Electronic Journal of SADIO

http://www.dc.uba.ar/sadio/ejs

vol. 2, no. 1, pp. 30-47 (1999)

Mediators Metadata Management Services : An Implementation Using GOA++ System

Thaís Saldunbides Brügger[1] Paulo de Figueiredo Pires[1] Marta Mattoso [1]

[1]Computer Science Department
COPPE/UFRJ - Federal University of Rio de Janeiro
P.O. Box 68511, Rio de Janeiro, RJ, Brasil, 21945-970
Fax: +55 21 2906626
e-mail: thais, pires, marta @cos.ufrj.br

Abstract

The main contribution of this work is the development of a Metadata Manager to interconnect heterogeneous and autonomous information sources in a flexible, expandable and transparent way. The interoperability at the semantic level is reached using an integration layer, structured in a hierarchical way, based on the concept of Mediators. Services of a Mediator Metadata Manager (MMM) are specified and implemented using functions based on the Outlines of GOA++. The MMM services e are available in the form of a GOA++ API and they can be accessed remotely via CORBA or through local API calls.

1 Introduction

During the last years, a great research effort has been directed to the integration of Heterogeneous and Distributed Database Systems (HDDS) (BUKHRES, 1996). HDDS provide transparent and simultaneous access to independent databases using one single data manipulation and definition language. The implementation of a HDDS requires a more complex technology than the centralized systems. To solve the conflicts generated by the different models and schema, the HDDS must provide additional functionalities to the ones found in current centralized systems.

To address these issues the heterogeneous architecture HIMPAR - ***H****eterogeneous* ***I****nteroperable* ***M****ediators and* ***P****arallel* ***Ar****chitecture* (PIRES, 1997) has been implemented. HIMPAR is based on the concepts of HDDS and adopts the technologies of client/server, object orientation and open systems as an infrastructure for the integration of several different data repositories. The semantic aspects of the interoperability is addressed by HIMPAR through an integration layer hierarchically structured based on the concept of Mediators (WIEDERHOLD, 1992) and Wrappers. At the communication level the interoperability is achieved through the CORBA (OMG, 1995, 1998) standard. The data model used by HIMPAR for the data integration is an extension of the ODMG-93 (CATTEL, 1997) model. This architecture is still under development, however a first prototype has been developed, providing the integration of three object oriented database systems, the O_2 (O_2, 1996), the GOA++ (MAURO, 1997) and the PARGOA system (MEYER, 1997). This prototype has been built using a CORBA compliant implementation from Visigenic Software, Inc. (VISIGENIC, 1996) called VisiBroker C++, and its current version is running on Sun workstations with Solaris 2.x. operational system.

Despite the many research projects in HDDS, the HIMPAR architecture is innovative because it uses Mediators with a strongly adherence to the object technology standards. The adoption of Mediators, with a canonical data model based on the ODMG-93 standard, to achieve interoperability at the semantic level is not an innovation, since other projects, such as the DISCO (TOMASIC, 1995) and the Garlic (CAREY *et al*., 1995), apply the same idea. However, the use of a Distributed Object Management (DOM) platform compatible with the CORBA standard to provide interoperability at the communication level has not yet been fully explored by the other projects based on the concept of Mediators. In addition, these works do not present specific services for the Mediator’s metadata management. This metadata management influences directly the flexibility and extensibility of a Mediator based architecture.

This work presents a Mediator Metadata Manager (MMM) that provides the necessary services for the definition of the data types stored at the repository of a mediator. This definition of mediators offers type construction through aggregation, generalization and specification. Besides, the MMM handles ad-hoc queries and manipulation of the defined types. The MMM implementation uses services of the GOA++ object server, particularly schema manager and query processing. MMM services are available through an API on top of GOA++ and can be issued remotely through CORBA or by local API calls. MMM was designed for the HIMPAR architecture, however its services may be used by other HDDSs based on mediators. The MMM enhanced the previous HIMPAR (PIRES, 1996) architecture adding flexibility on the mediators management.

The remaining of this work is organized as follows. Section 2 presents the HIMPAR architecture. The specification of the mediators manage is found in Section 3 as well as a case study. Implementation issues of MMM are discussed in Section 4. Finally, Section 5 concludes this work.

2 HIMPAR architecture

HIMPAR (PIRES, 1996; 1997a, 1998) - ***H****eterogeneous* ***I****nteroperable* ***M****ediators and* ***P****arallelism* ***Ar****chitecture* – is a project, based on the HDDS notion that is being developed at PESC/COPPE[1]. HIMPAR enables users to access distributed databases transparently and with no concern about local operational details, such as query languages or operational procedures. End users see a set of homogeneous objects that can be accessed through a standard interface. This architecture is based on components named **Mediators**. According to Wiederhold (1992), a Mediator is a software component, which explores the knowledge represented in a set or subset of data to generate information for applications residing in an upper layer. Each Mediator encapsulates the representation of multiple data sources and provides the functionality of uniform access to data. Thus, this component solves conflicts that commonly arise in an environment like this, such as those concerning knowledge representation (different schemas).

The user accesses system data through queries, written in a global language, and it is the Mediator that submits them to the local systems. The Mediator transforms the queries into sub-queries and sends them to the adequate local data repositories.

The sub-queries generated by the Mediators must be translated from the global language into the query language of each data repository. The **Wrapper** components are responsible for this functionality. These components map the sub-queries written in the global language into the local query language and return the reformatted responses to the appropriate Mediator. This component solves problems related to differences in the query expressiveness of each repository.

In the HIMPAR architecture, the interoperability of Mediators and Wrappers is accomplished through the CORBA standard. The access to local data repositories is achieved through an ORB, using a subset of commands from the OQL (Object Query Language). When a client application activates an OQL global query to access multiple bases, this query is decomposed into local sub-queries by the Mediator that sends them to the ORB. The ORB transfers the sub-queries to the corresponding Wrappers, which act as object servers. At the server node, the local sub-query is executed by the local routine and the response is returned through the ORB. The returned responses receive further treatment at the client, if necessary. The definition of Wrappers is based on a generic database interface. Once this interface is defined, other interfaces are specialized and implemented according to the particular functionality of each component system. Multiple implementations of the same interface are provided for component systems that support the same functionality.

Each Mediator is implemented by two CORBA objects: The Mediator Query Manager (MQM) and the Mediator Metadata Manager (MMM). The Wrapper layer is implemented by other two CORBA objects: the Wrapper and the Container. There is another object, named HIMPAR Service Manager (HSM), responsible for management services and system maintenance. All the components are connected through a local

[1] Programa de Engenharia de Sistemas e Computação (PESC) do Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa em Engenharia (COPPE).

network where the communication agent is the ORB. Figure 1 shows the architecture components in detail. The arrows are directed from the requesting object to the provider.

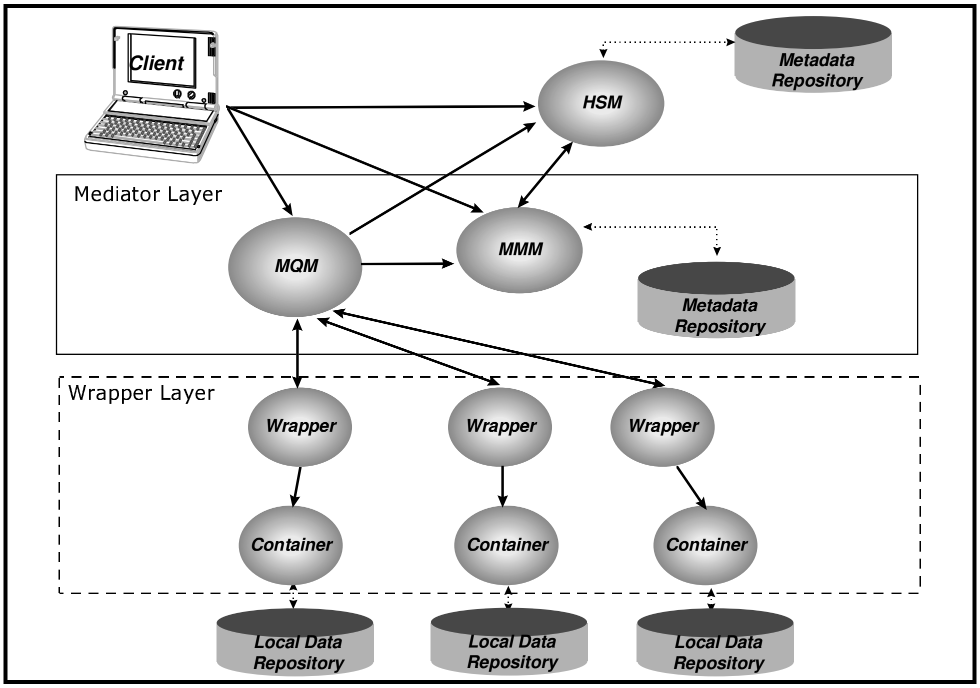

Figure 1 - HIMPAR Components

Our experience with the project and implementation of the HIMPAR prototype has shown that the CORBA standard provides a very useful methodology to be applied in the design and implementation of distributed systems based on the object technology. This becomes quite clear if we consider the implementation of an object-oriented HDDS without the resources provided by CORBA implementations. Without the use of an ORB, the implementation tools would be the client-server architecture with a communication protocol, such as, for instance, the TCP/IP. To build the required interoperability layer it would be firstly necessary to implement the functionality that are equivalent to those provided by ORBs. Since this task involves a lot of effort to be spent in programming, its cost is too high to justify its use in the implementation of a specific HDDS. Note that such a system would face interoperability problems on account of the lack of standards to relate the HDDS and the other information systems that one might need to integrate. Also, a system based on proprietary solutions increases interoperability problems because of the lack of standards between the HDDS and the candidate systems to be integrated.

2.1 Related work

We can find in the literature a large variety of projects concerning HDDSs. The proposed solutions can be classified according to their autonomy degree and the type of integration among the components of the system, which range from strongly coupled to weakly coupled systems (RAM, 1991; SHET, 1990). Strongly coupled systems have the advantage of the high level of synchronization among the components of the system, which leads to an efficient global processing. However, the creation and maintenance of a unified global integration schema becomes increasingly hard to manage as the number of component systems increase. On the other hand, in weakly coupled systems there is no integration schema. Since no global structure exists, the problems associated with the creation, maintenance and storage of the global schema are eliminated. However, global users must know the local representation of the data that they intend to access, as well as its location. In this Section, we present a description of some projects concerning the several different research fields on HDDSs. At the end of this Section, we point out the differences and similarities between these works and our own.

HDDSs with Global Schema. Projects Pegasus (DU, 1996), UniSQL/M (KIM *et al*, 1993), MERMAID (TEMPLETON, 1987), and IRO-DB (GARDARIN, 1996) are examples of systems adopting the global schema solution. These projects have research lines that study the resolution of conflicts among different schemas and data models. Although the existence of a unified global schema provides complete transparency to data access, the global system scaling is an unsolved problem.

Federate Databases. In these systems, there is no unified global schema; each local component has both an import and an export schemas (HEIMBIGNER, 1985; PU, 1987). The import schema is a description of the information shared between the local component and the global system. The import schema is a description of the origin and representation of the data from the remote nodes that can be globally accessed. The integration of the schemas is static. The alteration of local schemas and the addition of new information sources require the import schemas to be changed accordingly. Thus, the scaling and maintenance of such systems is hard to accomplish when one requires the integration of a large number of information sources.

Multibase Query Languages. In contrast to the idea of a unified system capable of resolving all the conflicts involving entities of the local schema, systems based on multibase query languages have no integration schema. In this model, the global system supports all the global transactions through the use of query language tools which provide the integration of information at the local DBMSs. MRDSM (LITWIN, 1987), OMNIBASE (RUSINKIEWICZ, 1989) and CALIDA (JACOBSON *et al*, 1988) are projects that accomplish database interaction using multibase query languages. This proposal faces no problems related to the creation and maintenance of a global schema. However, this approach does not provide a transparent data access to accomplish query formulation, users must have information about the distribution and the semantics of data.

Distributed Object Management (DOM). Another way to model heterogeneous distributed systems is to represent the resources of the system as a collection of interacting objects (PITOURA, 1995; ÖZU, 1994; MANOLA *et al*., 1992). Each component system defines a service interface and provides the implementation of such services. The OMA (OMG, 1995) architecture and the ODMG (CATTEL, 1997) model are important research works devoted to this approach. MIND - ***METU In****teroperable* ***DBMS*** - - (DOGAC *et al.*, 1995; DOGAC, 1996) and Jupiter (MURPHY, 1995) are HDDSs designed on a DOM platform. In the MIND system, the integration of local sources is done through the classical approach of global schema, while Jupiter uses the multibase language approach.

Mediators. Another proposal of a generic architecture for the integration of information sources involves the systems based on Mediators (*Intelligent Information Integration (I^3) Mediation*) (WIEDERHOLD, 1992; WIEDERHOLD, 1995). Several projects based on this model have been developed, such as the projects TSIMMIS

(PAPAKONSTANTINOU, 1995), Garlic (CAREY *et al.*, 1995), DISCO (TOMASIC, 1995) and DIOM (LIU, 1996). These projects intend to integrate structured and non-structured (with no data schema) data sources. They also deal with issues related to the diversity in querying power of the different data sources and propose several techniques to handle the reformulation of queries so as to resolve this mismatch.

The HIMPAR system is based on the architecture of Mediators. In this kind of architecture, we create specialized Mediators that apply to a specific application domain. Differently from the strongly coupled systems, in HIMPAR there is no unified integration schema to integrate all the information sources. Thus, this architecture does not present the problems concerning the creation and maintenance of the global schema. On the other hand, data access is transparent in this approach, differently from the weakly coupled systems, such as the multibase languages, for instance. Each Mediator represents a customized view that is intended to meet the needs of a specific group of users. This domain fragmentation enables a high level of autonomy and isolation of the architectural components. Thus, Mediators can be constructed and maintained independently. A Mediator that represents complex objects can be constructed out of simpler Mediators. Architectures based on Mediators are strongly scalable and comprise the integration of an increasing number of information sources. They are also capable of meeting the needs of different groups of users and reflect the natural organization usually observed in integrated systems. In many cases, there is no need that every data from every information source in the HDDS represent one only view. However, if an application requires such an integration level, the adequate Mediator is equivalent to a global schema, but only in this particular case.

The HIMPAR architecture has the same advantages as the other systems based on Mediators, additionally its approach is strongly based on standards. The data model used is based on the ODMG standard, while other projects, such as the TSIMMIS and the DIOM use specific models. The interoperability of architectural components is achieved through the CORBA standard. The communication between two Mediators and between a Mediator and a Wrapper uses a standard query language, the OQL (Object Query Language). The DISCO project also uses the ODMG data model, yet the communication among Mediators and Wrappers is accomplished through the use of logical operators instead of a standard query language defined by the ODMG specification (OQL). The use of these standards is intended to ease the integration of new systems. Database systems, which are compatible with the ODMG data model, are automatically integrated without the inclusion of specific Wrappers. It is important to note that the new generation Object Relational Database Management Systems (ORDBMS) also support OQL (through SQL3), so they can be also automatically integrated in the same way ODMG compatibles do. The use of the CORBA standard at the interoperability layer favors implementation of the system, since both the Wrapper modules and the Mediators can be implemented in any programming language, according to the preferences of each user group. In summary, the HIMPAR architecture is the combination of some features of the Mediators technology and the new standards for the object technology.

3 Mediator Metadata Management

Each Mediator component of the heterogeneous architecture must have a repository that stores the required information to the integration of the data involved in the application domain relative to this Mediator. These informations are called metadata of the Mediators. More precisely, these informations consist of data structures that store the Mediator's "global" schema, the export schemas of the repository sources (local

repositories or other integrated Mediators) and the mapping among the Mediator's global schema and the data schemas of the repositories that are integrated by the Mediator.
The main objective of these metadata is to allow the clients of a Mediator to issue its queries based on the "global" data model of the Mediator. These queries are decomposed automatically in subqueries that will be sent to the corresponding repositories. Partial query results received by the Mediator, will be packed and sent to the clients in a transparent way. Also, the incorporation of new repositories of data, to an already existent Mediator, should not demand changes to the query model of the Mediator clients. Therefore, we propose a specification of special interfaces of metadata and a group of services for these metadata management.

3.1 Mediators Metadata Specification

The data model used by the HIMPAR architecture is based on the ODMG-2.0 standard (CATTEL, 1997). This model consists of an object data model, an object definition language (ODL), a query language (OQL) and programming language interfaces (bindings). In this data model, an interface defines a signature associated with a type (or class) to allow the access to a certain object. An extent, associated with an interface, indicates the system to automatically maintain a collection of objects of the interface. Thus, a variable extent contains the collection of all the objects of the associated interface. Extents are the entry points to access the stored data.
TOMASIC (1995) proposes the extention of the interface and extent concepts of an interface to represent Mediators. The HIMPAR architecture adopts this idea of extending the concepts of the interface and extent of an interface. The extents of a Mediator interface type are composed by a group of other extents. Each extent of this group points to a collection of objects from a particular information source, associated to the interface type of the Mediator. Thus, the HIMPAR architecture defines a special metadata interface named ComponentExtent and its respective extent named as componentExtents.
ComponentExtent is a standard interface that associates multiple extensions to a type interface defined in the Mediator. Each ComponentExtent is associated to a type interface that represents resident objects in a local information source; this interface contains methods responsible for the mapping of the information between the data types of the Mediator and the data types of the local sources. The interface ComponentExtent is defined as follows:

```
interface ComponentExtent
(extent componentExtents
key name)
{   attribute String name;
    attribute Type interface;
    attribute String localTypeName;
    attribute Map mapList (MAX _NUM_ATRIB);
    attribute String origin;
   // operations
   Id idGlobalAttrib( string globalAttribName );
   String globalAttribName ( Id idGlobalAttrib );
   Map returnLocalAttrib( in Id idGlobalAttrib );
}
```

The name attribute stores the name of the extent that contains the data of the origin repository identified in the attribute origin. The interface attribute contains a reference to the object interface of the Mediator that integrates the type of the origin repository specified in the localTypeName attribute. The mapList attribute holds an indexed list of references to objects of the type specified by the Map interface which maps the attributes of the "global" interface of the Mediator into the attributes of the origin repository. The Mediator uses the returnLocalAttrib mapping function on the global queries to build the derived local queries.

```
interface Map
{
   Id idLocalAttrib;
   String nameLocalAttrib;
   String sigMethodDomainMap;
   String sigMethodMapKey;
}
```

In the Map interface the idLocalAttrib attribute stores an index that identifies the attribute inside of its own local interface. The sigMethodDomainMap attribute contains the signature of the method (stored in the repository of the metadata of the Mediator) that converts the value of the attribute of the domain of the Mediator into the domain of the origin repository, if necessary. The sigMethodMapKey attribute is only used when the attribute influences the indexation of the involved extent. In this case, a conversion from the value of the key in the domain of the Mediator into the domain of the origin repository takes place.

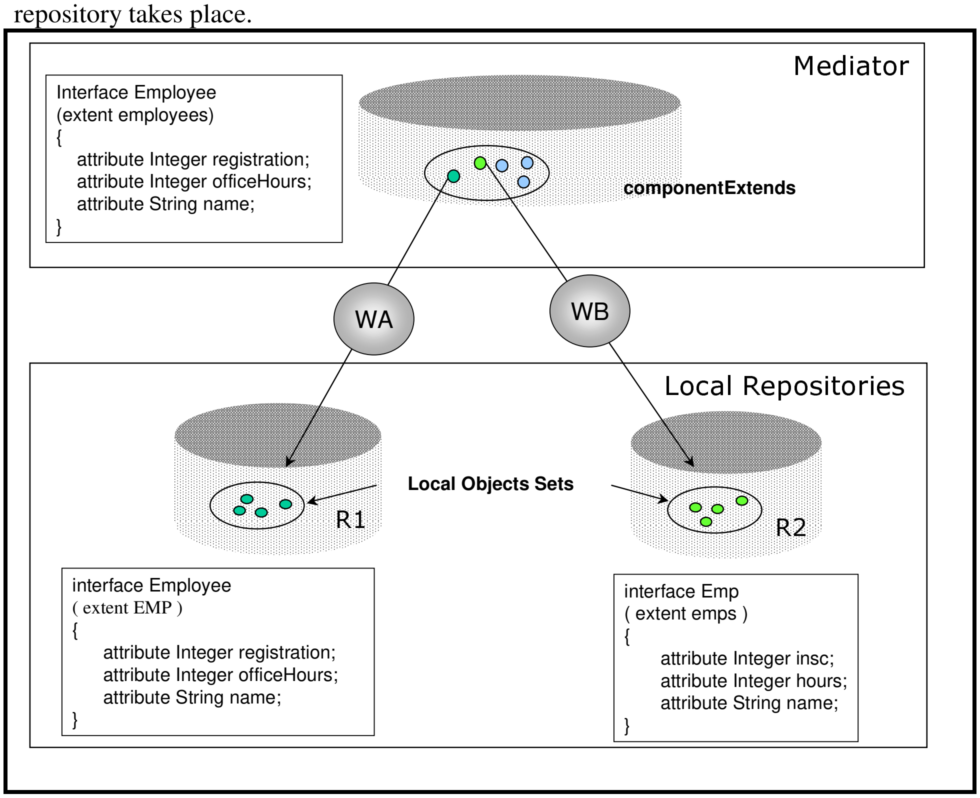

Figure 2 – Mediator Interface Mapping.

The extents of the types of the Mediator are maintained through the instanciation or exclusion of objects of the ComponentExtent type. The mapList attribute and the returnLocalAttrib operation of the ComponentExtent interface aim at the conversion of attributes among the types defined in the Mediator and the types defined in the repository sources. The specification of the Map metadata is an extension of the MMM of HIMPAR presented in (PIRES, 1996), which was restricted to the automatic conversion of attributes names. Using the information contained in these metadata, one can query all the extensions associated to a type interface of the Mediator, as illustrated in Figure 2.

For example, consider two repositories of data that contain information on employees of a company. The repository R1 contains the type Employee where its extension is named EMP and the repository R2 contains the type Emp where its extension is named emps, according to the IDL definitions (export schema) as follows:

```
interface Employee                          interface Emp
{                                           {
      attribute Integer registration;             attribute Integer insc;
      attribute Integer officeHours;              attribute Integer hours;
      attribute String name;                      attribute String name;
}                                           }
```

For integration of these information a type must be defined in the Mediator which corresponds to the data types of each local repository. Thus, using ODL, the interface type Employee is defined as follows:

```
Interface Employee
{
   attribute Integer registration;
   attribute Integer officeHours;
   attribute String name;
}
```

Objects of the ComponentExtent type, representing the objects of the local repositories must be created. The HIMPAR architecture provides a specific syntax for creation of new objects of the ComponentExtent type:

```
create ComponentExtent EMP of Employee
        localType = Employee origin = WA repository = R1
create ComponentExtent emps of Employee
        localType = Emp origin = WB repository = R2
```

Here WA and WB are object Wrapper identifiers which access the repositories R1 and R2, respectively. These expressions create two objects of the ComponentExtent type inserting them, automatically, to the componentExtents extend. The names EMP and emps of each ComponentExtent are determined by the collection names, which contains the objects (data) of each local repository. Thus, each ComponentExtent represents a collection of data in a local repository. As can be seen in this example, the extents associated to a type of the Mediator can be inserted through the insertion of objects of the ComponentExtent type.

The use of this metadata provides the recovery of all the extents associated to a type of the Mediator, starting with the declaration of an extension in the definition of the interface. Thus, the following declaration of the Employee interface assumes implicitly the definition of a corresponding query to the employees extent:

```
Interface Employee
(extent employees)
{
```

```
    attribute Integer registration;
    attribute Integer officeHours;
    attribute String name;
}
```

```
    define employees as
      flatten (        select x from x in componentExtents
                     where x.interface = Employee );
```

This query definition expression accesses the componentExtents extent that is stored in the repository of the Mediator and dynamically select all of the extents that are associated with the Employee type. Thus, the following query dynamically accesses all the extents defined for the Employee type.

```
select x.name from x in employee
         where x.registration > 190865;
```

A query involving the Employee type issued by a client program to this Mediator, with the previous ODL definitions, will have access to the repositories R1 and R2 generating the desired results. Notice that the extents associated to the types of the Mediators are always associated to the views, while the extents associated to the types of the local repositories are stored as objects of the ComponentExtent type. This model allow the management of the insertions and exclusions of data repositories in a simple way, facilitating the integration of a great number of information into the system, without alterations inside the schema of the Mediator.

The **interfaces** are classified in two categories: base interfaces and composed interfaces. The semantics of the construction of interfaces, as specified by ODMG 2.0 standard, is extended with operations for the creation of inter-repository views, through the composition of interfaces. These operations are aggregation, generalization, specialization, and import/ suppression of attributes. These view definitions are stored in the metadata repository of the Mediator under the form of complex objects (LIU, 1996).

A **base interface** refers to the interface in which all the types involved in the definition are originating from only one data source. With the objective of providing connections and abstract relationships among data from different repositories, each entity type, defined in the repositories, of data that it is visible for the Mediator, should be declared in terms of an base interface.

The **composed interfaces** are built through successive applications of the available composition operators in HIMPAR: specialization, generalization, aggregation, and importation/ suppression of attributes. These operators will be described further on. A composed interface can be seen as a strongly typed contract among interfaces to provide a small collection of data and operations semantically related. Each interface defines an expected behavior and a group of responsibilities of a group of data and operations. The scope of a composed interface is the number of data repositories on which the definition of the interface is based.

Next, we describe the operators for the creation of composed interfaces:

Aggregation - this operator composes a new interface starting from a number of existent interfaces so that the new generated interface constitutes a joined view of the component interfaces. Objects of the newly created interface have direct access to the objects of the component interfaces that are manipulated in a transparent way. Therefore, the operations defined in the component interfaces can be invoked through the aggregated interface. One of the advantages of the use of the aggregation operation is to allow objects that are located in different repositories to be connected in agreement with the necessity of the consumers of the information, controlled by the Mediator.

Generalization – this operator provides the merging of several semantically similar interfaces (although different) in a more generalized interface. The intention of the generalization is to define a new interface starting from the extraction of the common properties and operations of some existent base or composed interfaces. This operation allows the manipulation of objects, which reside in different data repositories, through the generalized interface.

Specialization - this operator is used for the construction of a new interface in terms of some pre-existent interface, through the addition of new attributes, relationships and/or operations. This operator defines a new interface through the specialization of an existent base interface, followed by the inclusion of the new properties.

Import/Suppression of Attributes – the import operator is used to import selected data portions of a certain information source instead of importing everything that is available. This operator is used with the others operators during the definition of the desired composed interface, allowing a great flexibility in view creation.

3.2 Case Study

This section presents a case study to exemplify the definition of the data types of a Mediator and the definition of views on these types. The following example describes the application: a university needs to control the publications produced by its teachers and masters degree students. Assuming that, the available important sources of information include three repositories, their description is as follows: R1 - a library with information about books, published by the university teachers, and stored in a relational DBMS (Oracle). R2 - a documentation center, with information about papers and technical reports, and stored in an object database (O2). R3 - another documentation center, with information about published theses, and stored in another object database (O2).

The export schemas are defined through OMG IDL as described following (an IDL module for each data repository):

```
module R1
{
   interface Book;
   interface Author;
   interface Publisher;

   interface Book {
       attribute string bookName;
       attribute string code;
       attribute long pubYear;
       attribute sequence<Author> authors;
       attribute Publisher publisher;
   };

   interface Author{
       attribute string name;
       attribute string espec;
       attribute sequence <Book> books;
   };

   interface Publisher {
```

```
        attribute string name;
        attribute string place;
        attribute sequence<Book> books;
    };

};
```

```
module R2
{       interface TecPub;
        interface Paper;
        interface TechnicalReport;

        interface TecPub{
              attribute string title;
              attribute long pubDate;
              attribute sequence<string> authorsNames;
        };
        interface Paper: TecPub {
             attribute string congress;
             attribute long pages;
        };
          interface TechnicalReport: TecPub {
             attribute string code;
        };
};
```

```
module R3
{
     interface Thesis;
     interface Student;
     interface Advisor;

     interface Thesis {
         attribute string title;
         attribute string grade;
         attribute long date;
         attribute Student student;
    };

    interface Student {
        attribute string name;
        attribute string regist;
        attribute string time;
        attribute sequence < Thesis > thesis;
        attribute sequence < Advisor > advisor;
    };

    interface Advisor {
        attribute string name;
        attribute sequence< Student > students;
```

```
      attribute string dep;
};
```

After the identification of the possible equivalencies among types, which are resident in different data repositories, the types, corresponding to the types desired in the local data repositories, are defined in the Mediator. In the case the same type appears in more than one repository, it will be mapped for one single Mediator type. The composition operators are used to help the integration and the construction of the user's query.

```
interface Publication
 (extend publications){
    attribute string title;
    attribute long publicationDate;
 };

 interface Book: Publication
 (extend books){
       attribute string code;
       relationship Set<Author> authorsList inverse Author::booksList;
       relationship Set< Publisher > publisher inverse Publisher:: publicationList;
 };

interface Thesis: Publication
(extend theses){
          attribute string grade;
          relationship Student student inverse Student::thesesList;
 }
```

The Publication interface is a generalization of the Book interface and Thesis interface and implicitly defines the following OQL query for the publications extent:

```
define publications as
flatten(   select x from x in componentExtents
           where x.interface = Publication and
                 x.interface = Book and
                 x.interface = Thesis );
```

The next section presents the MMM implementation using some features of the GOA++ system.

4 MMM - The Mediator Metadata Manager and The GOA++

The Mediator Metadata Manager (MMM) stores all the information required for the integration of data involved in the application domain of the Mediator. This object is responsible for the creation, maintenance and management of the information contained in the Mediator metadata repositories. The MMM knows the definition of all the interfaces that composes the Mediator schema and uses the special interface ComponentExtent to integrate the information from local repositories. Next, we describe the MMM available services:

The operation export is invoked by the MQM (Mediator Query Manager) object to recover the state of an object of the type ComponentExtent. The MQM object uses the mapping contained in the metadata ComponentExtent to make the attribute conversion and the automatic decomposition of queries for the target repositories. These objects have as an entry parameter (extentName), the name of a target extent of a specific

global query and as out parameter (componentExtentList) a list that contains the component extents associated to the target extent of the query.
The Clients of HIMPAR to query the Mediator integration schema, and the MQM objects to syntactically analyze the OQL global queries submitted by the Clients of HIMPAR, invokes the operation querySchema. The operation querySchema has as entry parameter (query) an OQL query involving the data schema of the mediator and as out parameter (result) the objects associated to the result of the query. It is important to note that, the possibility to have the metadata stored as complex objects of an object database, that is understood by a standard query language like OQL, allows the construction of mediators in a flexible and dynamic way.
The operation createBaseInterface is invoked by the Clients of HIMPAR to create an interface in the metadata repository. This operation has as entry parameter an interface declaration in ODL. It doesn't have out parameter. Its effect is the creation of an interface object and the insertion of this object in the collection of the Mediator schema.
The Clients of HIMPAR to create, in the metadata repository of the mediator, a view composed by interfaces that can reside in distinct data repositories, invokes the operation createCompositInterface. This operation has as entry parameters a composition operator and a list of interface names or a key word used to recover a group of interfaces through OQL. Its effect is the creation of an object that has a view definition and the insertion of this object in the collection of the mediator schema. The view object constitutes an OQL definition of a query involving the interfaces that composes the view. This query will be executed on the target extents of the involved local repositories.
The operations agregateInterfaces, generalizeInterfaces and specializeInterfaces correspond to the operations of creation of the composed interfaces whose composition operators are respectively aggregation, generalization and specialization.
The operation hideAttributes is used during the creation of a composed interface to omit some attributes of the component interfaces. The operation hideInterfaces hides certain interfaces of an already existent schema, of which one wants to use the definition of a subset of its interfaces. The operation importInterfaces imports certain interfaces of an already existent schema, of which one want to use the definition of a subset of its interfaces. The operation createComponentExtent create an object ComponentExtent that must be filled with the values of the mapping between the type interface of the mediator and the type interface of the local repositories.
The following code describes the IDL specification of the MMM object:

```
interface GMM {
 void          export( in string extentName,  out TComponentExtentList
componentExtentList)
               raises( schemaException );
 void          querySchema( in string query, out ResultType result )
               raises(schemaException);
 void          createBaseInterface ( in string ODL_Definition )
               raises(schemaException);
 void          createCompositInterface ( in string d_ODL_Est )
               raises(schemaException);
 void          createComponentExtent( TExtComp newExtent )
               raises(schemaException);
 void          agregateInterfaces(TLstInterface lstInterface)
               raises(schemaException);
```

```
void        generalizeInterfaces(TLstInterface lstInterface)
            raises(schemaException);
void        especializeInterfaces(TLstInterface lstInterface)
            raises(schemaException);
void        hideAttribute( TAttribName attributeName,
                                   TypeInterface interface )
            raises(schemaException);
void        hideInterfaces(in TLstInterface lstInterface)
            raises(schemaException);
void        importInterfaces(in TLstInterface lstInterface)
            raises(schemaException);}
```

The implementation of the metadata management services demands some facilities from an OODBS, such as, the ability to manage schemas and to execute queries on these schemas. Therefore, the work included the use of the GOA++ Schema Manager (SM). The SM services are used to create base classes for metadata creation. These base classes are GOA_Interface and GOA_View (GOA_Generalization, GOA_Specialization, and GOA_Aggregation). The mediator metadata repository contains objects that are instances of these meta classes and they are inserted in the collection of the mediator schema.

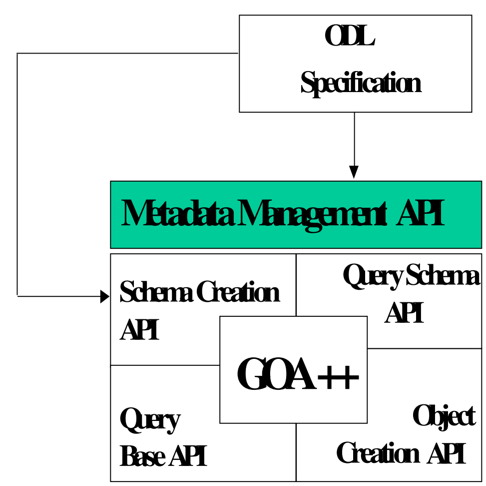

Figure 3 – The GMM to GOA++ interface

The API for the metadata management is built on the top of the Schema Manager of the GOA++. Figure 3 presents a general view of the architecture.

5 Conclusion

The integration of different information systems is a crucial issue in many computational environments. Several research projects indicate that Heterogeneous Database Systems (HDDSs), in particular those that support object-orientation, are a viable solution to the problem of integrating information systems in heterogeneous distributed environments. Within the context of HDDSs, we propose the heterogeneous architecture HIMPAR (***H**eterogeneous **I**nteroperable **M**ediators and **P**arallel **Ar**chitecture*). The main goal of this work is to develop an open and extensible architecture, for the interoperability of resources and services within a distributed and heterogeneous environment.

HIMPAR provides semantic integration through the construction of specialized mediators that apply to a specific application domain. There is no unique integration schema to integrate all the information sources, as in the strongly coupled systems (Global Schema). Therefore, the HIMPAR does not face any problems concerning the creation and maintenance of the global schema. On the other hand, the HIMPAR approach provides transparency to data access, differently from the weakly coupled systems, such as the multibase languages. Each Mediator represents a view designed for a specific group of information users.

This work presented a metadata management system for mediators providing services for data type definition of each mediator repository. The definition of interfaces offers type composition through aggregation and generalization. The MMM implementation uses schema manager and query processing services of the GOA++ object server. This implementation has shown a great flexibility due to use of high level services found in object management systems. MMM services are available through an API on top of GOA++ and can be issued remotely through CORBA or by local API calls. MMM was designed for the HIMPAR architecture, however its services may be used by other HDDS based on mediators.

The architecture of HIMPAR is strongly adherent to object oriented standards. This approach has shown its adequacy in the integration of new systems. Database systems that are compatible with the ODMG-93 model are automatically integrated having no need of specific Wrappers. Note that, in case the data repository is an ODBMS compatible with the ODMG-93 standard, the Wrapper will send the sub-query directly to the DBMS, since there is no need of translating it. This is an important advantage of adopting the OQL as the communication means between the Mediator and the Wrapper.

References

BUKHRES, A.O., ELMAGARMID, A.K., 1996, *Object Oriented Multidatabase Systems*, Prentice Hall.

CAREY, M.J., *et al*, 1995, *Towards Heterogeneous Multimedia Information Systems: The Garlic Approach*. Technical Report, IBM Almaden Research Center, USA.

CATTEL, R. G. G., BARRY, D.K. (eds) 1997, *The Object Databases Standard: ODMG-2.0*, Morgan Kaufmann Publishers, USA.

DOGAC, A., *et al.*, 1995, *METU Interoperable Database System*, In: Technical Report 6-1, Software Research and Development Center, Middle East Technical University, Ankara Turkiye, Jun.

DU, W., SHAN, M., 1996, "Query Processing in Pegasus". In: (BUKHRES, 1996a), pp. 449-471.

GARDARIN, G., GANNOUNI, S., FINANCE, B., 1996, "IRO-DB: A Distributed System Federating Object and Relational Databases", In: (BUKHRES, 1996a), pp. 684-712.

HEIMBIGNER, D., MCLEOD, D., 1985, "A Federated Architecture for Information Management". In: *ACM Trans. Office Information Systems*, v.3(3), pp. 253-278.

JACOBSON, G., *et al.*, 1988, "CALIDA: A Knowledge-based System for Integrating Multiple Heterogeneous Databases". In: *Proceedings of 3th Int. Conf. on Data and Knowledge Bases*, Jerusalem, Israel, Jun.

KIM, W., *et al.*, 1993, "On Resolving Schematic Heterogeneity in Multidatabase Systems", In: *Distributed and Parallel Databases*, v.1(3).

LITWIN, W., ABDELLATIF, A., 1987, "An Overview of the Multi-Database Manipulation Language MDSL". In: *Proceedings IEEE*, v.75(5), pp. 621-632, May.

LIU, L., PU, C., LEE, Y., 1996, "An Adaptive approach to query mediation across heterogeneous databases". In: *Proceedings of the Int. Conf. on Cooperative Information Systems*, IEEE Press, pp. 144-156, Jun.

MANOLA, F., *et al.*, 1992, Distributed Object Management, In: *International Journal of Intelligent and Cooperative Information Systems*, Jun.

MAURO R C, et al., 1997, "GOA++: tecnology, implementation and extensions on the Object Manager", In: *Proceedings of the XII* Brazilian Symp. Databases, Brasil, Oct.(in Portuguese).

MEYER, L.A.V.C, MATTOSO, M.L.Q. "Parallel Query Processing in Shared-Nothing Object Database Systems," in *Proc. 3rd International Meeting on Vector and Parallel Processing (VECPAR'98)*, Porto, Portugal, June 1998, pp.1007-1020.

MURPHY, J., GRIMSON, J., 1995, "The Jupiter System: An Environment for Multidatabase Interoperability", *Information and Software Technology,* v.37, pp. 503-513.

O2 TECHNOLOGY, 1996, *The O2 System Administration Guide*, release 4.6, Apr.

OMG (Object Management Group) Home Page, 1998, Avaliable at: http://www.omg.org/, Abr.

OMG (Object Management Group), 1995, *The Common Object Request Broker: Architecture and Specification*, Rev. 2.0, Jul.

PAPAKONSTANTINOU, Y., *et al*, 1995, "The TSIMMIS Approach to Mediation: Data Models and Languages". In: *NGITS workshop* .

PIRES, P.F., 1997, HIMPAR, uma Arquitetura para Interoperabilidade de Objetos Distribuídos [HIMPAR, an Architecture for Distributed Object Interoperability], master's thesis, COPPE/UFRJ, Rio de Janeiro, Brazil, Apr. (in Portuguese).

PIRES, P.F., MATTOSO, M.L.Q., 1996, "Aspectos de Implementação na Arquitetura Heterogênea HIMPAR [Interoperability Issues in the HIMPAR Heterogeneous Architecture]," Proc. of the XXI Brazilian Symp. Databases, São Carlos, Brazil, Oct., pp.43-57 (in Portuguese).

PIRES, P.F., MATTOSO, M.L.Q., 1998, "A CORBA Based Architecture for Heterogeneous Information Source Interoperability", *25th Technology of Object-Oriented Languages and Systems (TOOLS-25 '97),* November 24-28, 1997, Melbourne Australia. IEEE Press, Junho, ISBN 0-8186-8485-2.

PITOURA, E., BUKHERS, O., ELMAGARMID, A., 1995, "Object Orientation in Multidatabase Systems". *In: ACM Computing Survey*, v.27(2), Jun.

PU, C., 1987, "Superdatabases: Transactions Across Database Boundaries". In: *Database Engineering*, v.10(3), pp. 143-149.

RAM, S., 1991, "Guest Editor's Introduction: Heterogeneous Distributed Database Systems". In: *IEEE Computer*, v.24(12), Dec.

RUSINKIEWICZ, M., *et al.*, 1989, "OMNIBASE: Design and Implementation of a Multidatabase System". In: *Proceedings of 1th Annual Symposium in Parallel and Distributed Processing*, Dallas, Texas, May.

SHETH, A.P., LARSON, J.A., 1990, "Federated Database System for Managing Distributed, Heterogeneous, and Autonomous Databases". In: *ACM Computing Surveys*, v.22(3), Sep.

TEMPLETON, M., *et al.*, 1987, "Mermaid - A Front End to Distributed Heterogeneous Databases". In: *Proceedings of the IEEE*, May.

TOMASIC, A. et al., 1995, *Scaling Heterogeneous Database and the design of DISCO*, Tech. Report nº 2704, INRIA, France, Nov.

VISIGENIC SOFTWARE, 1996, *VisiBroker C++ User's Guide version 2.0*, Visigenic Software, Inc., USA.

WIEDERHOLD, G., 1992, "Mediators in the architecture of future information systems". In: *IEEE Computer*, v.25, pp. 38-49.

WIEDERHOLD, G., 1995, "Value-added Mediation". In: *Proceedings of the IFIP DS-6 Conference*.