Integrating GIS Components with Mediators and CORBA

Bernd Amann

Cedric CNAM
292 Rue St. Martin
75141 Paris Cedex 03 France

May 15, 1996

Abstract

System integration is a fundamental problem for Geographic Information Systems (GIS). This is essentially due to the fact that available software is complex and would be costly to reimplement or to modify. Building systems that combine resources and services in a distributed and heterogeneous environment raises a number of new issues concerning the architectural framework for such systems. In this paper, we describe a mediator-based architecture for integrating spatial components. We discuss some issues concerning the mediation of spatial database queries and present a prototype implementation based on CORBA.

1 Introduction

Different information sources often contain "spatial" data in form of addresses, maps, geographical objects (cities, districts, ...). All of these information entities have a common spatial semantics which can be exploited for their integration. For example, a person's address in a phone book corresponds to a position on a map representing the person's home town. This fact explains the growing interest and necessity in integrating geographic data existing on different platforms and resulting from different application domains. For GIS, the systems integration problem [1] is essential the available software often is complex and would be costly to reimplement or to modify. Representative examples of system components in geographic applications are database management systems, modeling systems and graphical user interfaces.

Interoperability has become the general term for describing issues concerning the cooperation of software in the joint execution of a task [18]. Building systems that combine resources and services in a distributed and heterogeneous environment raises a number of new issues concerning the exchange, modification (transaction management) and integration (semantic interoperability) of distributed data. All of these issues depend on the system's architecture which becomes a key issue for their implementation.

The current evolution of client-server architectures to mediated architectures based on middleware standards like OMG's Common Object Request Broker Architecture (CORBA) [13] tries to tackle the problem of heterogeneity at the system level. These standards provide clients and servers with high-level programming interfaces by encapsulating differences in network protocols, programming languages and operating systems.

In this paper, we describe a mediator based architecture for integrating spatial components. This architecture corresponds to an embedded system configuration for many-component sys-

tems [1] and has been validated by a prototype implementation in top of CORBA. The prototype provides different Graphical User Interface (GUI) clients with a uniform access to several Database Management Systems (DBMS), i.e. an object-oriented (O_2) and two relational (Postgres95, mSQL) database management systems.

A mediator based architecture for implementing open GIS systems is presented in Section 2. Section 3 describes some issues concerning the mediation of spatial queries. In Section 4 we give a brief overview of CORBA and show how it can be used to implement the architecture presented in Section 2. Finally, a prototype implementation is presented in Section 5. This prototype adopts CORBA as middleware and provides access to different databases, i.e. an object-oriented DBMS and two relational DBMS, by means of a unique mediator interface.

2 A Mediator Based Architecture for GIS

Data heterogeneity is an essential issue to existing and future GIS applications and can be solved by adopting a layered architecture as it is proposed by the Open Geodata Interoperability Specification (OGIS) [4, 12], in general distributed database systems [20, 18], and in heterogeneous multimedia information systems (e.g. TSIMMIS [7] and Garlic [5]). An example of a system implementing the different layers of such an architecture is shown in Figure 1.

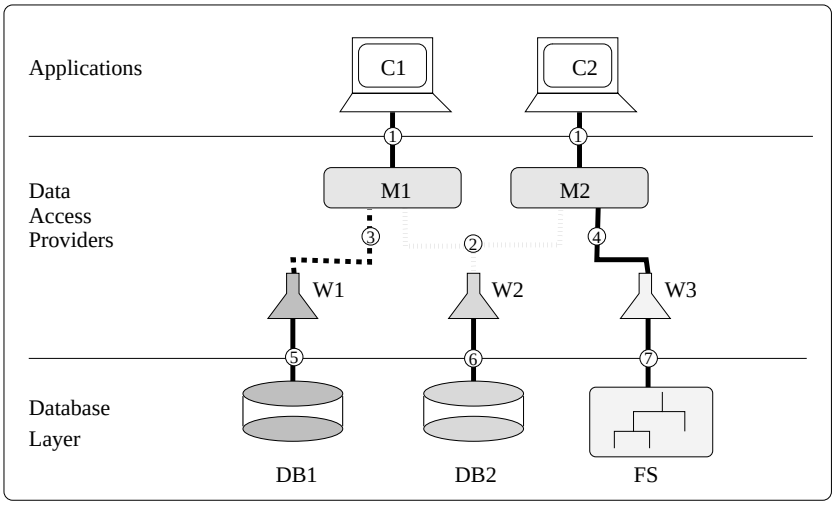

Figure 1: Three layered architecture

Three information sources (`DB1`, `DB2` and `FS`) can be accessed by different clients (`C1` and `C2`). The database layer is part of the system services level and provides persistence to both spatial and non-spatial data. `DB1` and `DB2` are full-fledged database management systems with advanced query languages : `DB1` is an object-oriented DBMS which provides GIS functionalities in form of standard OQL queries extended by some spatial operators as described in [19]. `DB2` is a relational DBMS which stores alphanumeric information (e.g. yellow pages, etc.) in form of relational tables. The query language is SQL. `FS` is a web server which stores a set of files containing any information (maps, documents, images, audio, video) concerning cities, buildings, persons, etc... These files are accessed via the World-Wide Web protocol.

Each data source is accessed by a different wrapper (`W1`, `W2` and `W3`) transforming access and update requests according the corresponding interface definition, i.e. OQL queries for interface ⑤, SQL queries for interface ⑥ and URL requests for interface ⑦).

The whole information is available to GUI clients `C1` and `C2` by the way of interface ①. These

clients, then, can access different thematic maps, documents and images and modify corresponding alphanumeric information. Clients are not aware of the underlying information sources and communicate with mediators for accessing information. The main mediator task is to decompose and distribute client requests to the different wrappers and reassemble their responses.

Mediators `M1` and `M2` access database `DB2` through some common interface ② which allows to modify data provided by wrapper `W2`. Wrappers `W1` and `W3` are accessed through interfaces ③ and ④ respectively. Interface ③ can be used for evaluating spatial queries on database `DB1` whereas interface ④ gives access to data files via the World-Wide Web protocol.

3 Mediators for GIS

Applications access data via data access providers [4] which play the role of *mediators* [24], global layers [18] or master components [1]. Mediators break the strong coupling between servers and clients, caused by low-level interfaces allowing only restricted forms of abstraction. They are generally based on common data models restricted to some application domain and include various information processing (linking) tasks like format conversion (transformation operations), data access and integration (accessor and constructor operations) and data selection (filtering operations). These tasks might be implemented only partially and it is possible to classify integrating architectures according to the number of operations available [1].

An important mediator task is to decompose and distribute client requests to the different wrappers and reassemble their responses. Mediators provide integrated views of the underlying information. These views can be hard-coded but also be based on a high level source description as it is proposed in [15, 2, 16, 22].

We will try to clarify the role of a spatial mediator by an example. We assume that wrappers and mediators are described in the ODL interface definition language [6] which is part of the ODMG-93 standard and adapts OMG's IDL (Section 4) for database applications by adding new type constructors (set, list), object relationships and class extents. In the following example we will use the OQL query language, which is also part of the ODMG-93 standard, to query (spatial) databases. A detailed description of ODL and OQL is outside the scope of this paper and can be found in [6].

Mediator Interface : The geometry of spatial objects, e.g. roads or high-tension lines, is described in a spaghetti model [19], i.e. no topological relationships are stored in the database. Each spatial object is composed of several geometric components (point, line, polygon). This allows, for example to describe a high-tension line by its electric cable (line) and its pylons (points). The topology of each component is specified by an attribute (`topo`) with values "point", "line" and "region". Two methods allow the detection (`intersects`) and the calculation (`clipping`) of the intersection of two spatial objects. Roads are identified by their name. The kind of a road, e.g. street or highway, is stored in the attribute `kind`. A city is identified by its zip-code and contains its name and population. The complete ODL mediator interface definition is described in Appendix A.

Wrapper Interfaces : Spatial data is stored in a spatial database SDB implementing a network model, where the adjancy relationships between lines are stored explicitly. Lines, points and polygons are described in different subclasses of class `GEOM`. Spatial objects are identified by an object identifier stored in class `GEOM`. A relational database RDB stores alphanumeric information about highways (name) and cities (name, zip-code, population). The complete ODL description of both wrapper interfaces is defined in Appendix B.

All data provided by a wrapper or mediator is described in the corresponding interface definition. More precisely, this means that wrappers and mediators have to be able to handle all

queries on data they describe. The main mediator task is to handle mediator queries by using the existing wrapper interfaces for accessing data.

In the following, we will present some issues related to the rewriting of spatial queries and operations. We assume the existence of a a query rewriting algorithm which can be used for rewriting mediator (OQL) queries into subqueries evaluated against the different wrappers as it can be found in [9]. This algorithm is based on OQL and uses semantic knowledge for the rewriting of standard OQL queries.

Example 1 The first example is a mediator query which returns the names of all cities with a population of more than 10 000 citizens :

```
select c.name
  from c in cities
 where c.population > 10000
```

Relation `CITIES` in database `RDB` contains all necessary information (name and population) and it is easy to see that the answer can be obtained by the following (SQL) query evaluated against the relational database wrapper `RDB` :

```
select C.NAME
  from CITIES@RDB C
 where C.POPULATION > 10000)
```

The notation `CITIES@RDB` is used to make the fact that the named variable (relation) `CITIES` is provided by wrapper `RDB` more explicit.

Example 2 The following query selects the names of all cities on highway A86 with more than 10 000 citizens.

```
select c.name
  from r in roads, c in cities
 where r.kind = 'highway'
   and r.name = ''A86''
   and c.population > 10000
   and r.geom->intersects(c.geom)
```

This query involves information provided by wrappers `RDB` (road name and city population) and `SDB` (geometries). It is necessary to join objects of both databases by using the logical object identifier OID. The rewriting algorithm might generate different solutions. Each solution corresponds to a set of wrapper subqueries with different evaluation properties:

1. The first solution assumes that a wrapper query can only use *local* information in the `from`-clause, i.e. wrappers cannot access external data, i.e. data situated outside the corresponding database.

   The first sub-query `SQ1` selects the names of cities of more than 10 000 inhabitants and their logical object identifiers as well as the logical object identifiers of highways stored in database `RDB` :

   ```
   SQ1 := select struct(cname: C.NAME, coid: C.OID, hoid: H.OID)
          from C in CITIES@RDB, H in HIGHWAYS@RDB
         where C.POPULATION > 10000
           and H.NAME = 'A86'
   ```

The second sub-query SQ2 evaluated by wrapper SDB returns all couples of logical object identifiers (HG, CG) of intersecting geometries in SDB :

```
R1 := select struct(hgoid: HG.OID, cgoid: CG.OID)
        from HG in GEOMS@SDB, CG in GEOMS@SDB
       where HG->inter(CG)
```

The final query is evaluated by the mediator and gives the result by joining the information in SQ1 and SQ2 :

```
select b.cname
  from a in SQ1, b in SQ2
 where a.hoid = b.hgoid and a.coid = b.cgoid
```

or

```
select b.cname
  from a in (select struct(cname: C.NAME, coid: C.OID, hoid: H.OID)
               from C in CITIES@RDB, H in HIGHWAYS@RDB
              where C.POPULATION > 10000
                and H.NAME = 'A86')
       b in (select struct(hgoid: HG.OID, cgoid: CG.OID)
               from HG in GEOMS@SDB, CG in GEOMS@SDB
              where HG->inter(CG))
 where a.hgoid = b.hoid and a.cgoid = b.coid
```

Sub-queries SQ1 and SQ2 are evaluated by the corresponding wrapper interfaces and create two intermediate (constant) results. Each query is local [9] in the sense that it can be evaluated only by using data stored in the corresponding wrapper database.

Observe that query SQ2 evaluates the intersection of *all* spatial objects stored in the spatial database which is expensive in time and space. It is easy to see that it would be preferable to restrict the search space by selecting first all geometric objects corresponding to cities with more than 10 000 habitants and to highway A86 (sub-query SQ1), before testing their intersection. This is only possible by a sub-query evaluated by wrapper SDB using the result of a sub-query SQ1 (evaluated by wrapper RDB).

2. The second solution assumes that wrapper SDB accepts *external* information, i.e. the result of sub-query SQ1. After having evaluated sub-query SQ1, the mediator sends the following query to wrapper SDB, where RQ1 corresponds to the result of SQ1.

```
SQ3 = select r.cname
        from HG in GEOMS@SDB, CG in GEOMS@SDB, r in RQ1
       where HG.OID = r.hoid
         and CG.OID = r.coid
         and HG->inter(CG)
```

Observe that SQ3 depends on lambda variable r associated with query SQ1, i.e. SQ3 is not dependency-free [9]. Nevertheless, wrapper SDB can evaluate query SQ3 since the result RQ1) is a *constant* set of tuples.

4 Common Object Request Broker Architecture

Object-oriented concepts (inheritance, encapsulation, behavior) provide a natural framework for heterogeneous, autonomous and distributed systems [18]. According to a distributed object architecture, the different system components are modeled as objects with methods implementing various services. The object-oriented approach is well-adapted to the implementation of complex mediator tasks integrating and transforming data from different components in the database layer. Each component provides its own interface definition with different methods for accessing the provided services. Mediators then integrate these services in order to present a unique interface to their clients.

The object management architecture (OMA) [13] represents one of the prevailing standardization efforts in object based architectures. Other standardization efforts for distributed object-based architectures are OSF's DCE [14], Microsoft's OLE/COM and IBM's (D)SOM. The Common Object Request Broker Architecture (CORBA) [13] represent a core part of OMA. CORBA achieves interoperability by an object oriented client-server model. Clients are separated from providers of services through well-defined interfaces described in the Interface Definition Language (IDL). The IDL interface definition contains all method signatures and error exceptions which can be handled by objects implementing the corresponding service. A client issues a request by sending a message to an object identified by an object reference. An object implementation carries out the requested service. The Object Request Broker (ORB) provides all communication infrastructure needed (1) to deliver requests and their associated parameters to objects and (2) to return the results back to the clients. This infrastructure is often referred to as ORB core. More generally, the ORB core is responsible for object location, connection management and data exchange between clients and servers.

Figure 2 shows a distributed CORBA-based implementation of the layered architecture which is described in Section 2. The Object Request Broker (ORB) is the key communication ele-

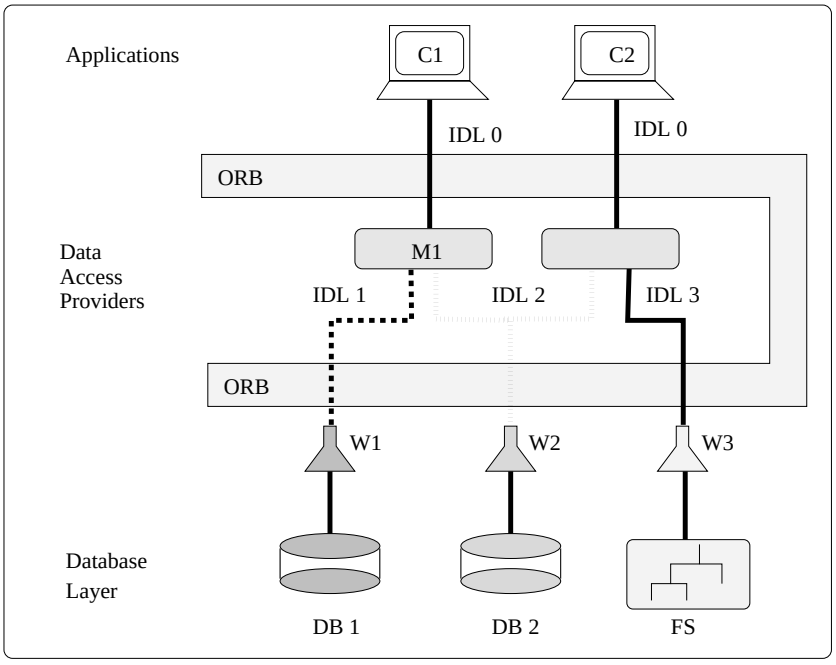

Figure 2: Mediated Architecture using CORBA

ment and provides the mechanisms by which objects transparently make requests and receive responses. Observe that the each interface of Figure 1 corresponds to a different IDL interface definition.

CORBA facilitates the integration of the different components (data servers, mediators and applications) in several ways :

1. IDL interface definitions can be compiled into different languages (Java, C++), and it is possible to create, for example, Java and C++ clients for the same service implemented in C++ without any modification on the server side.

2. The physical/logical location of the different components is transparent to the software and managed by the ORB core. It is possible to move the different modules in the ORB environment which can include different hosts and local networks.

5 Prototype Implementation

In this Section we present an integration of an existing GIS prototype [17] and the ORBeline [8] CORBA software. The prototype architecture is shown in Figure 3. The main functionalities

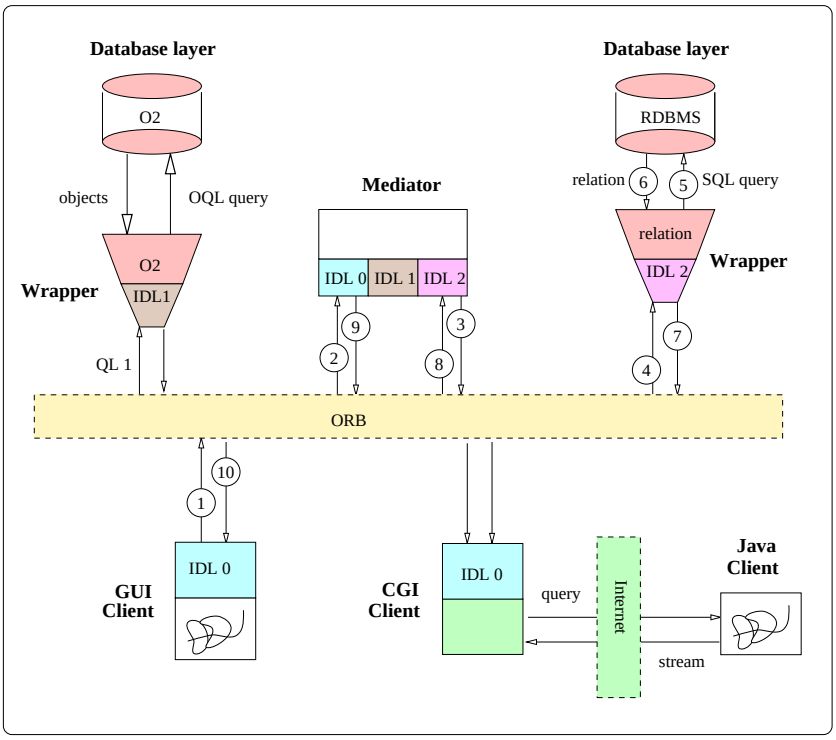

Figure 3: Prototype Architecture

can be summarized as follows :

- Clients access data by sending queries to mediators. These queries are standard OQL/SQL queries involving alphanumeric, spatial and visual (e.g. point/rectangle on the screen) data.

- Mediators reply by sets of spatial objects which are processed by the clients (e.g. displayed on the screen).

- Mediators play the role of data access servers between clients and the database layer. They encapsulate representations of multiple data sources and forward queries to the different wrapper services.

- The database layer is implemented by the object-oriented Database Management Systems (DBMS) O_2 (O_2Technology) [3] and two relational DBMS, i.e. Postgres95, an extended and freely available version of Postgres [21] and mSQL[1]. Wrappers are database clients (written in C/C++) built on top of an export schema described in [17]. Mediators communicate with wrappers and GUI clients via the ORBeline[2], version 1.0 [8] CORBA implementation. The IDL description of the mediator interface is a straight-forward translation of the O2 export schema [17] into IDL .

- Currently, there exist two graphical user interface clients. The first interface is a modification of the interface described in [17] where the strongly coupled O2/C++ connection has been replaced by a more flexible CORBA client/server communication. The second interface is implemented in form of a Java applet accessing mediator functionalities through a Common Gateway Interface (CGI) client transforming CORBA structures into a proper data exchange format. The JAVA interface functionalities are essentially the same as in the C++ user interface.

Figure 4 shows the Java user interface for displaying and modifying maps obtained by database queries or stored in files (URL queries). A thematic map is built by the overlay of several layers representing different themes (highways, rivers, equipments). The map shown in Figure 5 is the result of the overlay of two layers corresponding to the following two queries :

1. The first layer is the result of an OQL query accessing spatial data on national highways managed by the O_2 object-oriented DBMS :

```
select tuple(LEGEND: t.type,
             TEXT : "",
             ID : t.ID,
             GEOM : t.geom)
  from t in Roads
 where t.type = "Nationale"
```

2. The second layer corresponds to the following (extended) SQL query returning all equipments stored in a Postgres extended relational database :

```
select A.topo,
       A.class,
       A.id,
       G.type_spat,
       G.x, G.y
  from EQUIP A, EQUIP_spat G
 where A.id=G.id
```

The graphical Java client sends a query request (① and ② in Figure 3) to the mediator. The query is analyzed by the mediator which forwards the query to the Postgres database (`RDBMS`) implementing interface IDL 2. The wrapper sends an SQL request to the database layer (`RDBMS`, ⑤) and transforms the query result (⑥) according to the interface definition `IDL 2`. The result received by the mediator (⑦ and ⑧) is then sent back to the graphical user interface client (⑨ and ⑩).

[1] mSQL (Mini SQL) has been developed as part of the Minerva Network Management Environment and is freely available at the following address : `ftp://Bond.edu.au/pub/Minerva/msql`.

[2] VisiBroker, formerly ORBeline, is commercialized by Visigenic.

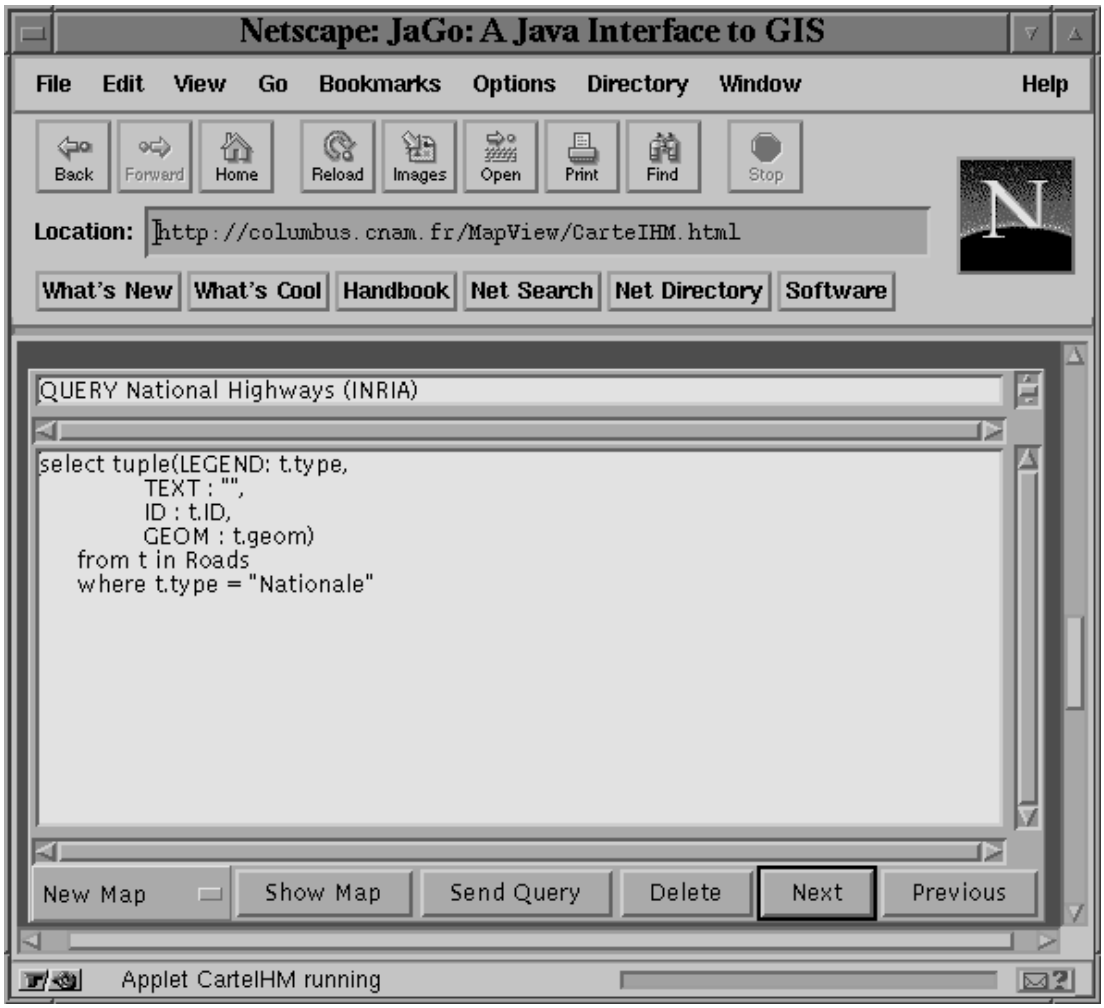

Figure 4: Java Graphical User Interface

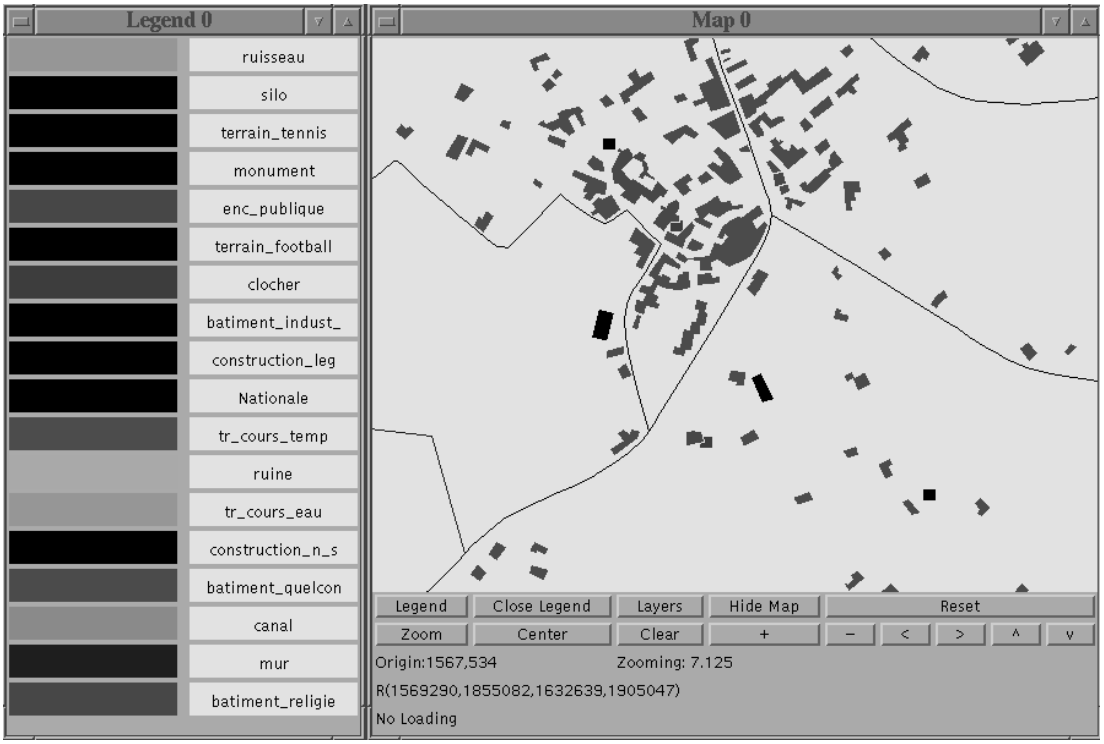

Figure 5: Thematic Map : Highways and Equipments

Observe that the location of each database is transparent to the clients. In the previous example, the O_2 database is situated in the domain of INRIA, Rocquencourt, and the Postgres database is installed in the domain of at CNAM, Paris. Each of these database can be moved transparently inside the ORB domain (CNAM and INRIA).

6 Related Work

Our prototype can be considered as a (partial) implementation of the services model in the The Open Geodata Interoperability Specification (OGIS) framework. OGIS includes three parts [4] :

- The Open Geodata Model (OGM) tries achieves "data interoperability" by an extensible information model describing real world space/time phenomena at different abstraction levels (real-world, specification, implementation).
- The Services Model is based on the client-server paradigm, where service providers return requested data or (processing) functionalities to client programs (objects).
- The Information Communities Model achieves "institutional interoperability" based on the data and service models and by defining a scheme for automated translation between different geographic feature lexicons.

The Open Architecture Scientific Information System (OASIS) [12] provides different services for the exploration (catalog service), integration (conquest parallel execution environment) and storage (geoPOM) of geo-scientific data. The object model is based on the OGIS framework and defines the behavior (interface hierarchy) and the structure (object hierarchy) of a wide range of geoscientific data. The prototype is developed on top of SunSoft's CORBA-compliant NEO

software. The system presented in [12] uses the same data model for mediators, catalogs (data sources) and catalog wrappers. In order to be more flexible and to integrate existing software components, it should be possible to implement different models (interfaces) as presented in Section 3.

A federated architecture for an environmental information system based on CORBA (Orbeline and Orbix) is presented in [10]. This architecture distinguishes between information services and data sources. The information services are based on the environmental data catalogue UDK [11] and help users to find relevant data sources. The prototype implementation uses the GRASS GIS software for the display of raster maps.

A big number of spatial data services are accessible on Internet. Some of them are based on JAVA applets implementing some simple operations such as zooming and layer selection[3].

7 Future Work and Conclusion

In the current version of our prototype, the main mediator functionality is query dispatching to different database clients. The presentation of geographic information in the form of a map is the result of a variety of operations (map overlay, scale, generalization) which can be executed at the different levels of this architecture [23]. For example, it is possible to execute the overlay of two maps (1) at the database level, if both maps are stored in the same database, (2) at the data access level, if the maps exist in different databases (3), and at the application level, if the maps are not accessed by the same request. We are currently working on the description of more sophisticated GIS mediator tasks, integrating different data sources and evaluating complex queries over different spatial data sets.

Our experience showed that CORBA is well adapted to the systems integration problem for several reasons :

- Performance : The additional communication cost introduced by the new software components (ORB core, mediator, wrapper) remains insignificant compared to the global response time (query evaluation and result construction).
- Location and hard- and software independence : CORBA allows the integration of new data servers and applications very easily. The location of a component is completely transparent to the other components. In our prototype, it is for example possible to execute the mediator and client components on different hosts and different domains (CNAM and INRIA) without any recompilation of the source code.
- Code independent: IDL interface definitions can be compiled into different languages (Java, C++), and it is possible to create, for example, Java clients for services implemented in C++ without any modification on the server side. The used ORB software (Oberline Version 1.0) did not provide a JAVA compiler for IDL definitions. In the future, we plan to use more recent ORB software in order to remove the CGI-client (Figure 3) which is used by the Java client for communicating with the mediator component.

References

[1] D. Abel, P. Kilby, and J. Davis. The systems integration problem. *Intl. Journal on Geographical Information Systems (IJGIS)*, 8(1):1–12, 1994.

[3] A list of existing Internet based products and prototypes can be found in the following Web page : `http://www.gis.umn.edu/rsgisinfo/interactive.html`.

[2] J. Ordille A.Y. Levy, A. Rajaraman. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22th International Conference on Very Large Data Bases*, pages 251–262, Bombay, India, September 1996. Morgan Kaufmann.

[3] F. Bancilhon, C. Delobel, and P. Kannelakis. *The O_2Book*. Morgan Kaufmann, 1992.

[4] K. Buehler and L. McKee, editors. *The OpenGIS Guide*. Number 96-001 in OGIS TC Document. OGIS Project Technical Committee, Open GIS Consortium, Inc., 1996.

[5] M. Carey, L. Haas, P. Schwarz, M. Arya, W. Cody, R. Fagin, M. Flickner, A. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J. Williams, and E. Wimmers. Towards heterogeneous multimedia information systems: The Garlic approach. In Ming-Chien Shan Omran A. Bukhres, M. Tamer Özsu, editor, *5th Int. Workshop on Research Issues in Data Engineering - Distributed Object Management (RIDE-DOM)*, pages 124–131, Taipei, Taiwan, March 1995.

[6] R.G. Cattel, editor. *The Object Database Standard : ODMG-93 : Release 1.1*. Morgan Kaufmann series in data management systems. Morgan Kaufmann, 1994.

[7] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *Proceedings of IPSJ Conference*, Tokyo, Japan, October 1994. TSIMMIS project: http://www-db.stanford.edu/tsimmis.

[8] PostModern Computing. *ORBeline User's Guide*. PostModern Computing Technologies, Inc., Mountain View, CA, US, 1994.

[9] D. Florescu, L. Rachid, and P. Valduriez. Using heterogeneous equivalences for query rewriting in multidatabase systems. In *Third International Conference on Cooperative Information Systems (CoopIS-95)*, Vienna, Austria, May 1995.

[10] A. Koschel, R. Kramer, R. Nikolai, W. Hagg, J. Wiesel, and H. Jacobs. A federation architecture for an environmental information system incorporating GIS, the World Wide Web, and Corba. In *Third International Conference on Integrating GIS and Environmental Modeling*, Santa Fe, New Mexico, January 1996.

[11] H. Lessing, W. Swoboda, and O. Günther. UDK : A European environmental data catalogue. In *Third International Conference on Integrating GIS and Environmental Modeling*, Santa Fe, New Mexico, January 1996.

[12] E. Mesrobian, R. Muntz, E. Shek, S. Nittel, M. LaRouche, and M. Kriguer. Oasis: An open architecture scientific information system. In *Sixth International Workshop on Research Issues in Data Engineering: Interoperability of Nontraditional Database Systems*, New Orleans, Louisiana, February 1996.

[13] T.J. Mowbray and R. Zahavi. *The Essential CORBA: Systems Integration Using Distributed Objects*. Wiley/OMG, 1995.

[14] Open Group. *Introduction to OSF DCE Release 1.1*, 1996. OSF PTR Publications 18581-9 or F101P ISBN 0-13-185810-6.

[15] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *Proceedings of the 22th International Conference on Very Large Data Bases*, pages 413–424, Bombay, India, September 1996. Morgan Kaufmann.

[16] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. Medmaker: A mediation system based on declarative specifications. In *Proceedings of the 12th International Conference on Data Engineering*, New Orleans, February 1996. TSIMMIS project: http://www-db.stanford.edu/tsimmis.

[17] J.P. Peloux and P. Rigaux. A loosely coupled interface to an object-oriented geographic database. In *Proc. Intl. Conf. on Spatial Information Theory (COSIT)*, 1995. The online version is available at ftp://ftp.cnam.fr/pub/CNAM/cedric/tech_reports/RRC-95-06.ps.gz.

[18] E. Pitoura, O.A. Bukhres, and A.K. Elmagarmid. Object orientation in multidatabase systems. *ACM Computing Surveys*, 27(2):141–195, June 1995.

[19] M. Scholl, A. Voisard, J.P. Peloux, L. Raynal, and P. Rigaux. *SGBD Géographiques - Spécificités*. International Thomson Publishing France, Paris, France, 1996.

[20] A.P. Sheth and J.A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(1):183–236, March 1990.

[21] M. Stonebraker and G. Kemnitz. The Postgres next generation database management system. *Communications of the ACM*, 34(10):78–92, October 1991.

[22] A. Tomasic, L. Raschid, and P. Valduriez. Scaling heterogeneous databases and the design of DISCO. Technical Report 2704, INRIA, November 1995. FTP: ftp://rodin.inria.fr/pub/publications/rapports/RR-2704.ps.gz.

[23] A. Voisard and H. Schweppe. A multilayer approach to the open GIS design problem. In *Proc. of the ACM GIS workshop*, December 1994.

[24] G. Wiederhold. Mediaton in information systems. *ACM Computing Surveys*, 27(2):265–267, June 1995.

A Mediator Interface Specification

```
interface Point
   ( extent points ) : persistent
   {
     attribute Long x ;
     attribute Long y ;
   };

interface Comp
   ( extent comps ) : persistent
   {
     attribute Enum Topo { point, line, polygon } topo ;
     relationship List<Point> points ;
   };

interface Geometry
   ( extent geoms ) : persistent
   {
     relationship Set<Comp> comps ;
     Geometry     clipping(in Geometry geom) ;
     Boolean      intersects(in Geometry geom) ;
   };

interface Road
   ( extent roads
     keys name ) : persistent
   {
     attribute String name ;
     attribute Enum Kind { street, highway, path } kind;
     relationship Geometry geom ;
   };

interface City
   ( extent cities
     keys zip ) : persistent
   {
     attribute String name ;
     attribute Long zip ;
     attribute Long population ;
     relationship Geometry geom ;
   };
```

B Wrapper Interface Specifications

B.1 Spatial Database Wrapper

```
interface GEOM
   ( extent GEOMS
     keys OID ) : persistent
   {
     attribute Long OID;
```

```
    GEOM clip(in GEOM geom);
    Boolean  inter(in GEOM geom);
  };

interface POINT : GEOM
  ( extent POINTS
    keys X,Y ) : persistent
  {
    attribute Long X;
    attribute Long Y;
    relationship Set<LINE> start_of;
    relationship Set<LINE> end_of;
  };

interface LINE : GEOM
  ( extent LINES ) : persistent
  {
    relationship List<POINT> points;
    relationship <POINT> starts_with inverse POINT::start_of;
    relationship <POINT> ends_with inverse POINT::end_of;
  };

interface POLYGON : GEOM
  ( extent POLYGONS ) : persistent
  {
    relationship List<POINT> points;
  };
```

B.2 Relational Database Wrapper

```
interface HIGHWAY
  ( extent HIGHWAYS
    keys OID ) : persistent
  {
    attribute Long OID ;
    attribute String NAME ;
  };

interface CITY
  ( extent CITIES
    keys OID ) : persistent
  {
    attribute Long OID ;
    attribute String NAME ;
    attribute Long  ZIP ;
    attribute Long  POPULATION ;
  };
```