   b. traditional mediators do not allow expressive queries based on semantics.

Mediation research has dealt with data integration using a monolithic approach to the problem. But this does not provide scalable and reusable solutions. Thus, our proposal tries to break out of traditional mediation architecture. That is, we hope that by uncoupling all mediation components we can obtain a more scaleable and reusable application. Our approach is a distributed architecture for semantic mediation (see Section 3). We define semantic mediation architecture as a mediation architecture that is capable of taking advantage of data source semantics. Thus, the proposed architecture includes directories in which an ontology and semantic information of resources are published. We also propose to improve the wrapper implementation process by publishing them as web services, and making their semantics accessible. This evolution from traditional wrappers to data services is motivated by several factors:

- Data services can be reused by other mediators or any other data accessing application.
- The semantics of data services is published on the web so that the services are readily available for other applications.
- Wrapper query capabilities can be enveloped into one or more services.

This paper is organised as follows. Section 2 describes several related works. Section 3 presents a novel architecture for semantic mediation, which makes use of web services to allow dynamic integration. In section 4, we briefly describe use in a biological case study to validate our proposal. Finally, the paper concludes with discussion and future works sections.

2. RELATED WORKS

The wrapper-mediator approach provides an interface to a group of (semi) structured data sources, combining their local schemas into a global one and integrating the information of local sources. Therefore, the views of the data that mediators offer are coherent, performing semantic reconciliation of the common data model representations carried out by the wrappers. Some good examples of wrapper-mediator systems are TSIMMIS [2] and Manifold [1]. Several improvements have been made of traditional mediators. One of the most important is the use of standard representation languages, like XML. Thus, the MIX [3] (the successor to the TSIMMIS project) and MOCHA [4] projects are XML-based.

The next level of abstraction on Web integration corresponds to ontology-based systems. Their main advantage with respect to mediators is their capacity to manage schemas that are unknown a priori. This is achieved by means of a mechanism that allows contents and query capabilities of the data source to be described declaratively. OBSERVER [5] uses different ontologies to represent information data sources. Users explicitly select the ontology that will be used for query evaluation. The existence of mappings among ontologies allows the user to change the ontology initially selected.

Model-Based Mediation [6] is a paradigm for data integration in which data sources can be integrated, taking advantage of an auxiliary expert knowledge. This knowledge includes information about the domain and it is the glue that joins data source schemas together. The expert knowledge is captured in a data structure called Knowledge Map. In Model-Based Mediation the mediation architecture is extended, carrying data sources from the data level without semantics to the conceptual model level. This architecture introduces semantics into data sources and mediators, but they are not published and accessible to agents or applications. Mediators are monolithic systems and they are strongly coupled to wrappers, limiting dynamic integration and interoperability.

3. A SEMANTIC MEDIATION ARCHITECTURE

The main problem in traditional mediation systems is the coupling between mediators and wrappers. For this reason our major goal is to propose a distributed architecture, which makes wrappers independent entities and eliminates their ties with the mediator; thus their reusability increases in different applications by means of semantics. Since we have also defined the term "semantic mediation architecture" as "mediation architecture that is capable of taking advantage of semantics", we propose a distributed semantic mediation architecture.

Due to the need for distribution we make use of the P2P philosophy. P2P is based on contents sharing, by means of peer connections, without a server. Thus, the availability rate is higher than in a client/server architecture. There are four P2P architectures:

- Pure P2P: this model implements an architecture in which each node can act as a server, a client or a router. Thus, each node can search nodes making use of Multicast or Broadcast.
- P2P with node queries: this model implements an architecture in which a node can query a server to inquire about active nodes. Then, the node can connect with an active node to share resources. Each node must report to the server in order to maintain service integrity. It is important to note that in this case the server only offers a service to find active nodes, and thus differs from a server in a client/server architecture.

- P2P with node and resource queries: this model implements a server which stores active nodes and whose contents are shared.
- P2P with node and resource queries, and content sources: this model implements a Server which has two tasks. The first one is to store active nodes, the contents of which are shared. The second is to store shared contents of connected nodes.

We emulate P2P hybrid systems (P2P with node and resource queries) which implement a directory with location information of available resources and information about which contents are shared. In these systems the applications access the resources directly by means of peer to peer connections that the directory has provided. Therefore, the flow of information is greatly reduced w.r.t. the one generated in traditional client-server architectures.

Our proposal for semantic mediation arises from two main considerations regarding the basic architecture of mediation:

1. on the one hand the isolation of wrappers, which are encapsulated as web services (Data Services for us); and
2. on the other, the added directory (Semantic Directory) with information about these Data Services (See Figure 1). Although the basic configuration has been developed with only one directory, nothing prevents us from having distributed configurations and/or from talking about more complex architectures with several (maybe replicated) semantic directories (for reasons of failure tolerance, availability or scalability). This architecture allows wrappers to contribute data, information schemas and query capabilities in a decentralized and easily extensible manner.

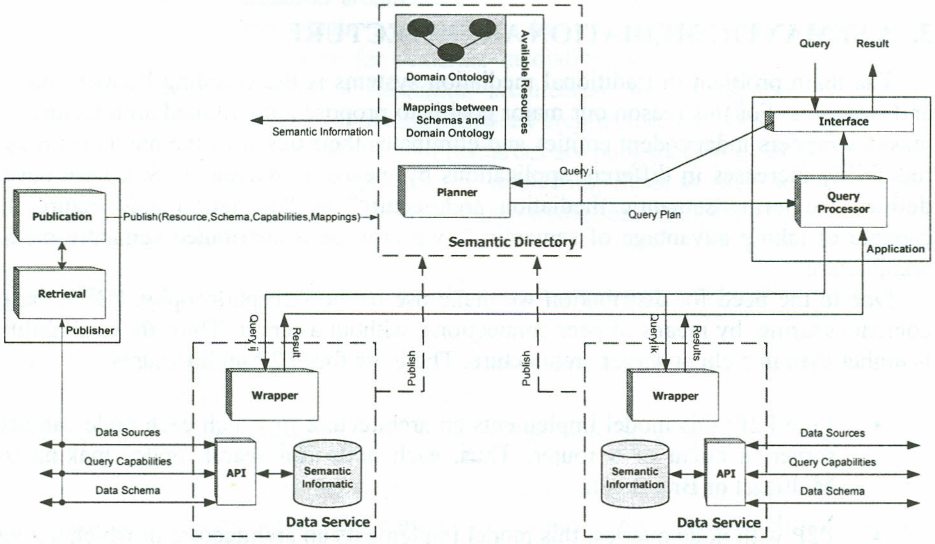


Figure 1: A Distributed Semantic Mediation Architecture

A data service needs to be registered (by means of a publisher agent) in one or more semantic directories in order to be used by a mediator or by any other software agent. In other words, data services, like P2P hybrid systems, must know the location of semantic directories that are in the same application domain. Finally, public interfaces of data services and semantic directories will allow applications that share their communication protocol to take advantage of knowledge about available directory resources. Below we present the components of the proposed architecture as well as their functionality.

3.1. SEMANTIC DATA SERVICES

Semantic directories offer essential services for obtaining stored information and semantics through several web methods, such as getting and/or navigating over the domain ontology, the resources' address, and the mappings. Besides, each directory can offer a set of additional services. Figure 1 shows a service for query processing which returns query plans (Planner). These plans can be used and processed by a specific type of application: the mediators. These applications need to have a minimal query processor, which must understand the plans, send sub-queries to data resources and compose the partial results. Note that the planners return sub-queries in a language that data services can evaluate. However, semantic directories could be increased with other services in order to support other types of applications/agents.

On the other hand, data services provide the minimal elements for solving queries. We will define this type of service and then describe how it can access a resource that has been included in a query plan.

Definition 1: A wrapper is a function W: (Q , R) $\rightarrow$ D. Given a query Q and a resource R, a wrapper returns a (semi)structured document D with the results of evaluating query Q in resource R.

Definition 2: A data service DS is a web service that offers several web methods for obtaining semantics and other information to query a wrapper (it is not the wrapper itself).

The goal of data services is to allow applications to access wrapper functionalities by means of web services. In this way, we have designed an extensible and adaptive architecture in which we can define a data service as "a service which offers wrapper query capabilities using web protocols". The publication of these online web services using UDDI (Universal Description Discovery Integration) could allow other applications to dynamically discover them by means of an easy interface. However, our data services have been devised for publication in a specialized type of directory: the semantic directory.

Figure 2 shows the internal architecture of a data service that accesses data sources using a wrapper, which solves a query and returns an XML document, and exports the wrapper's query capabilities and its semantics as a web service. The web service's semantics includes information about query capabilities, data schemas and data provenance. The latter is necessary, for example, in the context of bioinformatics, where it is important to know the source of information that is being used (due to different data quality and reliability). Data services need to describe their source capabilities for translating user queries into sub-queries that are understood by a specific source interface.

In a first approach we used the type of query capabilities described in [7], and we defined the annotations for each attribute or element. These query capabilities were expressive enough to represent the capabilities of web resources. However, we wanted to provide a solution for heterogeneous data sources, because the biological resources could be relational databases, web resources or plain text. In our present approach we use p-Datalog [8] to describe query capabilities. It is a Datalog variant which copes with the need of a more powerful description language. The p-Datalog source description language allows defining capabilities for conjunctive queries. A result of the cited paper [8] is that p-Datalog can not describe capabilities of certain powerful sources.

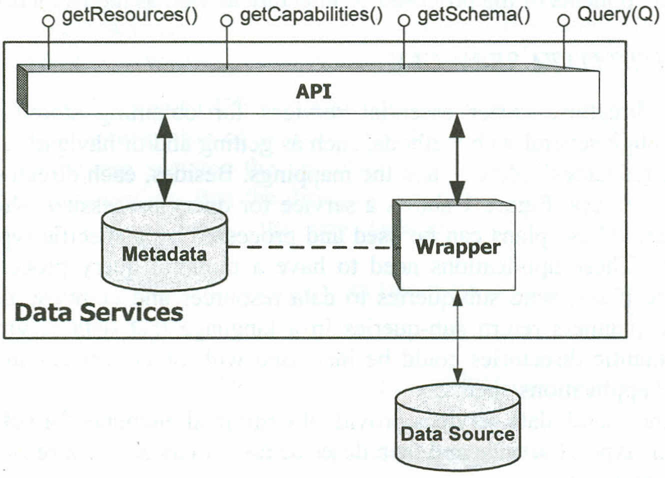


Figure 2: Data Service Architecture

This type of service not only encapsulates a wrapper's query service, but also provides access to the metadata they store. For this purpose the getSchema(), getCapabilities() and getSourcesInformation() methods are published, and they return the data's schema, the query capabilities, and information related to data sources used by the data service, respectively.

3.2. SEMANTIC (RESOURCE) DIRECTORIES

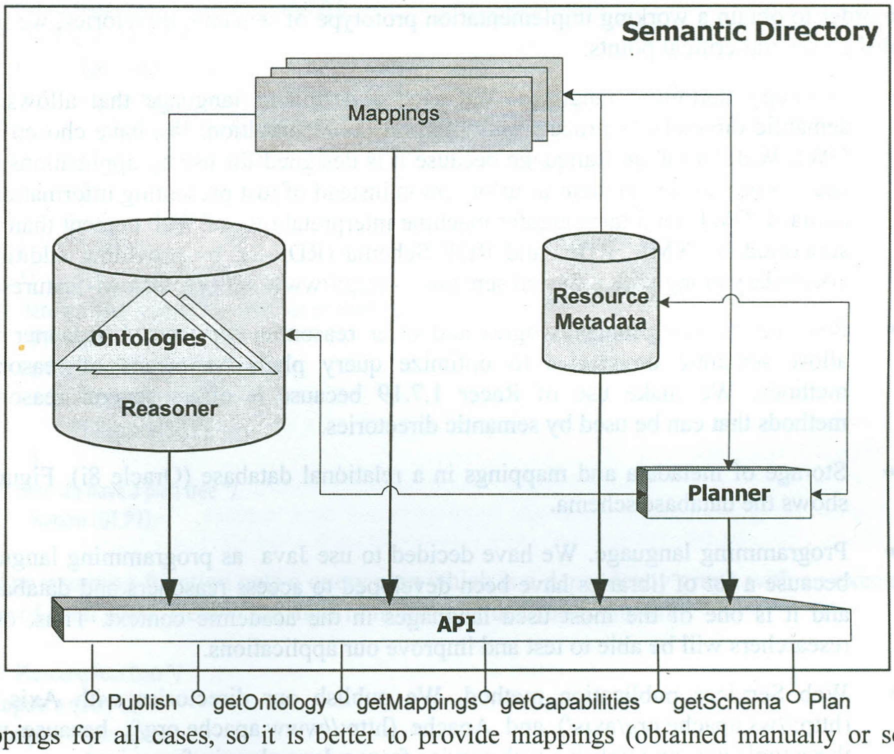
Semantic directories are the core of this architecture, because they provide essential services for application domain users. Below we will define several necessary terms for the definition of semantic directory and describe this type of resource directory.

Definition 3: A query plan QP is a tuple <Qs , Rs , Cp> where Qs is a vector of queries; Rs is a vector that contains the resource address where these queries must be evaluated and Cp is a composition plan that includes a method to compose the result of evaluating each query Q in the correct resource R.

Definition 4: A planner P is a function P: Q → QP. Q is a query and QP is a query plan.

Definition 5: A Semantic Directory SD is a server that stores one or more domain ontologies DOs, mappings between data resource schemas (M) and DOs, and information about available resources. Besides, it can offer a set of services, such as a planner P. That is, we can define a semantic directory as "a server that offers information about available web resources (Data Services), one or more domain ontologies, mappings between resource schemas and these ontologies, and provides services (for example, a query planner) to application domain users".

A semantic directory stores ontologies, which must be generic for the application domain (Domain Ontologies). These ontologies describe the core knowledge that is shared by a set of applications or a user community. For our purpose, the Domain Ontologies can be seen as an abstraction of the knowledge of the resource's schemas. Thus each schema could be considered as a refinement of a domain ontology. The main functionality of the directory is to provide access to semantic information stored in it. However, it can offer several value added services: for example, to return an execution plan. This type of service increases its functionality whenever the similarity between both models (a Domain Ontology and the resource's) also increases. Information about data services will be added to a semantic directory when services register in it by means of a publisher agent, for example. Thus, the data service owner (or a publisher agent) must use the publish method (offered by the directory, see Figure 3) in order to publish his/her service. This method saves information about the service, and if mappings are not provided it calculates mappings between one of the directory's domain ontologies and the service's schema (by means of automatic matching). However, automatic matching does not provide good



mappings for all cases, so it is better to provide mappings (obtained manually or semi-automatically) to the publish method.

Figure 3: Internal Architecture of a Semantic Directory

Semantic directories periodically ask registered data services for updated information.

This allows keeping schemas, query capabilities and availability updated at all times and consistent with data services. If there are changes, then the directory recalculates mappings taking advantage of previous mappings. Besides, the owner of a service can disconnect it, making it unavailable to be included in query plans.

The storage of mappings between service schemas and domain ontologies is very important, due to their use in query planning, or to find sources related with a concept of a domain ontology. In an elementary case, applications send queries (Q_i) in terms of one of the ontologies to the semantic directory, which returns a query plan (QP) for this query.

Each plan includes links to services, a set of sub-queries for each service (Q_{i1} ... Q_{in}), a query tree and a precedence tree that can be used to apply a composition strategy. The plans are based on mappings stored in the semantic directory. Note that this architecture does not implement a query processor, so the integration applications can evaluate the QPs, releasing the semantic directory of this task. Note that a mediator is just one of the possible applications which have a query processor to perform the query plan.

Besides, information stored in directories could be used by other applications or agents that search for information about ontologies, data services whose schema accomplish a condition, etc. In a special context agents can implement their own query planner. For this reason, the directory's API provides several methods to retrieve information stored in it, namely getOntology(O), getResources(), getSchema(R), getMappings() and getMappings(R).

4. PROTOTYPE IMPLEMENTATION DETAILS

In order to obtain a working implementation prototype of semantic directories, we have established several critical points:

- Ontology definition language. We need a definition language that allows the semantic directory to process and interpret the information. We have chosen the OWL Web Ontology Language because it is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S), by providing additional vocabulary along with a formal semantics (http://www.w3.org/TR/owl-features/).
- Reasoner to store the ontologies and offer reasoning services. A reasoner will allow semantic directories to optimize query plans by means of reasoning methods. We make use of Racer 1.7.19 because it offers several reasoning methods that can be used by semantic directories.
- Storage of metadata and mappings in a relational database (Oracle 8i). Figure 4 shows the database schema.
- Programming language. We have decided to use Java as programming language because a lot of libraries have been developed to access reasoners and databases, and it is one of the most used languages in the academic context. Thus, other researchers will be able to test and improve our applications.
- Web Services publication method. We publish our directories with Axis 1.0 (http://ws.apache.org/axis/) and Apache (http://www.apache.org/), because with these tools we can publish a web service from a Java class in few minutes.

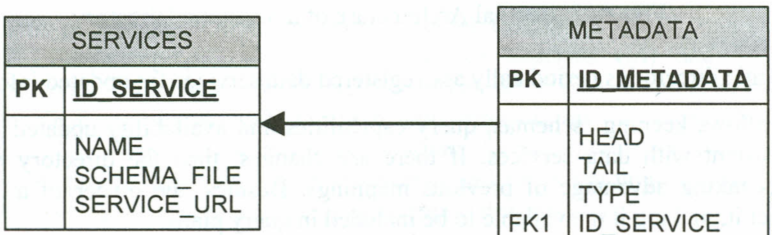


Figure 4: Semantic Directory Database Schema

It is also necessary to implement a planner, so we have designed the planning algorithm shown below:

```
Input: a conjunctive query and the reference of a domain ontology of the semantic directory.
Output: query result in terms of a domain ontology.
Pseudo-code:
    Connect with the Oracle database;
    Q = Read the input query;
    QT = new_query_tree();
    PT = new_plan_tree();
  QT = Semantic_Optimization(Q);
  /*
  Optimization can include, among other elements, search sub-classes of each sub-goal and reformulate the
  query. For example:
  For each sub-goal (SG) of the query DO {
      If SG is a class THEN {
          Get all descendants;
          For each descendant (D) DO {
              New_Q = Q.Replace(SG,D);
                  Add(New_Q,QT);
          }
      }
  }
  */
/* Here we have a query tree with one or more queries */
    PT = QT;
  For each sub-Query (SQ) in QT DO {
      P = Plan(SQ);
          PT.Replace(SQ,P);
    }
/* Here we have a plan tree */
    Return (QT,PT);
```

The planning function gets a query plan which is a dependency graph with an execution order of each node. Below the algorithm for this function is shown:

```
/* Planning function */
Input: a query (Q)
Output: a query plan
Pseudo-code:
    Create two buckets;
    For each subgoal (SG) in Q DO {
        If Arity(SG) == 1 DO {
            Bucket1.Insert(SG);
        }
        Else {
            Bucket2.Insert(SG);
        }
    }
  For each element (E) in Bucket2 DO {
        Pair = Bucket1.Find_Pair(E);
    /* Two elements are a pair if they have a common argument */
```

```
            Pair_List.Add(Pair);
        }
    For each pair (P) in Pair_List DO {
        Find correspondences in table Metadata (Type = "Mapping");
          Replace each pair for head value of the table;
    }
    /* Here we have a set of elements in terms of the data sources */
    /* Create the dependency graph */
    For each data source for which we have a term in the Quero DO {
        Create a node with:
            Un-Bounded parameters;
            Relations with other nodes (arcs between nodes);
    }
    Establish an execution order taking advantage of a heuristic;
    Return the dependency graph, with the execution order;
```

At this point the prototype can be used to store resources of any domain. When we define the domain ontologies of the semantic directory, it is able to receive information about resources of this domain.

5. CASE STUDY

The accumulated biological knowledge needed to produce a more complete view of any biological process (e.g. sequences, structures, gene-expression data, pathways) is disseminated around the world in the form of biological sequences and structure databases, frequently as flat files, as well as image and scheme-based libraries, web-based information, particular and specific query systems, etc. Although it is a common place statement that the volume of data in biology is growing at exponential rates, nonetheless, the key characteristic of biological data is not so much its volume as its diversity, heterogeneity and dispersion [9].

BioBroker [10] is an XML mediator which supports biologists working in the field of proteomics and genomics. It integrates heterogeneous data sources: nucleotide (EMBL) and protein (SWISS-PROT) sequence information [11], the worldwide repository of three-dimensional structures of biological macromolecules (PDB) [12] and the MICADO relational database devoted to microbial genomes [13]. These databases have been selected on the basis of their difference in content, format, access mechanism, and geographical location. We are currently developing a mediator for integrating biological resources-, which will interact with a semantic directory and several data services. This will be the natural evolution of the previous XML mediator (BioBroker). Below we present a sample use of the proposed architecture for integrating gene expression databases.

The semantic directory for BioBroker resources includes an ontology which covers the concepts genomic and proteomic (see Figure 5). This ontology, which treats knowledge about organisms, genes and proteins, is related to the data sources through mappings between the data schemas and the domain ontology.

We are developing the data services for each BioBroker wrapper. Figure 6 shows (from top to bottom) the schemas of three data services which access Micado, Swiss-Prot, Factor database of Transfac and Gene database of Transfac. In each one we must define its semantics (query capabilities, data provenance, etc.). Figure 7 shows some of the mappings between these schemas and the domain ontology. Finally, query capabilities for the MICADO data service are shown in Figure 8.

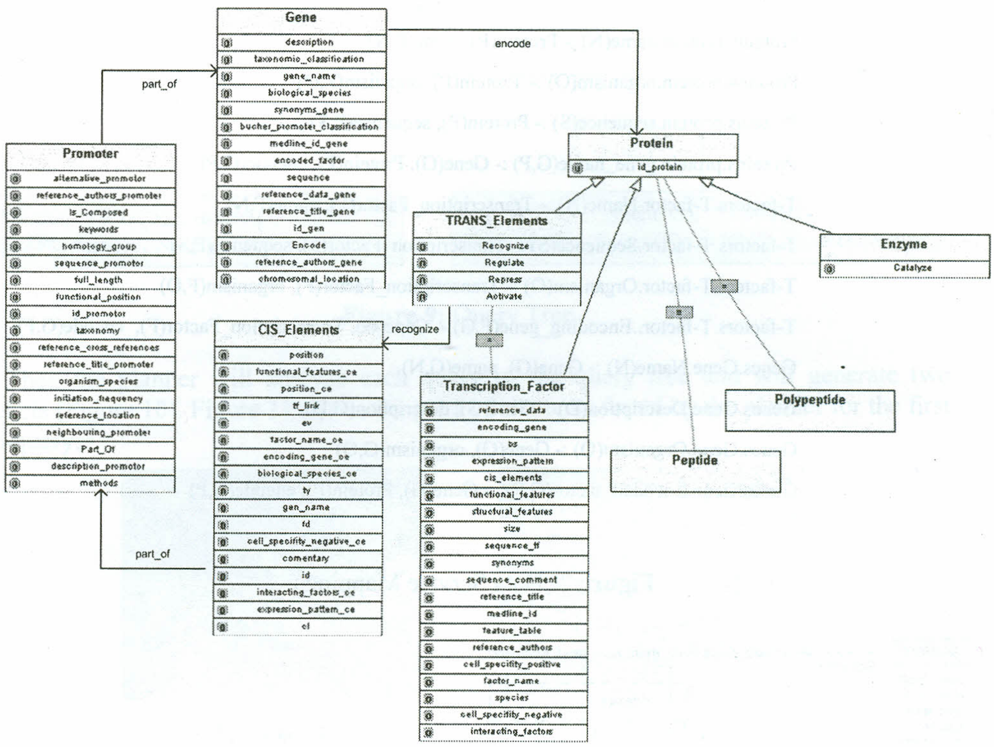


Figure 5: Domain Ontology

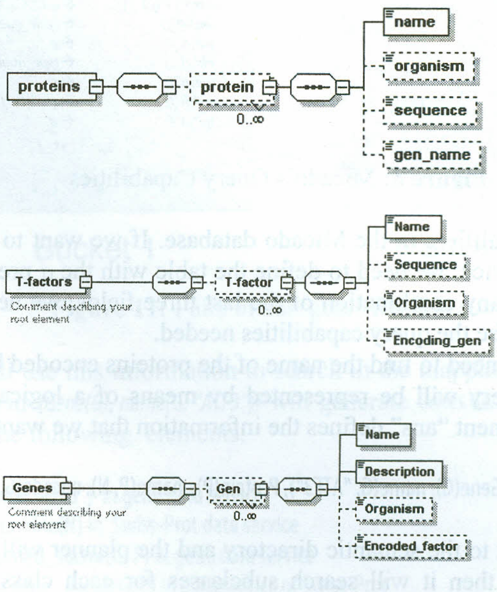


Figure 6: Data Service Schemas

Proteins.protein.name(N) :- Protein(P), name(P,N)

Proteins.protein.organism(O) :- 'Protein(P), organism(P,O)

Proteins.protein.sequence(S) :- Protein(P), sequence(P,S)

Proteins.protein.gene_name(G,P) :- Gene(G), Protein(P), encode(G,P)

T-factors.T-factor.Name(N) :- Transcription_Factor(F), name(F,N)

T-factors.T-factor.Sequence(S) :- Transcription_Factor(F), sequence(F,S)

T-factors.T-factor.Organism(O) :- Transcription_Factor(F), organism(F,O)

T-factors.T-factor..Encoding_gene(F,G):- Gene(G), Transcription_Factor(F), encode(G,F)

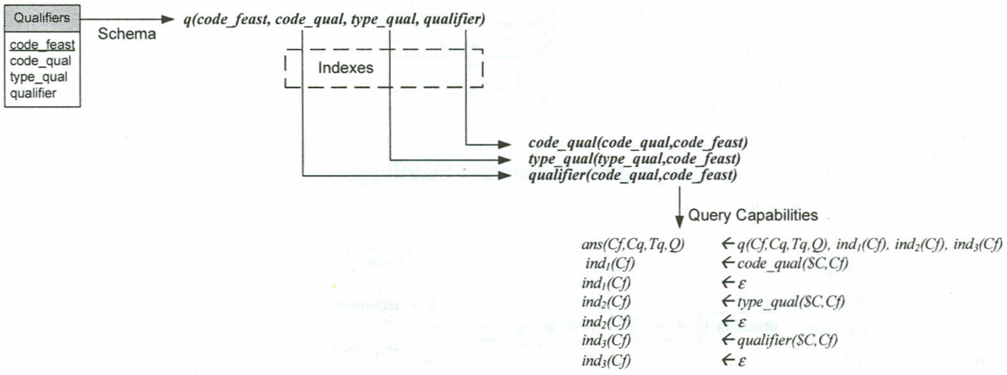
Genes.Gene.Name(N) :- Gene(G), name(G,N)

Genes.Gene.Description(D) :- Gene(G), description(G,D)

Genes.Gene.Organism(O) :- Gene(G), organism(G,O)

Genes.Gene.Encoded_factor(G,P)' :- Gene(G), Protein(P), encode(G,P)

Figure 7: Data Service Mappings


Qualifiers
code_feast
code_qual
type_qual
qualifier
Schema
q(code_feast, code_qual, type_qual, qualifier)
Indexes
code_qual(code_qual,code_feast)
type_qual(type_qual,code_feast)
qualifier(code_qual,code_feast)
Query Capabilities

$ans(Cf,Cq,Tq,Q)$	$\leftarrow q(Cf,Cq,Tq,Q),\ ind_1(Cf),\ ind_2(Cf),\ ind_3(Cf)$
$ind_1(Cf)$	$\leftarrow code\_qual(\$C,Cf)$
$ind_1(Cf)$	$\leftarrow \varepsilon$
$ind_2(Cf)$	$\leftarrow type\_qual(\$C,Cf)$
$ind_2(Cf)$	$\leftarrow \varepsilon$
$ind_3(Cf)$	$\leftarrow qualifier(\$C,Cf)$
$ind_3(Cf)$	$\leftarrow \varepsilon$

Figure 8: Micado's Query Capabilities

We have a table of qualifiers in the Micado database. If we want to provide access to query this using a data service, we need to define the table with the *q* predicate. In order to allow users to query with any combination of the last three fields, we need to define three indexes. Finally, we describe the query capabilities needed.

Let us suppose that we need to find the name of the proteins encoded by the gene whose name is "ATF·". This query will be represented by means of a logical and conjunctive formula (Note that the element "ans" defines the information that we want to obtain):

ans(N) :- Gene(G), name(G, "ATF3"), Protein(P), name(P, N), encode(G, P);

This query will be sent to the semantic directory and the planner will analyze each sub-goal to find classes, and then it will search subclasses for each class. In this case, the Protein class has a subclass Transcription_Factor, so the query tree that the planner obtains is shown in Figure 9.

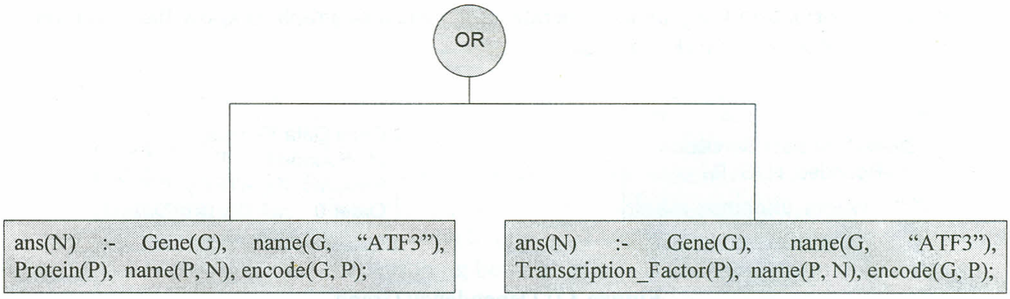

OR
ans(N) :- Gene(G), name(G, "ATF3"), Protein(P), name(P, N), encode(G, P);
ans(N) :- Gene(G), name(G, "ATF3"), Transcription_Factor(P), name(P, N), encode(G, P);

Figure 9: Query Tree

Then the planner will analyze each query of the query tree and will generate two buckets (Figure 10). Figure 11 shows the pairs of elements found by the planner for the first sub-query.

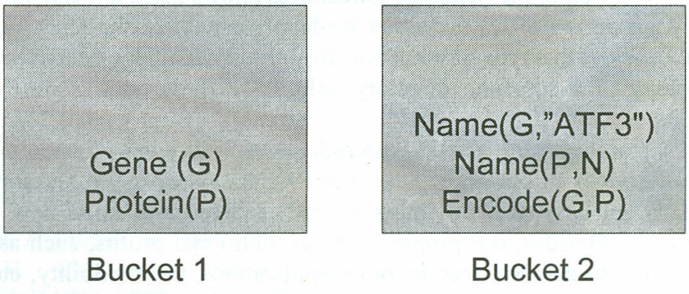


Figure 10: Buckets for first sub-query

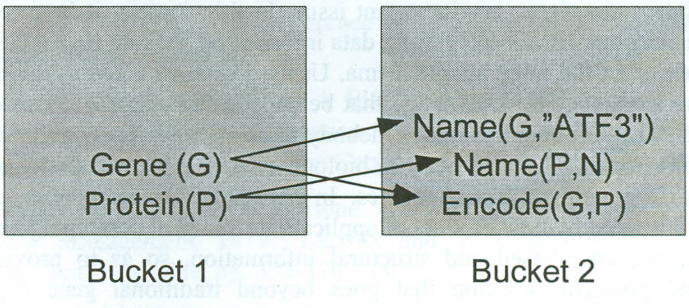


Figure 11: Pairs of sub-goals for first sub-query

The planner will use this information to search in the mappings stored in our directory. For example, the pair *(Gene(G), name(G, "ATF3"))* will generate *Genes.Gene.Name("ATF3")*. Thus, in this example we have the following elements:

Genes.Gene.Name("ATF3") in gene data service
Proteins.protein.name(N) in Swiss-Prot data service
Genes.Gene.Encoded_factor(G, P) in gene data service
Proteins.protein.gene_name(G, P) in data service of Swiss-Prot

With this information the planner generates a dependency graph to know the execution order by means of a heuristic (Figure 12).

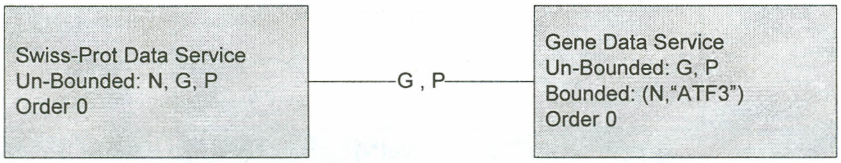


Figure 12: Dependency Graph

6. DISCUSSION

In this paper we describe an architecture for semantic mediation, based-on a P2P and Web Services, that presents advantages with respect to traditional mediation approaches. The semantics introduced in the Semantic Directories allows users to make more expressive queries, namely semantic queries that other mediators cannot solve. They greatly increase the query capabilities of this type of mediator. Besides, the mappings between schemas and a domain ontology of a semantic directory help solve these queries over the available resources.

The use of an architecture like P2P introduces a high level of uncoupling between wrappers (Data Services in our architecture) and mediators or any other application. The directories supply an easy way to integrate data sources and open new directions in dynamic integration. Besides, our proposal entails additional profits, such as the reuse of data services, access to data services by other applications, use flexibility, etc. It provides elements to obtain major interoperability among integration systems that cooperate in the same application domain or that belong to another domain with which they have certain relationships.

Dynamic integration is a very important issue. In this context, it is necessary to give users a simple environment for integrating data information without the modification of the mediator's code or of the integration schema. Using a domain ontology, users can design queries starting from specific knowledge that belongs to their field of research. However, the proposed architecture requires that somebody implement wrappers and publish them in the semantic directories. The case study in biology evidences the suitability of this kind of architecture for the field of bioinformatics. In particular, the usefulness of a mediator system is demonstrated by a diverse set of applications aimed at combining expression data with genomic, sequence-based and structural information, so as to provide a general, transparent and powerful solution that goes beyond traditional gene expression data clustering.

7. FUTURE WORKS

As future works, we propose several areas of mediator development, such as increasing the automation level in wrapper creation and adding to this process data service generation, which will reduce even more their cost of development. Another interesting area is to study the possibility of giving more semantics to these data services, taking into account service quality, relations with other domain ontologies, etc. Using this additional information, we could generate alternative query execution plans, allowing applications to choose the most suitable one or generating them based on certain features (using local resources for the application location). Besides, the scalability of this architecture will provide the possibility of integrating not only data services but also semantic directories, enabling the full semantic integration of resources and interoperability between applications. Thus, we will provide elements to achieve interoperability between semantic integration systems that cooperate in the same application domain or have certain relations.

We plan to study automatic mapping between schemas and ontologies, taking into account a previous experience [14]. It can be applied to establish correspondences between retrieved document schemas and directory ontologies in those new systems developed using our architecture. Finally, we are interested in establishing a semantic model to define the data service's query capabilities, which improves query planning by adding inferences about query capabilities to the reasoning between schemas and the domain ontology.

ACKNOWLEDGEMENTS

This work has been supported by the Spanish MCyT Grant (TIC2002-04586-C04-04)

REFERENCES

[1] A. Levy, A. Rajaraman, J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In Proc. VLDB, pages 251-262, Mumbai (Bombay), India, 1996.

[2] Y. Papakonstantinou, H. Garcia-Molina, J. Widom. Object Exchange Across Heterogeneous Information Sources. In Proc. ICDE, pages 251-260, 1995., Taipei, Taiwan, March.

[3] C. Baru, A. Gupta, A. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, V. Chu, V. XML-based Information Mediation with MIX. In Demostrations, ACM/SIGMOD, 1999., pages 597-599.

[4] Manuel Rodríguez-Martínez and Nick Roussopoulos. MOCHA: a self-extensible database middleware system for distributed data sources. Proceedings of the 2000 ACM SIGMOD international conference on Management of data, 2000, pages 213-224, Dallas, Texas, United States.

[5] E. Mena, V. Kashyap, A. Sheth, A Illarramendi. OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies. In conference on Cooperative Information Systems, 1996..

[6] B. Ludäscher, A. Gupta, M. E. Martone. A Model Based Mediator System for Scientific Data Management. In Z. Lacroix and T. Critchlow(eds.), Bioinformatics: Managing Scientific Data, pp. 335-370, 2003.

[7] R. Yerneni, C. Li, H. Garcia-Molina, J. D. Ullman, J.D. Computing capabilities of mediators. In SIGMOD, 1999., pages 443-454.

[8] V. Vassalos, Y. Papakonstantinou. Describing and Using Query Capabilities of Heterogeneous Sources. In *VLDB'97, 23rd International Conference on Very Large Data Bases,* 1997.

[9] F. Rechenmann. From Data to Knowledge, Bioinformatics, v.16.n5 pp 411, 2000.

[10] J. F. Aldana, M. Roldán, I. Navas, A. J. Pérez, O. Trelles. Integrating Biological Data Sources and Data Analysis Tools through Mediators. ACM Symposium on Applied Computing. Nicosia, Cyprus, 2004.

[11] A. Bairoch, R. Apweiler. The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000. In Nucleic Acids Res 28.

[12] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, P. E. Bourne. The Protein Data Bank. Nucleic Acids Research, Vol. 28, No. 1, 2000, 235-242.

[13] V. Biaudet, F. Samson, P. Bessieres. Micado a network-oriented database for microbial genomes. In Bioinformatics, Vol 13, 1997, 431-438. Oxford University Press.

[14] J. Madhavan, P. Bernstein, P. Domingos, A. Halevy. Representing and Reasoning about Mappings Between Domain Models. In proceedings of the AAAI Eighteenth National Conference on Artificial Intelligence, 2002.

Received: 14 June 2004
Accepted: 08 December 2004