Integrating spatial data with computing infrastructure and field application

by

Jiangyi Pan

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee
Leslie Miller, Major Professor
Sarah Nusser
Shashi Gadia

Iowa State University
Ames, Iowa
2004

Copyright © Jiangyi Pan, 2004. All rights reserved.
Graduate College
Iowa State University

This is to certify that the master's thesis of

Jiangyi Pan

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy
TABLE OF CONTENTS

ABSTRACT............................................................................................................................ iv

CHAPTER 1 INTRODUCTION AND BACKGROUND.................................................... 1
  1.1 Introduction..................................................................................................................... 1
  1.2 Data Integration Technology ..................................................................................... 2
  1.3 Wrapper Technology .................................................................................................. 4
  1.4 CORBA Technology................................................................................................... 5

CHAPTER 2 MODEL OF THE INFRASTRUCTURE ..................................................... 6
  2.1 Conceptual Model of Infrastructure ........................................................................ 6
    2.1.1 Field Application ................................................................................................ 7
    2.1.2 Infrastructure Based on Object-Oriented Views ............................................. 8
  2.2 More about Infrastructure ....................................................................................... 13

CHAPTER 3 DESIGN OF DATA ACCESS COMPONENT .......................................... 16
  3.1 Common Design of Data Access Component ....................................................... 16
  3.2 Communication Platforms ..................................................................................... 18

CHAPTER 4 DESIGN OF METADATA AND RULES............................................... 25
  4.1 Metadata and Rules ................................................................................................. 26
  4.2 Query Generation ................................................................................................. 29

CHAPTER 5 IMPLEMENTATION OF INFRASTRUCTURE .................................... 34
  5.1 Prototype of CORBA Client-server Model Using File System ......................... 34
  5.2 Prototype of Mobile Agent Model Using Oracle Spatial Database .................. 36
  5.3 Prototype of CORBA Client-server Model Using Oracle Spatial Database ....... 38

CHAPTER 6 CONCLUSIONS AND FUTURE WORK ........................................... 41
  6.1 Summary of Current Work ..................................................................................... 41
  6.2 Future Work ........................................................................................................... 41

REFERENCES...................................................................................................................... 43
ABSTRACT

A significant portion of government agencies’ activities require mobile data collection and analysis in the field. To accommodate such tasks, various data sources are involved to provide the supporting information. The data integration, especially spatial data integration in these applications turns out to be a big issue due to the multitude and heterogeneity of possible data sources. An aim of our research was to develop a flexible and extensible infrastructure to facilitate the field applications with integrated data sources. New designs have been proposed to treat heterogeneous data sources as set of object views. This object view approach helps to integrate multiple data sources into existent object-oriented view system. This research also goes a further step from the previous version of infrastructure in terms of extending data processing and communication capabilities. Oracle spatial database has been included as a new type of spatial data source. CORBA based client-server model is also implemented as another communication resort for the infrastructure to interact with the data access component.
1.1 Introduction

A wide array of federal, state and local agencies collect and analyze data to carry out their missions, evaluate programs and services, or provide information on characteristics of the population and the economy. A significant portion of these activities require data integration to seamlessly access various heterogeneous databases.

Data integration basically deals with the various information resources of heterogeneous natures. Although a great deal of work has been focused on integrating business data, limited efforts have been made to integrate geospatial data sources. Besides, our current infrastructure does not have a general approach to integrate multiple heterogeneous data sources. Some efforts need to be made to integrate spatial data source and to issue an overall integration approach.

As another important issue of data integration, query generation for a given data source is still absent in our current infrastructure. In this sense, the current mediator is a pseudo mediator instead of a real one. A considerable part of this thesis focuses on how to design and implement rule based query generation mechanism for a spatial data source.

In terms of communication platform, our current infrastructure only has mobile agent based scheme. This sole communication resort may not be able to handle all the possible cases. Since a lot of data sources involved in our infrastructure do not support mobile agent based access due to security concerns, some other communication scheme must be designed and implemented.

The contribution of this thesis lies in several aspects. First, improvements are made to integrate multiple data sources using object view paradigm. These improvements enhance the
infrastructure’s capability of solving heterogeneity of various data sources. Second, new designs have been proposed to turn the previous pseudo mediator into a real one. In order to provide a method to access spatial information automatically, rule based query generation mechanism was issued as a solution. By this way, mediator can handle the requests sent from users and figure out how to access the data without intervention of a human. Finally, CORBA based client-server communication platform is also implemented. As an extension of communication capability to the previous infrastructure, CORBA based communication can deal with cases that can not be otherwise handled by mobile agents in the previous infrastructure.

This thesis is organized as follows. After a brief overview on background and current approaches in the rest of Chapter 1, the design model and improvement of the infrastructure are given in Chapter 2. The design details of Data wrapper are discussed in Chapter 3. The design details of rules used for mediation are discussed in Chapter 4. Implementation of prototype system is presented in Chapter 5. Finally, Chapter 6 draws conclusions and proposes future works.

1.2 Data Integration Technology

Data integration is the problem of combining data residing at different sources, and providing the user with a unified view of these data. Since sources are in general autonomous status, in many real-world applications the problem arises from heterogeneous data sources. In practice, this problem is generally dealt with by means of suitable transformation and cleaning procedures applied to data retrieved from the sources.

Data integration technology is relatively mature when dealing with business data [1, 2, 3]. However, in terms of geo-spatial data, strategies and methodologies have not kept pace with advances in data collection. It remains difficult to analyze even two spatial data sets acquired at different times, for different purposes, using different datum and levels of in situ sampling
or enumeration precision. Without the ability to integrate data from different sources, we are faced with extensive duplication of effort and unnecessary cost.

Spatial data integration has been crucial to support field application. Field applications need to transfer attributes from old versions of feature geometry to new, more accurate versions; to the detection of changes by comparing images of an area from n different dates; or to automatic registration of one data set to another through the recognition of common features.

Some work has been done so far to integrate spatial data sources of heterogeneous natures. A CORBA based approach of data and system integration has been proposed in [4], which allows various applications to exchange data as objects. As a result, not only data but also data processing methods can be made accessible for the remote applications. However, interfaces with data sources were defined in the style of GeoToolKit, which might limit its usage for other applications. There was another approach as described in [5], it provided the support for integrating geospatial data from multiple sources, including Web. Mobile agents were used to do the integration, which provided autonomy and flexibility. However, the limitation of this method is that it integrates spatial data from multiple sources into a single database system for subsequent access and retrieval. There is an additional cost of managing such a database. Considering the vast amount and redundancies of data, this approach is not practical.

As an emergent trend [8], views have been involved into data integration process regarding multiple heterogeneous data sources. When using views, users of a data integration system pose queries against a mediated schema, rather than the schemas in which the data is actually stored. The system catalog includes a set of source descriptions that provide semantic mappings between the relations in the mediated schema and those in the data sources.

Object-Oriented view is usually used to describe the characteristics of interfaces and system internals based on an extensible object model (EOM). The view object type is
defined as being an extension of the object model (EOM). In this paper, the view definition is an extension of work in [3, 6, 7]. Current view type has been extended to include public attributes, private attributes, a derivation section, and a method section. The public attributes, private attributes, and method sections have the normal meanings. The derivations section is used to generate the public and private attributes of each object instance created through a view.

1.3 Wrapper Technology

In the domain of computer science, a wrapper is a program that acts as an interface between a caller requesting an action and a wrapped component that is needed to execute the action. The purpose of wrapper is to create interoperability between the caller and the wrapped component, which may exist in environments that differ sufficiently to require some kind of translation between the caller and the invoked component. Usually, the wrapper encapsulates the low level details of the wrapped component and provides high level information about the wrapped component.

The model of a wrapper based system is shown in Figure 1-1. A caller’s request to the wrapped component is translated by the wrapper into the format of the wrapped component. Results returned from the component are translated via the wrapper into the caller’s format.

![Figure 1-1. Model of wrapper based system](image_url)
In general, a wrapper provides an interface between the user or caller's environment and the wrapped component. By the translation of a wrapper, heterogeneity of different computing platforms can be overcome.

1.4 CORBA Technology

CORBA (Common Object Request Broker Architecture) was developed by the Object Management Group [12]. The goal behind the development of CORBA was to enable open interconnection of a wide variety of languages, implementations, and platforms.

Software in CORBA environments are performed at two sides: client-side and server-side. A server-side application developer implements object implementations for services. After that, it describes the interfaces to the provided services with IDL (Interface Definition Language). A client-side application developer makes reference to the IDL written by a server-side developer, and implements client applications.

CORBA-based approach is currently playing an active role in data and systems integration. It allows for various applications to exchange data as object and makes data processing methods accessible for remote application [4]. It also helps to wrap components of the existing systems and to invoke them from the object-oriented distributed environment [9]. New generation of geographic information systems (GIS) also apply CORBA to achieve an open design [10], which provides an open architecture, interoperability and extensibility.

Some CORBA based products have already been developed to deal with infrastructure based on heterogeneous systems. HORB [11] is a Java ORB (Object Request Broker) that extends Java for distributed object oriented computing. It provides remote object creation as well as connection, remote method call, object passing, security and so on.
CHAPTER 2 MODEL OF THE INFRASTRUCTURE

First, a conceptual model of infrastructure will be described here as our existent work. Details about the infrastructure, some of which are improvements made toward the current infrastructure are then discussed so as to provide a further vision of how different components work coordinately to support field applications of data collection and analysis.

2.1 Conceptual Model of Infrastructure

Conceptual model of the infrastructure is shown in Figure 2-1. Next, we will briefly introduce components of the infrastructure respectively.

Figure 2-1. Conceptual model of infrastructure
2.1.1 Field Application

A field application is a field task carried out by certain field representatives in some specific field situation. Usually, we envision a field application as being combined with the field environment. Field environment can be characterized as having three components: the data collection task, the user’s characteristics and the field computing resources. The data collection process is divided into three phases: planning, navigation and data collection per se. Each of these tasks drives the type of geospatial data needed in the field. The user’s characteristics include the assumed level of exposure to geospatial data (high for a forest or soil scientist, low for a decennial census enumerator), spatial strategy (route-based, map-based), and physical attributes that have to do with limitations on interaction modes and speed of travel. Computing resources are described by capacity to download, manipulate and store spatial data on the client computer and characteristics of the interface that impact the data product delivered to the user. The way in which these factors impact infrastructure decisions can be thought of as determining the appropriate format to transfer and manipulate geospatial data.

A field worker is a field representative with specific data collection skills such as interviewing, pricing, or biological data collection. These skills are of primary importance in collecting high quality data.

Handheld is a certain type of field computing device, which is a part of the mobile computing environment. In most mobile settings, the field computing device is extremely limited in computing capacity, bandwidth and display interfaces relative to the large volumes of information that exist in distributed geospatial data repositories.

In a field task, a field worker may require access to information resources provided by different agencies. Instead of sending a request directly to the data source, which is the typical case in the Client-Server model, the field computing device will send the request to the field wrapper.
2.1.2 Infrastructure Based on Object-Oriented Views

When the field worker is in place, the question of how to provide him the access to various geospatial data sources becomes the major concern. To do this, we should have an overall perspective of the infrastructure. Based on such a mechanism, flexibility and extensibility of the data exchange through the network must be achieved. To fulfill this requirement, we have chosen to make use of object-oriented views, which provide an excellent basis for deriving the data for the user's request and creates a great deal of flexibility in the location for integrating or analyzing data.

There are three levels of views that are used to fetch data from the operational data sources: Local Interface View, Base View and Global View.

The local interface view is a view type object that is used by the local data administrator to provide a mechanism to make the local data accessible to integrated environment. The local interface allows distribution transparency and representation transparency, while hiding or converting (mapping) some of the data from the data source. The local interface belongs to the local data source. In interacts directly with the data source and passes the result to the wrapper which controls communication with external users.

As described above, Local Interface View is provided by the data source, which represents the accessible characteristics of the data source; Base View has a lot to do with the operations expected either by user or system; Global View is a view at the highest level. Global Views can be derived based on any number of local interfaces, base views or other global views by applying composition algorithm.

The remaining components of the infrastructure are actually constructed based on the object-oriented view system.

- Field Wrapper

Field wrapper is an extension of existing wrapper concepts to support mobile computing environments. The particular application we consider is data collection that involves the use
of geospatial data, although these concepts apply more broadly. The field wrapper model we propose is shown in Figure 2-1.

![Figure 2-1. Model of field wrapper involved system](image)

As in Figure 2-1, field user’s request is translated by the wrapper into the language that could be understood by the next infrastructure component. Meanwhile, user’s request is reformatted and cast into a global view, which stands for user’s demands for data and operation. The infrastructure component that connected to the infrastructure then takes subsequent action. Again, results returned from the infrastructure are translated back via the field wrapper into the pre-defined format.

The field wrapper must take the field view as global view and generate information for the target infrastructure entity. To achieve this, efforts are required to parameterize field environment. Certainly, the data request initiated by field task is also described by some sort of parameters.

- **Data Wrapper**

A data wrapper is a wrapper that encapsulates the data sources. It localizes the details associated with heterogeneity in data sources. Usually, a data wrapper plays the role of translator between the infrastructure that asks for information and the local interface view through which that information could be accessed. A data wrapper receives a request sent by the computation server. The request may be derived from certain number of global views or
base views. The data wrapper then rewrites the sub-requests contained in those views to queries understandable to the local interface view. The model of the data wrapper in such a data exchange process is depicted in Figure 2-2.

![Figure 2-2. Model of data wrapper in the system](image)

As shown in Figure 2-5, a data wrapper may also execute some data operations initially contained in certain views other than local interface views. Hence, the functionality of data wrapper is more than information retrieval.

- **Mediator**

A mediator is a class of software modules that mediate between the field device/wrapper and the data sources. It uses supporting information such as metadata and object-oriented views to generate a query string according to the rules.

As whole, a mediator looks like an information processing center. Upon receiving a user request from the field wrapper, the mediator finds the locations of data sources, generates source specific queries as well as operation indicators over retrieved data. Then it sends the request to the next component, the computation server. The internal structure of mediator is shown in Figure 2-3.
Basically, there are two kinds of parameters corresponding to data requests sent from field wrapper. One is symbolic, it is in text format. For example, use “Iowa” to retrieve the map of Iowa State. The other is numeric, which characterizes the data request by providing numeric parameters. For example, use longitude and latitude pair, such as (119 W, 40 N), to get a map of the specified position.

If the data request from field wrapper is symbolic, the mediation process will go downward by taking the left part of the triangle (refer to Figure 2-3). That is, an ontology based index will help the request to find a group of candidate data sources. Then, rule set A is applied to determine which data source to access. If the data request is location specific, it will use the right part of the mediator. The process contains using R-tree based index first to locate the data sources that may contain relevant spatial information and then apply rule set A to pick one from those candidate data sources. Now that the mediator knows where to get information, it focuses on how to get them. Rule set B takes charge at this time to generate source specific query strings.
At last, the data request translated by the mediator will have an output format of $O_A(S_1, O_B(S_2, S_3))$. In this mediator generated string, $O_A$ and $O_B$ stand for operations that need to be executed, for example, clip, raster_vector_fusion; $S_1$, $S_2$ and $S_3$ are source specific query strings, such as “Select A.shape from Vector_Shape A where A.name='soil'” if the data source is a relational database and shape data is stored in a table named Vector_Shape.

- **Computation Server**

  Computation servers are used to provide computation that can not be handled by either the field wrapper or the source wrapper. The primary example is processing data that comes from more than one source. A simple example is a conflation algorithm used on images from different sources.

  To support the necessary computation, a computation server can either execute the methods of the view object and/or make use of the tools supported by an object-oriented data warehouse that exists at most computation servers. The data warehouse sees the results of the data coming back from the data source wrapper as materialized views. The advantage of using the data warehouse to store and operate on the data is that the data warehouse can be used as an active cache. Based on our preliminary analysis of user needs, it looks likely that the user will need to use the same (or very similar) data again after the initial query. For example, for a small device the image used on the first request will only be a subset of a larger image (retrieved or constructed) as the field worker moves around the field he/she will need to have the image adjusted. In many cases the data in the data warehouse can be used without additional retrieval. The warehouse is viewed as an active cache, since the tool set (or view object methods) can be used to modify the data as needed by the user.

  The extra computational support is also critical to allow the environment to combine data from multiple data sites and to create an efficient environment.
2.2 More about Infrastructure

In this section, we will discuss some more details about the infrastructure and explain how it works.

- How data wrapper manages information from local data source

To give a clear description, here we provide an example. Suppose there are three data sources available. One is geometry & shape data stored in a relational database (e.g. Oracle Spatial) as relation (table) named Road_Shape, the schema of the table is shown in Table 2-1; the second is normal file system that stores images of maps; the third one is legends of a specific map's layer retrievable through Web when providing image name and layer name of the map. For the sake of simplicity, we name these three data sources as A, B and C respectively.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>GID</td>
<td>Integer</td>
</tr>
<tr>
<td>Shape</td>
<td>Geometry</td>
</tr>
</tbody>
</table>

Table 2-1. Schema of table Road_Shape

Source A contains vector data corresponding to road layer of a map; it provides Oracle extended SQL as query interface. Images of maps stored in source B could be retrieved by providing name of the image file; the image size could also be retrieved by providing the image file name. The web query interface of source C requires two parameters as input, one is image name and the other is layer name. Then, these two parameters would be passed to a CGI program to retrieve corresponding legends.

Information contained in these three heterogeneous data sources could be transformed into object views by using object definition language [13]. As shown in Figure 2-4, the definitions strictly stick to what could be retrieved from local data source and the way of accessing them.
Keyword “attribute” is used to indicate data visible through interface; keyword “derivation” indicates the method used to get all attributes. They should both have public property as refer to object oriented paradigm. Methods defined as “get_size”, “get_image” and “get_legends” are used to derive part of the attributes in the corresponding interface.

These definitions could also be easily collected and transformed to local interface views that are defined in the underlying object-oriented view system we have described previously. The advantages of defining them this way is that the wrappers could manage local data sources more conveniently and directly. Also, it helps mediator to uniquely identify the information source, which is important to do the mediation. We will address this issue in the next section.
• How mediator locate information sources

Given the method of defining information sources into object views, we should also be able to identify a specific information source by referring to the defined object. To achieve this goal, we must solve naming space conflict. We know that inside a specific data source, there should be no naming space conflict. For example, in a relational database, relation name is identical, which could differentiate one relation from another. Since there are multiple data sources, additional steps must be taken to avoid naming conflict.

We devise a naming mechanism here, which assigns a unique ID to a defined object view. This ID is called object ID. Object ID is separated into two parts, one is the data wrapper ID, which is assigned by mediator and guaranteed to be identical; the other is the key of the information source defined as interface, which is within the scope of a specific data wrapper. We could use the internal identical name of the information resource as the key. For example, the relational repository is corresponding to wrapper1, and internal name of the object view is Road_Shape. When we combined them together, the object view presents itself as a pair (wrapper1, Road_Shape). It's an unambiguous description of the information resource corresponding to relation Road_Shape. Further, if attribute GID is also included, we will have a unique reference to a tuple in the relation. Another example, the Object ID for the web Legends interface could be (wrapper2, URL of web interface), again first part determined by wrapper ID and the second part by URL of web interface. This URL could be “http://www.legends.com/query.exe?”, “query.exe” stands for the CGI program used to retrieve legends information. Again, if combined with more parameters, which makes the URL into “http://www.legends.com/query.exe?Ames&Road”, the legends of road layer of map Ames would be located without ambiguity.
CHAPTER 3 DESIGN OF DATA ACCESS COMPONENT

In Chapter 2, we briefly described the overall model of the infrastructure. Inside the infrastructure, a data wrapper is responsible for processing requests from the computation server and then retrieving data from local data sources. The data wrapper may also do some operations on the retrieved data if required by the computing infrastructure.

When dealing with the data exchange and transformation, data wrapper has to interact with the local interface view of local data source. Local interface view is actually a reachable object view provided by the data source. For example, if data source is a relational database and owns several relations as information sources, each relation is actually defined as an object view. Each of these object views could be retrieved by its corresponding local interface view. According to the discussion in Chapter 2, we know that an object view could be differentiated by the name of the relation and thus be retrieved if provided with the name of the relation as input. Here, we call data wrapper and local interface view as data access component of the infrastructure. In this chapter, we will address the design issues regarding data access component. Besides, we will also design the communication platforms that deal with the interaction between data access component and computing infrastructure.

3.1 Common Design of Data Access Component

The design model of data access component is shown in Figure 3-1. Inside the data wrapper, there is a processing module. Its main function is carrying out data retrieval and manipulation tasks required by the computing infrastructure. It also communicates with entities on both sides of the data wrapper. A data wrapper has two communication interfaces,
one is to the computing infrastructure and the other is to the data source. Numbers on the arrows stand for different data flows. We list these data flows as follows:

I. Data request from computing infrastructure to data wrapper
II. Translated data request from data wrapper to local interface view
III. Data request inside data source sent by local interface view to local data source
IV. Retrieved data from local data source to local interface view
V. Data output by local interface view to data wrapper
VI. Final data sent back from data wrapper to computing infrastructure

Figure 3-1 Design model of data wrapper

Note, data flows I and VI follow the communication protocol of infrastructure, that is, they eventually have something to do with some certain views, e.g. global view or base view. II and V have the data exchange format mostly determined by local interface view, for example, JDBC for most of relational databases. III and IV are inner data flows of data source. For example, for relational data base, III could be SQL statements or stored procedures and IV could be DBMS generated result set. Obviously, II, III, IV and V are less
flexible and relatively easy to deal with. However, things are different when considering I and VI.

Among all these data flows, I and VI are the most crucial part and may vary due to overall concerns such as availability, performance and security. For example, data wrapper may be unable to execute customized operation on retrieved data, which is required by a certain object-oriented view. In next section, we will focus on different designs regarding I and VI as well as the reasons to make the choices.

3.2 Communication Platforms

Basically, there are two kinds of communication platforms. One is agent based, the other is client-server based. Both of them can effectively solve heterogeneity. Before going into design details, we first introduce some necessary discussion about mobile agent, CORBA based client-server model and their relationships with object-oriented view.

An agent is a computer program that performs a set of tasks on behalf of a user with some degree of autonomy [14, 15, 16, 17]. An agent consists of program code, a persistent internal state, and a set of attributes (e.g. movement history). Mobile agents are agents that can move in a computer network from host to host as needed.

As we have mentioned before, object-oriented view system plays a very important role in the infrastructure in the sense both conceptually and practically. An object-oriented view is made up of three parts: data attributes description, data derivation section and methods section.

There is a natural link between an object-oriented view and an agent. For example, the attributes of a view could be transformed to the attributes of the view agent; the derivation section of a view could be used as the agent "program" to extract only the data needed. Hence, views could be combined with mobile agents to facilitate extracting the required data from the data sources. Moreover, the object-oriented view system could also be incorporated
into the mobile search agents to provide the basis of transforming heterogeneous data types to provide a consistent format for the infrastructure.

Mobile agent based communication platform are used in many places of the infrastructure. The mobile agent is regarded as a medium or messenger. For example, requests from field wrapper to mediator are packed in mobile agents. After the mediator has analyzed the request, it will create an instance of the object view by generating a new mobile agent. Next, the spawned agent is caught by computation server and gets analyzed. The computation server then sends requests to data wrapper and waits for the relevant data. After processing the data, it launches a result agent and sends back to the field wrapper.

The CORBA based client-server model is a method calling mechanism similar to RPC (Remote Procedure Call). Nevertheless, CORBA is more powerful than RPC. A CORBA supported object can invoke another such object without knowing where the peer objects resides or in what language the requested objects are implemented. We depict the working CORBA based client-server model in Figure 3-2.

![Figure 3-2. Working model of CORBA client-server method call](image)

The diagram in Figure 3-2 shows a method request sent from a client to a CORBA object implementation in a server. A client is any code (perhaps itself a CORBA object) that
invokes a method on a CORBA object. The servant is an instance of the object implementation - the actual code and data that implements the CORBA object.

The client of a CORBA object has an object reference for the object and the client uses this object reference to issue method requests. If the server object is remote, the object reference points to a stub function, which uses the ORB machinery to forward invocations to the server object. The stub code uses the ORB to identify the machine that runs the server object and asks that machine's ORB for a connection to the object's server. When the stub code has the connection, it sends the object reference and parameters to the skeleton code linked to the destination object's implementation. The skeleton code transforms the call and parameters into the required implementation-specific format and calls the object. Any results or exceptions are returned along the same path.

The client has no knowledge of the CORBA object's location, implementation details, nor which ORB is used to access the object. Different ORBs communicate via the OMG-specified Internet InterORB Protocol (IIOP).

A client may only invoke methods that are specified in the CORBA object's interface. A CORBA object's interface is defined using the OMG Interface Definition Language (IDL). An interface defines an object type and specifies a set of named methods and parameters, as well as the exception types that these methods may return. An IDL compiler translates the CORBA object definitions into a specific programming language according to the appropriate OMG language mapping.

This CORBA client-server model has a strict limitation on the behaviors of client object since only the methods defined in server's interface are accessible. Moreover, in terms of communications between computing infrastructure and data wrapper, CORBA client-server has a more direct impact on local interface view since the client object regards its peer as an interface. Hence, a data wrapper could easily handle the request by forwarding it to a local interface view.
Now that we have overviewed the two choices for communication platforms, we will briefly discuss the relative advantages and disadvantages of these platforms, which help to explain why in different situations one may be more appropriate than the other.

The mobile agent based communication platforms have several advantages. They are listed as follows:

- **Autonomy**

  A mobile agent has everything it needs to carry out a specific task. After it’s spawned, it will go to the destination and do the job independently.

- **Flexibility**

  Since a mobile agent is itself an object, an object-oriented view can be easily embedded into a view agent. A mobile agent can also choose to carry just part of an existing object view or to carry multiple object views according to different working scenarios. It’s flexible to determine the overload of a mobile agent and distribute work between computing infrastructure and data wrappers.

- **Better performance when guaranteed availability**

  According to [16, 17], mobile agent based communication platform has a better performance since it requires less bandwidth. Moreover, due to its capability of executing complex tasks at the data wrapper, the overall network traffic between computing infrastructure and data wrapper could also be reduced.

  However, mobile agent based communication platform also has some constraints. First, the mobile agent bears its own code that will be executed without notifying the data wrapper, which might lead to some security problem. For example, hackers may intercept the agent from network and modify it by adding some malicious code, then sends to data wrapper. This compromised agent may be dangerous to the host system where the data wrapper resides. Second, mobile agents are capable of executing methods at the data wrapper side. If the methods are too complicated and resource consuming, it may cause the performance of data wrapper to drop down dramatically. Third, a mobile agent may do a complex job
independently. During the execution, the computing infrastructure is unaware of current status and has no control. In some cases, the computing infrastructure would rather do the task step by step by calling method at the data wrapper side and know what’s going to those steps.

The CORBA based client-server model could help to achieve the goals that can’t be otherwise made by mobile agent based communication platform. The CORBA client-server model has following advantages:

- **Security**
  Data wrapper may have a strict security policy that only authorized operations can be executed. Since CORBA based client-server model can only call the methods that are provided by the data wrapper, the security requirement could be easily met.

- **Feasibility**
  The client object regards the server object as an accessible interface. This makes the work of data wrapper easier. The data wrapper simply provides methods which forward the data request to the local interface view and send back the result.

- **Better control & response**
  When applying the CORBA based client-server model, the computing infrastructure is responsible to divide the work into steps. For example, the computing infrastructure may divide a complex data request into several simple sub-requests, then sends out those data requests one by one. By working this way, the computing infrastructure has a better control and the interactions are similar to the behaviors of those “stateful” communication protocols, which are usually more reliable.

  Since the client-server model is connection oriented, which means one interaction (data request, result return) completes within one connection, the response is also quicker when compared with mobile agent based communication.
On the other hand, there are also disadvantages of using client-server model. One is that more bandwidth is occupied. The other is the overhead of more interactions that occur when serving the data requests from a certain object-oriented view.

As discussed above, the two communication platforms both have advantages and disadvantages. Our design supports dual communication mechanisms to let the computing infrastructure decide which one to use according to different working scenarios. The system design is shown in Figure 3-3.

![Figure 3-3. Design model with dual communication support](image)

As shown in Figure 3-3, two data wrappers are involved to provide dual communication. Meanwhile, local interface view remains intact. The computation server can choose either one as communication platform to carry out its tasks.

Note, there is a difference between mobile agent based communication and the client-server based one. We use two lines to describe the interaction between mobile agent based CS wrapper and mobile agent based data wrapper. Arrow A stands for CS wrapper spawns a
query agent; Arrow B stands for data wrapper sends back a result agent. Double arrow C indicates that only one connection is actually needed to serve one data request.
CHAPTER 4 DESIGN OF METADATA AND RULES

As mentioned in Chapter 2, the mediator is crucial in the whole infrastructure. The mediator will handle all data requests from field applications and determine what actions must be taken to serve these requests. Generally, the mediator is responsible to determine where is the information source, what queries to make against the specific data source and what operations need to be executed before sending result back to the field device. When the mediation process is completed, mediator should spawn view agents to computation server and let the computation server do the job. During the course of mediation, metadata and rules are heavily relied on since they are the “knowledge” that the mediator has to support its decision making and query generation. In this chapter, we will discuss how to design metadata and rules as well as how to generate query strings.

![Work flow of mediator](image)

Figure 4-1. Work flow of mediator
Before we can go into the design details of metadata and rules, we must first have a clear understanding of the mediator’s work. To address this issue, we give the work flow of mediator in Figure 4-1.

4.1 Metadata and Rules

During the processing of the mediator, metadata is involved as important supporting information. Metadata is the data about data. In our infrastructure, the mediator uses metadata as the knowledge about the data sources. In terms of the second phase of mediation process, the metadata are divided into two categories by the mediator. The first category is “What the data source has?”. We describe the main part of this type of metadata as follows:

- All object views of the data source
- Details inside the object views, such as data type, key attributes
- Functions and operators owned by the data source
- Metadata due to data’s nature

As listed above, functions and operators owned by a data source is important in dividing work between operation part and query part of the final string. We will always try to put more operations into the query part since we can use the data source more sufficiently and reduce the future cost of computation and network transmission. Metadata due to data’s nature are the properties introduced by some particular data types. For example, spatial data has some special properties like coordinate system, tolerance etc.

The second category covers “How to access the data source?”. Again, important parts are listed below:

- Query template and syntax
- Access privileges

Query template is a common query model complied by all queries toward one data source. For example, the oracle spatial database uses the query model “Select ... From... Where...”. 
The web interface may have the query template "URL? keyword". The access privileges are also important if a data source grants privileges regarding certain data objects. For example, most commercial database systems including Oracle have such capability. Basically, metadata about privileges should know "Who has what privileges on which data object".

After all metadata are ready, the mediator should apply rules in rule set B to generate query strings. In this section, we will briefly discuss how these rules are organized. In the next section we will describe how the mediator generates query strings by applying these rules.

The rules that are involved in generating a query string against a specific data source could be put into following categories:

- Rules for privileges checking

As in the first phase of mediation, the mediator has already known what object views are involved. At the beginning of the second phase, the mediator will check if the field user has enough privileges on those object views. For example, followings are the typical rules.

i) If user doesn’t exist then return message "invalid user, access denied"
ii) If user is valid but not enough privileges then return message "insufficient privileges"

- Rules for generating query template

We combine a certain query template with a specific data source. The query template will first be located before generating the actual query string. For example, following are the two rules that can be applied to oracle spatial database and web data repository respectively.

i) If data source name='Oracle Spatial', then query template = 'Select <sec>A</sec> From <sec>B</sec> Where <sec>C</sec>,'

ii) If data source name='web', then query template = 'URL? keyword=<sec>A</sec>'

Here, tag pair <sec> and </sec> are used to indicate the part that needs to be filled according to partitioned data request.
• Rules for dividing the data request

The request could be divided into several parts according to the structure of query template. For example, if the query template is “Select … From… Where”, we may divide the data request into three parts.

• Rules for composing the query string

This part is about “filling” the query template with the divided data request. In terms of Oracle Spatial database, rules for composing the query string can be divided further into sub-categories.

As the first sub-category, rules in it will refine the query template and generate a more specific template. For example, part A of the query template of Oracle Spatial query may involve with some spatial functions, which means the rules applied here should determine if functions should be used and what functions to use. Similarly, for part C of the query template, rules may be applied to decide issues regarding operators. Following samples present some typical rules in this category.

i) If operation involved then put function into A

ii) If operation name='Intersect' then use function SDO_GEOM.SDO_INTERSECTION()

iii) If condition is spatial dependent then put operator into C

iv) If relationship =’touch’ then use operator SDO_RELATE().

In the second sub-category, rules here will deal with the issues of formulating the query given the detailed template. For example, if a function is present in the template, there should be rules that know how to fill in parameters. Also, there may be connections between different parts of the template. For instance, the variables declared in part B of query template may be referred in both part A and part C. Rules here should handle the interactions between those related parts. Following are the typical rules from this category:
i) If function = 'SDO_GEOM.SDO_INTERSECTION' and two parameters are involved then use SDO_GEOM.SDO_INTERSECTION (obj1.Geometry, obj2.Geometry, tolerance)

ii) If tables are involved then convert part B into 'From table name obji (1...n),' till all tables are included.

4.2 Query Generation

In the second phase of mediation, rule set B and metadata will be used to generate query strings. The important step of this process is to generate source specific query string. We depict the step in Figure 4-3.

As shown in Figure 4-3, the process of generating source specific query strings can be divided into four steps. As the first step, the mediator does privileges checking. If the field user doesn't have enough privileges on those object views, a notice would be sent to field user to report this problem and quit the mediation process. In the second step, a query template of a certain data source would be generated. Then, the third step will divide the data request from the field application into pieces such that it could fit in the query template. The fourth step of this query generation process is by far the hardest one since it combines the result from step 2 and step 3 and generate the exact query. When executing the fourth step, the mediator must apply the rules which are derived from the query syntax of that specific data source.

After the partitioning of data request in the third step, the output will be provided in XML format. The reason of putting it into XML format is due to the feasibility, extensibility and flexibility of XML. In order to give more details about this query generation process, especially for spatial queries, we will show how the mediator generates query string for oracle spatial database.
We already know that the query template of oracle spatial database is “Select A___From B___Where C___ ”. A could be any of the following or the arbitrary combination of them:

- Simple attributes
- Geometry objects
- Spatial functions operated on selected geometry objects

Simple attributes are attributes in a relation that have nothing to do with geometry objects. They could be regarded as the data that can be retrieved by plain SQL statement. Spatial functions are provided by oracle spatial database to operate on geometry objects. These functions help to reduce the future computation and transmission costs of the infrastructure.
Table 4-1. Some spatial functions from oracle spatial database

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDO_GEOM.SDO_INTERSECTION</td>
<td>Returns a geometry object that is the topological intersection (AND operation) of two geometry objects.</td>
</tr>
<tr>
<td>SDO_GEOM.SDO_XOR</td>
<td>Returns a geometry object that is the topological symmetric difference (XOR operation) of two geometry objects.</td>
</tr>
<tr>
<td>SDO_GEOM.SDO_DIFFERENCE</td>
<td>Returns a geometry object that is the topological difference (MINUS operation) of two geometry objects.</td>
</tr>
<tr>
<td>SDO_GEOM.WITHIN_DISTANCE</td>
<td>Determines if two geometries are within a specified Euclidean distance from one another.</td>
</tr>
</tbody>
</table>

Some important spatial functions [18] are shown in Table 4-1.

B could be any of the following:

- One table name corresponding to an object view
- Multiple table names corresponding to a set of object views

The involved object views above are chosen by the first phase of mediation process. If multiple object views are present, the corresponding tables need to be joined to get the result.

C could be either of the following:

- Normal conditions
- Operator involved condition

Normal conditions are those normal SQL understandable conditions. In oracle spatial database, some spatial operators may be used to determine the relations between geometry objects by returning boolean result. All operators [18] that may be used are listed in Table 4-2.
Table 4-2. Spatial operators from oracle spatial database

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDO_FILTER</td>
<td>Specifies which geometries may interact with a given geometry.</td>
</tr>
<tr>
<td>SDO_NN</td>
<td>Determines the nearest neighbor geometries to a given geometry.</td>
</tr>
<tr>
<td>SDO_RELATE</td>
<td>Determines whether or not two geometries interact in a specified way.</td>
</tr>
<tr>
<td>SDO_WITHIN_DISTANCE</td>
<td>Determines if two geometries are within a specified distance from one another.</td>
</tr>
</tbody>
</table>

Now let's focus on IV, the most crucial step of the query generation process as shown in Figure 4-3. After Step II and Step III, the output is given as an XML wrapped up text. In order to describe the process more specifically, let's imagine a real application scenario. A field user wants to see the part of a road inside a park. Suppose the name of the park is 'rose' and it's stored in a table named 'parcel'; the name of road is 'R38' and it's stored in a table named 'roads'. After the processing of Step III, the divided request is presented as shown in Figure 4-4.

From Step II, the query template “Select A from B Where C” has already been generated. From this point, rules will be applied to generate more detailed template and fill the template with variables and parameters. We describe the process as below:

i) Apply rule: If <oper> in <what> section then put function into section A

ii) Apply rule : If <oper=Intersect> then use function SDO_GEOM.SDO_INTERSECTION()

iii) Apply rule: If two parameters involved then use SDO_GEOM.SDO_INTERSECTION (obj1.Geometry,obj2.Geometry,tolerance)

iv) Apply rule: If <from> section contains only <table> tags then seg2=seg2+table.attribute+' '+table.value till all tables included
v) Apply rule: If <condition> section contains only <unit> and <bool> then
seg2=seg2+unit.attribute+'. '+unit.value+ bool.value till all units and bools included

After the execution of iii), the template would be “Select SDO_GEOM.SDO_INTERSECTION (obj1.Geometry, obj2.Geometry, tolerance) From seg2 Where seg3”.

```xml
<what>
    <oper=Intersect>
        <obj>obj1</obj><obj>obj2</obj>
    </oper>
</what>
</from>
<table=roads>obj1</table>
<table=parcel>obj2</table>
<condition>
    <unit=objc>road_name='R38'</unit>
    <bool>and</bool>
    <unit=objc>parcel_name='rose'</unit>
</condition>
```

Figure 4-4. Divided request in XML format

After the execution of iii), the template would be “Select SDO_GEOM.SDO_INTERSECTION (obj1.Geometry, obj2.Geometry, tolerance) From seg2 Where seg3”.

After iv), the template is rewritten into “Select SDO_GEOM.SDO_INTERSECTION (obj1.Geometry, obj2.Geometry, tolerance) From roads obj1, parcel obj2 Where seg3”.

Finally, the output will be “Select SDO_GEOM.SDO_INTERSECTION (obj1.Geometry, obj2.Geometry, tolerance) From roads obj1, parcel obj2 Where obj1.road_name='R38' and obj2.parcel_name='rose'”.
CHAPTER 5 IMPLEMENTATION OF INFRASTRUCTURE

The implementation so far is focused on the data access component of the infrastructure. In the previous version of the infrastructure, a simplified data wrapper has been implemented to retrieve data from local file system. In this chapter, we will discuss the implementation of the data wrapper and the local interface view under the design of Chapter 3.

Both communication platforms, mobile agent based and CORBA client-server based, are implemented to support the interaction between the computing infrastructure and the data access component.

The local data sources that are involved include oracle spatial database and local file system. Certainly, there should be four implementations due to different data sources and different communication platforms. We will describe three implemented prototypes here. One is the local file system using CORBA client-server. The second is oracle spatial database based on mobile agents. The third is oracle spatial database using CORBA client-server.

5.1 Prototype of CORBA Client-server Model Using File System

The prototype of CORBA client-server based implementation is shown in Figure 5-1.
This prototype is implemented by HORB, which is a Java based CORBA application. At the client side, the server proxy has the same function as the stub in a standard CORBA architecture. More details are given below:

- **HCSWrapper**

  HCSWrapper is the client object residing at the computation server. It sends data requests by calling the methods provided by the server object. The dotted lines stand for virtual connection between the client and the server. The solid lines represent the physical communication path.

- **HCNRIWrapper**

  HCNRIWrapper is the sever object working as a data wrapper. It implements three methods to provide data access toward the local data source. HCSWrapper will call Get_File to get an image file and Get_WorldParameter to get the parameters in a world file. Before Get_WorldParameter is called, Get_WorldFile must be called to fetch the corresponding world file from local data source.

- **File Access Interface**
File Access Interface is one of the possible cases of local interface view. In this prototype, the translation between data wrapper and local interface view is straightforward due to the properties of CORBA client-server architecture. Hence, for the sake of simplicity, there is no additional method in the file access interface.

5.2 Prototype of Mobile Agent Model Using Oracle Spatial Database

The mobile agent based model is more complex than CORBA based client-server model. In our prototype, we will discuss the implementation of accessing spatial data from the Oracle spatial database. The schema of table that stores shape data is shown in Table 5-1.

Table 5-1. Schema of shape data tables

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>GID</td>
<td>NUMBER</td>
</tr>
<tr>
<td>GEOMETRY</td>
<td>SDO_GEOMETRY</td>
</tr>
</tbody>
</table>

The prototype of mobile agent based model is shown in Figure 5-2.
This prototype is implemented by Voyager, which is a Java based mobile agent application. More details are given as below:

- OraCSWrapper

  OraCSWrapper is the mobile agent wrapper at the computation server side. It sends out OraQueryAgent, which is the mobile agent bearing data request.

- OraWrapper

  OraWrapper is the implementation of data wrapper in this mobile agent based scenario. Upon receiving the query agent sent by OraCSWrapper, the OraWrapper object will call a method of ServiceAgent. Since the current computation server can only handle the spatial data in ESRI's shape file format, the OraQueryAgent will use the corresponding shape file name to indicate the spatial data it asks for. On the other hand, a table stored in oracle spatial database stores all geometries in a corresponding shape file. The name of the table is same as the shape file. Hence, we can use the shape file name to retrieve all geometries in its corresponding table. After the result is returned from OraLIV, a result agent is spawned to send result back to computation server.

- OraLIV

  ServiceAgent will call the methods in the oracle local interface. SetDataName is called to set the shape file name; SetUser and SetPwd are called to set the username and password which will be authenticated by the oracle spatial database; GetShpData is the method called to retrieve data from the oracle spatial database. GetShpData will call AccessData, which calls queryObject to get data from the database by JDBC connection. The method queryObject should not only query the data source but also transforms it to the ESRI's shape file format. The transformation process is shown in Figure 5-3. There are two steps. In the first step, SDO geometries stored in oracle spatial database tables are converted to a Geometry object. As the next step, the Geometry is transformed to shape file format using the shape file adapter.
Shape file adapter has some standard interfaces that are required by oracle spatial database. Beyond that, a shape file reader is implemented to actually read and write shape file.

In our prototype, the local interface view seems different from the design. The reason is we are lack of a real mediator. That is, query strings can’t be generated by the current mediator.

5.3 Prototype of CORBA Client-server Model Using Oracle Spatial Database

The prototype of CORBA client-server based implementation using oracle spatial database is shown in Figure 5-4.

Similarly, this prototype is implemented by HORB too except that the data source is oracle spatial database. More details are discussed as follows:
There are three classes defined at the server side.

- **CExecutor**
  
  CExecutor is used to start CWrapperDaemon object. It contains a main method. When user input exit, it will print out some message and tell the CWrapperDaemon object to stop all working wrappers, then quit.

- **CWrapperDaemon**
  
  It’s a Class to provide daemon object. When client tries to connect, it will first contact this daemon object. The daemon object keeps a list of all created wrappers. If a new request
comes, the daemon first checks in the list for an available wrapper. If available, provide the URL of the existing wrapper. Otherwise, create a new wrapper object and return the URL of this new wrapper. Certainly, this newly created wrapper should also be put into the list. When a client finishes using a wrapper object, it must free the wrapper for future use.

- **CWrapper**
  It's the parent class of all possible CORBA data wrapper. In its definition, it describes some common interface.

- **CWrapperORA**
  This class extends from CWrapper to implement a CORBA data wrapper for the oracle spatial database.

There is only one class defined at the client side.

- **CTestStub**
  To test the functions of server side objects.

The workflow can be described as follows:

i) CExecutor creates CORBA daemon object.

ii) The client sends requests to daemon object.

iii) The daemon object retrieves or creates a new wrapper object from the list. Return corresponding URL to the client.

iv) The client calls the wrapper object for data request.

v) The oracle CORBA wrapper object creates LIV and access data from oracle spatial database; Data is stored in a byte array (in shape file format) and return to the client.
CHAPTER 6 CONCLUSIONS AND FUTURE WORK

6.1 Summary of Current Work

The data integration, especially spatial data integration of various heterogeneous data sources to support field data collection and other field application is a big challenge toward current information technology. In our research, we focus on developing an infrastructure to provide user transparent data processing and integration capabilities.

Based on the previous work, improvements are made to achieve a more flexible and extensible architecture. New designs of treating heterogeneous data sources as set of object views help the object-oriented view system work more smoothly. This also helps the mediator to locate data source and generate query strings. Designs regarding metadata and rules give a more clear vision of how the mediator works. In terms of the data access component, data wrapper and local interface view are implemented to integrate Oracle spatial database. Besides, the CORBA based client-server platform is implemented to extend the communication capability of the infrastructure.

6.2 Future Work

Further efforts need to be made in the following aspects:

- Scalability of the infrastructure

The infrastructure is designed to do data integration based on a lot of heterogeneous databases. When the number of data source grows, the work overhead of the infrastructure
should increase in a reasonable way. In order to do this, the mediator, computation server, data wrapper’s functionality should be extended somehow to manage a growing data sources. For example, mediator may need to learn the relationships between databases which could accelerate the mediation process. As another example, if the mediator can know the general characteristics of one type of data sources and install the data wrapper automatically, the cost will be reduced greatly. Besides, data wrapper could also be more active and autonomous. It may collect metadata for the mediator and report the change of data source automatically.

- Mediation capability of the mediator

Mediator is by far the most important component in the whole infrastructure. In a real application, the mediation capability of the mediator will greatly affect the performance of the whole infrastructure since it determines the behaviors of other components in the infrastructure. However, the current functionality of the mediator remains limited. Hence, the overall efficiency of mediation needs to be improved. This probably could be achieved by two means. First, the mediator may do the optimization if there are several ways to get the same result set. Second, mediator should learn more from its previous mediations, such as data source’s processing ability, response time. This way, the mediator could help the field application to run faster and faster.
REFERENCES


