A Middleware Architecture for Open and Interoperable GISs

Steven H. Wong and Steven L. Swartz
National Oceanic and Atmospheric Administration

Dilip Sarkar
University of Miami

The volume and number of data sets about Earth are rapidly growing. However, sharing and integrating them is difficult due to incompatible data formats and platforms. We propose an abstract model for information sharing and integration and use it to develop an architecture for building open, component-based, interoperable systems.

A geographic information system is a computer-based tool for mapping and analyzing geospatial relationships between data sets. GIS data sets tend to have complex formats and large file sizes. Moreover, the volume of data about Earth is rapidly increasing. Due to the enormous costs involved in acquiring, producing, exploiting, and disseminating geospatial data such as satellite imagery, it's practically impossible for a single organization to own all the data it needs.

Current GISs are monolithic and platform-dependent applications containing redundant functions and databases, and it's difficult to share GIS data and geoprocessing methods among different software and hardware platforms. They also require excessive training because of the diverse user interfaces and they lack facilities to easily accommodate new methods and data types as they become available. Recent development of Internet map servers lets organizations build Web-based GISs in which users can view geospatial data via Web browsers. However, these proprietary GISs don't provide an easy way to integrate data among map servers or between the user's local data sets and map servers. To achieve integration, the user usually must download the data and process them locally. The exchange of data at the file level is inefficient, cumbersome for updating, and may involve complicated data conversions.[1]

A new computing paradigm for geoprocessing is necessary to move spatial information exchange from file transfer to a higher level. A new method should overcome barriers caused by incompatible GIS application platforms and multiple data sources and enable users to share, extend, and integrate functionality from different GISs in a distributed environment. To do just that, we propose an architecture for building open and interoperable GIS applications. An open and interoperable GIS adopts industry standards to create and manage GIS objects that perform spatial tasks in a distributed environment. Furthermore, encapsulating geospatial data and methods in a GIS object makes it possible for users to access information regardless of the locations of data sources and the application platforms involved in the implementation. (See the "Previous Work" sidebar for other approaches.)

Proposed architecture

Our architecture is based on an abstract model for information sharing and integration that enables peer-to-peer, object-oriented communication among data-handling applications via an object bus. In the proposed architecture, client applications access common object request broker architecture (Corba) objects for services. Thus, data sharing moves from the file-transfer level to the level of distributed objects conducting heavy transactions between multiple clients and servers. Corba interfaces in the middleware let server-side object implementations be transparent to clients. System implementation may involve using Java or other computer languages, as well as proprietary vendor applications. Consequently, the proposed middleware architecture provides a new way for GISs to share data and functionality in a platform-, vendor-, and location-neutral environment.

In addition, both users and developers benefit significantly from our techniques. Users have timely access to geospatial data and geoprocessing methods in a single, integrated, and virtual system. Developers gain the power of sharing and integrating GIS objects on the network through standard object interfaces. Another contribution is using Enterprise JavaBeans. We implement Corba objects with EJBs in the middleware. A major

1070-986X/02/$17.00 © 2002 IEEE

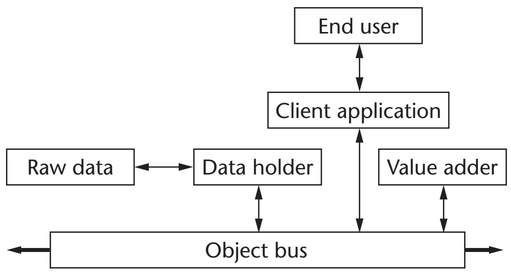


Figure 1. An abstract model for information sharing and integration. There can be multiple data holders, value adders, and client applications.

advantage of EJBs is that the bean developer and the client application programmer don't need to be concerned with service details such as transaction support, security, remote object access, and many other complicated error-prone issues.

Thus, taking advantage of the component-based feature of EJBs and Corba compatibility, our proposed architecture improves a GIS's flexibility and extensibility. We can also add functionality to a GIS by adding new components—GIS beans in the middleware. Clients can be Web browsers or any client application built around the Corba objects' distributed services, such as C++ and Java applications.

An abstract model

To provide a technology-independent framework that shares a variety of GIS data sets in a distributed, object-oriented, and peer-to-peer fashion, we've designed an abstract model for information sharing and integration (see Figure 1). In this model, both data holders and value-adders are data handlers that create programming objects to encapsulate data and processing methods to share and integrate information and services. Data holders might be government agencies, for example, that own raw data while value adders might be vendors that enhance the data's usability. Object interfaces are comparable across boundaries of data handlers. Data handlers make objects available by plugging them into the object bus. The plug-in capability of the objects is achieved by publishing them with a protocol that can be transported on the network.

Client applications serve as middlemen between end users and the services offered on the object bus. Data holders, value adders, or third parties can create client applications in several forms. Examples of these clients are Java applets that can run in Web browsers and C++ applications. Specifically, client applications

- collect information from objects on the object bus,

Previous Work

The Geospatial and Imagery Access Services (GIAS) specification from the US National Imagery and Mapping Agency (NIMA) is a milestone project dealing with geoprocessing interoperability. GIAS is a component in the US Imagery and Geospatial Information Service (USIGS) architecture (see http://164.214.2.59/sandi/arch/). GIAS defines an object-oriented application-programming interface (API) using Corba to remotely access and manipulate geospatial imagery.[1] Applying a subset of GIAS, Coddington et al.[2] have implemented a prototype distributed system for managing and accessing a digital library of geospatial imagery over a wide-area network. Cobb et al.[3] have investigated a system design for a Web-based object-oriented mapping database in which a Java client communicates with map applications in Smalltalk servers through a Corba interface. Wang and Jusoh[4] built a Web-based experimental GIS capable of integrating data from multiple servers via Corba interfaces. Abel et al.[5] proposed an architecture for a virtual GIS.

The OpenGIS Specification from the OpenGIS Consortium[6] is a comprehensive specification of a software framework for distributed access to geospatial data and geoprocessing resources. OGIS represents the GIS industry's most ambitious effort to facilitate the sharing and interoperability of spatial data so far. Among the more than 200 members of the OpenGIS Consortium are the Environmental Systems Research Institute, Microsoft, Oracle, and Sun Microsystems. NetGIS[7] experimentally implements OpenGIS Simple Features Specification for Corba[8] on the client side using a Java applet.

References

1. *USIGS Geospatial and Imagery Access Services* (GIAS) *Specification*, version 3.1, US National Imagery and Mapping Association, 1998.
2. P.D. Coddington et al., *Implementation of a Geospatial Imagery Digital Library Using Java and Corba*, tech. report DHPC-047, http://www.dhpc.adelaide.edu.au/reports/index.html, 1998.
3. M.A. Cobb et al., "An OO Database Migrates to the Web," *IEEE Software*, vol. 15, no. 3, May/June 1998, pp. 18-21.
4. F.J. Wang and S. Jusoh, "Integrating Multiple Web-Based Geographic Information Systems," *IEEE MultiMedia*, vol. 6, no. 1, Jan.–Mar. 1999, pp. 49-61.
5. D. Abel et al., "Towards Integrated Geographical Information Processing," *Int'l J. Geographical Information Science*, vol. 12, no. 4, June 1998, pp. 353-371.
6. *The OpenGIS Guide*, 3rd ed., Open GIS Consortium Technical Committee, http://www.opengis.org/techno/guide.htm, 1998.
7. J. Kahkonen, "Interactive Visualisation of Geographical Objects on the Internet," *Int'l J. Geographical Information Science*, vol. 13, no. 4, June 1999, pp. 429-438.
8. *OpenGIS Simple Features Specification for Corba,* revision 1.0, Open GIS Consortium, http://www.opengis.org/techno/specs.htm, 1998.

- present information to the end user through a user interface,

- incorporate information into the user's local data sets,

- let the user send local data to data handlers that create and publish new objects encapsulating the user's data, and

- enhance the data with a set of processing methods.

In the real world, the data-holder and value-adder roles become indistinct, especially in an environment in which multiple data handlers exist. Objects published by these data handlers complement and interact with one another through dynamic transactions at a peer-to-peer level. In other words, an object can both request and provide services. In the process, new data are generated and stored in servers operated by data handlers. Note that transactions conducted among objects on the object bus are transparent to end users.

Integrating geospatial data and functionality with mainstream business software is a shift in the GIS industry.[2] Together with other business objects on the object bus, GIS objects provide data and services in a distributed client–server environment represented by our model's framework. To design system architecture based on the abstract information model, we must adopt industry standards and define object interfaces to ensure interoperability and connectivity among applications across enterprises.

Corba interfaces for GIS objects

The goal for open and interoperable computing is to let applications communicate with one another no matter where they're located or who designed them. Corba can address this goal. Two important aspects of Corba[3] facilitate the interoperable computing paradigm:

- The Interface Definition Language (IDL) and the application programming interfaces (APIs) enable client–server object interaction within a specific implementation of an object request broker. The ORB serves as an object bus that handles all communication between a client and a server object.

- The Internet interORB protocol (IIOP) makes any Corba-conformant ORB instantly usable across the Internet without requiring any additional programming. IIOP's speed and efficiency lets it easily handle transaction-based functions. Companies like Netscape and AOL, Oracle, IONA, Inprise, IBM, Computer Associates, and dozens of others are incorporating Corba/IIOP in their products.[4] In fact, Netscape has incorporated ORBs in its Netscape Communicator 4.x.

For applying Corba to GISs, the OpenGIS Simple Features Specification for Corba[5] provides Corba IDL interfaces that define GIS Corba object attributes and behaviors. This specification lets developers create interoperable GIS applications to access and manipulate geospatial features with "simple" geometry (points, lines, and polygons). In the future, OpenGIS will likely formulate Corba specifications for more spatial types such as 3D features and raster coverages. (For brevity, the phrase *OpenGIS for Corba* in this article stands for the OpenGIS Simple Features Specification for Corba. OpenGIS for Corba might occasionally include future specifications for additional spatial types.)

The current paradigm for GIS applications represents the view that geoprocessing functionality is tightly coupled to its internal data models and structures. With GIS objects conforming to OpenGIS for Corba, GIS applications become temporary collaborations of GIS Corba objects that hide data models and structures.

We can implement Corba objects using Java, C++, or other computer languages. The advantages of using Java as the implementation language include portability across platforms, robustness through runtime memory management, easy program development and maintenance, easy client integration with Web browsers, object orientation, security, and multithreading.

EJBs are a component-based Java technology for server-side object implementation. The EJB specification[6] lays out the format of a bean and a set of services that the EJB container in which the bean runs must provide. This makes EJBs a powerful development methodology for component and distributed applications. Neither the bean developer nor the client application programmer need to be concerned with service details such as transaction support, security, remote object access, or many other complicated error-prone issues. The EJB server and container transparently provide these services for developers. Furthermore, EJBs offer portability. A bean developed on one EJB server should run on other EJB servers that meet the EJB specification, which was designed to be 100 percent Corba compatible. An EJB is actually a Corba server object defined using Java remote method invocation semantics.[4]

Our proposed architecture uses OpenGIS for Corba as an interface for communication between clients and GIS objects via the Corba ORB object bus. We can implement these objects using EJBs in the middleware servers that can obtain data from backend database servers. We can assemble and reuse the GIS beans without much difficulty, a benefit of a component architecture. To the best of our knowledge, researchers haven't adequately addressed applying EJBs to geospatial processing in the GIS community.

Figure 2 elaborates on how data holders and value adders can benefit from the model in Figure 1. Information services from both data holders and value adders are published through Corba objects. A data holder adopts middleware EJBs to implement the services and obtain information drawn from the raw data. In the process, we can create and archive derivative data. EJBs can also request services from Corba objects on the bus. This lets a value adder

- connect to Corba objects from data holders,
- add functionality to the objects, and
- publish new Corba objects.

Moreover, we can add additional derivative data to the value adder's database.

GIS beans can communicate with database servers through Java database connectivity (JDBC) and OpenGIS for Structures Query Language (SQL). The OpenGIS for SQL specification defines a standard SQL schema that supports storage, retrieval, query, and update of geospatial feature collections via the open database connectivity (ODBC) or JDBC API. (The OpenGIS Simple Features Specification for SQL is currently available.[7] The commercial products that conform to this specification are Oracle Spatial Option[8] and ESRI Spatial Database Engine.[9])

System architecture

Figure 3 (next page) depicts our architecture for building open and interoperable GISs. It's a Corba-based implementation of the abstract model for information sharing and integration. In this architecture, objects in clients and servers communicate through the Corba ORB object bus using the IIOP.

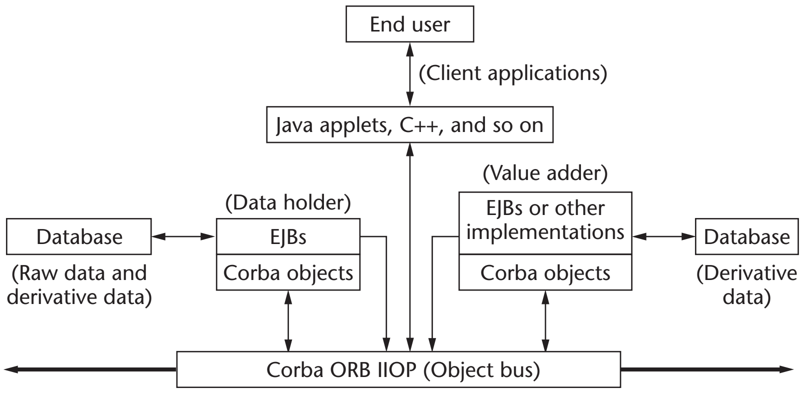


Figure 2. An implementation of the abstract model for information sharing and integration with Corba-based specifications. The thin arrow between EJBs and the object bus indicates that EJBs can request information and services from objects on the bus.

The architecture's core components are Corba objects in middleware Corba subsystems. EJBs and object wrappers in middleware servers capture the Corba objects' characteristics and implement their business logic. We propose to connect the EJBs to backend database servers through JDBC and OpenGIS for SQL. Clients interact with the GIS through Corba or HTML interfaces.

Corba subsystems

The Corba subsystems are OpenGIS Corba, Metadata Corba, Security Corba, and OpenVendor Corba. These subsystems publish Corba objects for clients to use. A client can be an application or an object accessing the Corba objects. Therefore, an object in one Corba subsystem can be a client of an object in another Corba subsystem.

To enable easy communication between Corba objects and EJBs, we use Java classes to materialize Corba objects. One of the software products that accomplishes that is VisiBroker for Java from Borland/Visigenic.[10] The Corba objects in the form of Java classes can delegate tasks to EJBs through Java naming and directory interfaces (JNDIs).

OpenGIS Corba subsystem. The OpenGIS for Corba specification[5] defines GIS Corba objects in this subsystem. Client applications interact with these objects through their interfaces to access and manipulate spatial information comprised of geometric features such as points, lines, and polygons. The OpenGIS for Corba contains three groups of interfaces: feature interfaces, spatial reference system interfaces, and geometry interfaces.

A Feature object represents a real-world entity or an abstraction of the real world. It's constructed from geometry objects, attributes (properties), a spatial referencing system, and associated meth-

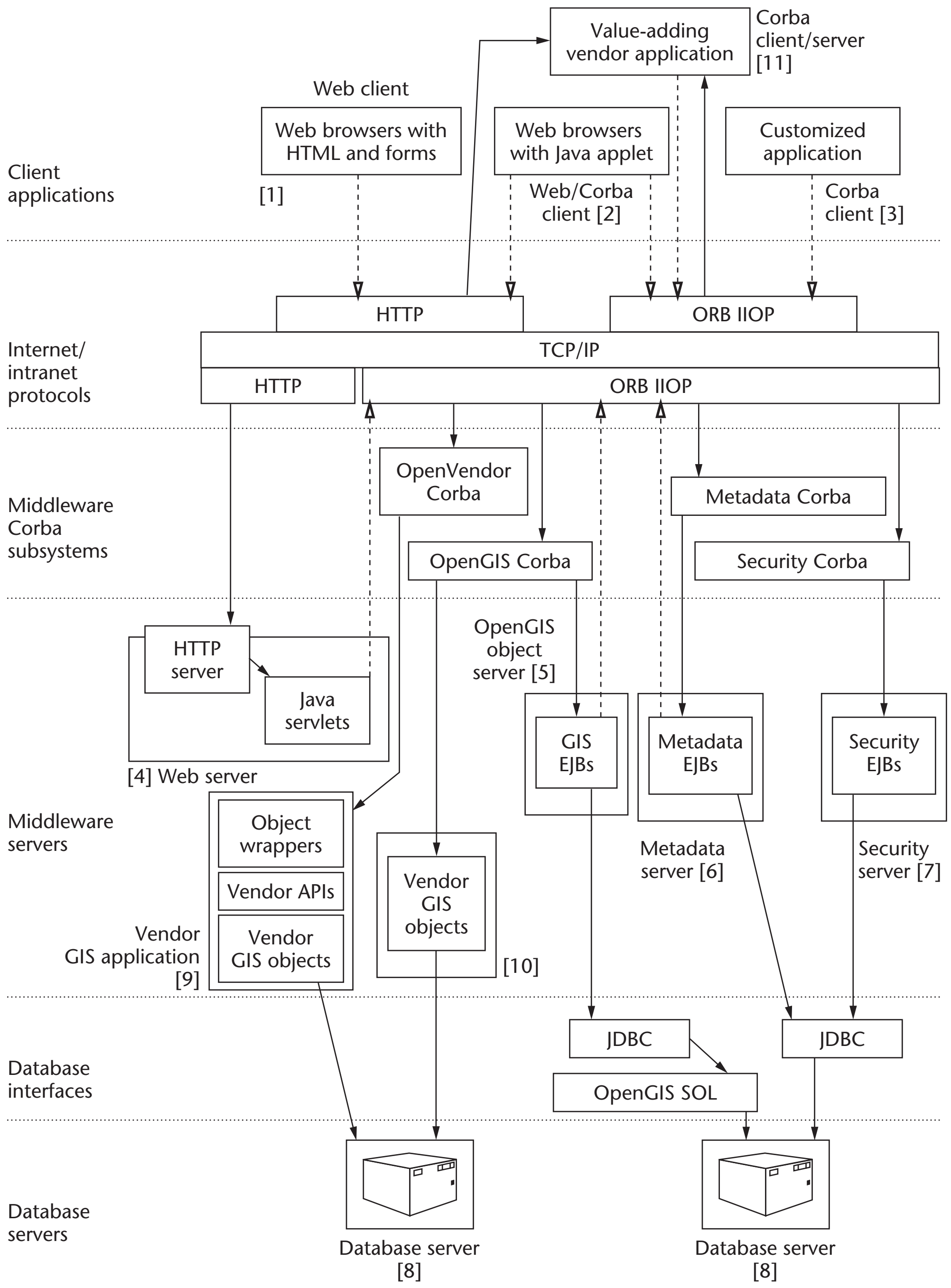


Figure 3. An architecture for open and interoperable GIS. Dotted lines with hollow arrows denote requests for services. Solid lines with solid arrows denote implementations of services.

ods. A Geometry object has coordinates that can be mapped into real-world positions by a SpatialReferenceInfo object representing a spatial reference system. A GIS layer is a collection of features. For example, a GIS layer might contain topographical features such as elevation contours (lines). A ContainerFeatureCollection or QueryableContainerFeatureCollection object represents a GIS layer. Each GIS layer exposes itself to clients through either Corba naming or Corba trader (catalog) services (the white and yellow pages, respectively, of a Corba service). More than 50 interfaces are defined in OpenGIS for Corba,[5] providing a rich set of geospatial characteristics.

The OpenGIS object server in the middleware (see [5] in Figure 3) implements the GIS Corba objects. We use EJBs (GIS beans) as components containing the business logic for geospatial data access and manipulation. The GIS beans accept and process requests from GIS Corba objects. The EJBs obtain necessary information from backend database servers through JDBC and OpenGIS for

SQL interfaces. These GIS beans can be clients of Corba objects outside of the OpenGIS Corba subsystem (see the dotted line between GIS EJBs [5] and the ORB object bus in Figure 3). Through this client–server mechanism, GIS beans can request services from Corba objects published by GISs in other organizations. GIS beans can also delegate certain implementations to objects in the OpenVendor Corba subsystem that maintains vendor GIS applications (see [9] in Figure 3) noncompliant with the OpenGIS specifications. (See the "Sample Mechanism" sidebar, next page, for an example.)

Vendor GIS applications that comply with the OpenGIS for Corba specification may be plugged into the middleware to implement some of the functionality of the OpenGIS Corba subsystem. This mechanism, along with the OpenVendor Corba subsystem we discuss later, provides extensibility to the system on the server side, which lets the GIS systematically incorporate commercial technology advances. Thanks to Corba interfaces, all implementations of vendor applications are transparent to clients regardless of vendors' compliance to the OpenGIS for Corba specifications.

Metadata Corba subsystem. The Metadata Corba subsystem contains metadata Corba objects that let users access information about GIS layer attributes and methods. The metadata objects contain attributes that are mapped to certain attributes defined in the Federal Geographic Data Committee (FGDC) metadata standard.[11] Clients access these metadata objects to search available GIS databases and database schema descriptions. Metadata EJBs in the metadata server (see [6] in Figure 3) implement the business logic of metadata Corba objects. The metadata EJBs obtain information about available GIS layers through two mechanisms:

- by accessing the backend database server via JDBC and
- by accessing FeatureType objects contained in GIS layers in the OpenGIS Corba subsystem.

The dotted line between the metadata EJBs and the ORB object bus in Figure 3 indicates the latter mechanism.

Most of the metadata for a GIS layer can be found in its FeatureType object. However, the metadata in the FeatureType object might not be organized according to the FGDC metadata standard. Metadata EJBs in the metadata server query the FeatureType object to obtain a GIS layer's metadata and present the information to clients in compliance with the FGDC metadata standard.

The Metadata Corba subsystem presents metadata held by the database servers to the GIS users. The users can then make out the meanings (semantics) of attributes in the GIS layers (OpenGIS Corba objects) and make informed choices for spatial and nonspatial features. The Metadata Corba subsystem, however, doesn't interact with metadata from other GISs to discover useful GIS layers from other GISs. This would be the task of the OpenGIS catalog services currently under development,[12] raising the GIS interoperability from a syntactic to a semantic level.[13]

In the future, we expect to replace Metadata Corba objects with OpenGIS catalog Corba interfaces.[14] With the catalog services playing the role of the interface between clients and metadata, the metadata's format becomes transparent to the clients. We can reuse the metadata EJBs in the current architecture to implement part of the catalog service interfaces in the improved architecture. We expect

- the catalog services to return metadata information about GIS layers held in its database servers in response to requests from catalog services of other GISs. To this end, we can reuse most of the metadata EJBs from the current middleware servers to serve the catalog Corba objects.
- the catalog services to discover useful geospatial features from other GISs by querying catalog interfaces of other GISs dynamically. We must create new metadata EJBs to help the catalog Corba objects with this task.
- the OpenGIS Catalog Service to be a major mechanism to dynamically resolve semantic heterogeneity between attributes from GIS layers held in different GISs.

Security Corba subsystem. The Security Corba subsystem contains security Corba objects that facilitate the authorized access of certain GIS databases. EJBs in the security server implement the security objects (see [7] in Figure 3). The security EJBs access the security database via JDBC. The system manager manages the security database from a database management system (DBMS) or a client application that communicates with the Security Corba objects.

Sample Mechanism

This hypothetical example demonstrates an application interoperating with an ArcView application in a distributed environment. In this example, a Corba client requests and obtains services on behalf of a MapInfo (from MapInfo Corporation) user from the OpenGIS Corba subsystem (see Figure A). The GIS application that provides the service is an ArcView application, most likely in a different computer system. ArcView GIS (from ESRI) doesn't conform to the OpenGIS for Corba specification. Therefore, we can't plug it into the OpenGIS Corba subsystem in a straightforward manner as in item [10] in Figure 3. Within our proposed architecture's framework, the GIS EJBs in item [5] of Figure 3 delegates the requests from the Corba client to the OpenVendor Corba subsystem that connects to the ArcView application.

The ArcView project (application) named Miami_Land_Project contains database tables, database displays (views), ArcView scripts (Avenue[1] codes), and so on for land-use information of Miami, Florida. The data include attributes for parcel polygons, aerial photos covering the Miami area, and zoning division polygons. ArcView provides a C library called AVExec to facilitate programs written in C or C++ to execute ArcView Avenue statements.[1] We created an OpenVendor Corba object called Miami Land with C++. It's an instance of the Corba class ArcView Corba (see the "Example IDL Descriptions for Corba Interfaces" on page 74 sidebar for a partial description). Arc ViewCorba delegates its method calls to the C++ class ArcViewObject Wrapper that has methods executing ArcView Avenue statements through the AVExec C library. ArcViewCorba OpenVendor Corba

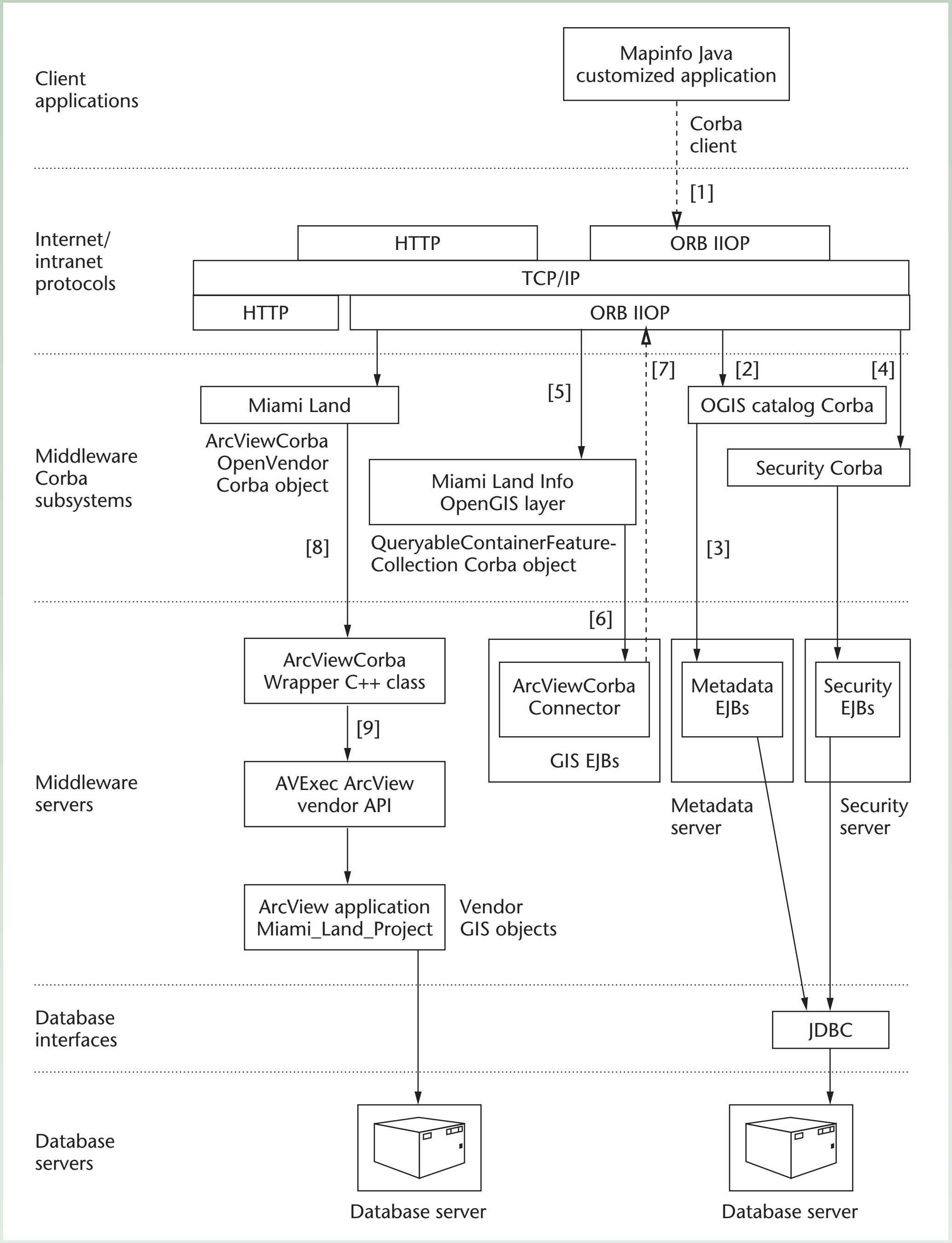


Figure A. A MapInfo client accesses data in ArcView through OpenGIS Corba interfaces. Dotted lines with hollow arrows denote requests for services. Solid lines with solid arrows denote implementations of services.

object can provide geospatial services from multiple types of GIS layers from an ArcView project, which the current OpenGIS specifications don't support. In this example, these multiple types of GIS layers include vector polygons (parcels and zoning divisions) and raster images (aerial photos).

The following steps show how the MapInfo application user obtains a parcel polygon from the remote ArcView application through an OpenGIS Corba interface:

1. The MapInfo user asks for a parcel polygon in the Miami area to overlay on some GIS layers in the user's local computer. The end user doesn't know where the parcel polygon is.

2. We can customize certain MapInfo applications, such as MapXtreme Java Edition, with Java (see http://www.mapinfo.com/software/mapxtreme/java/index.html). Therefore, a MapInfo application itself can be a Corba client written in Java. The client contacts an OpenGIS Catalog Corba object (likely in a remote server transparent to the client) with the request for retrieval of the parcel given its extent and spatial reference system (see [1] and [2] in Figure A). The Catalog Corba object sends the request to metadata EJBs (see [3] in Figure A) that search the information from a backend database. The metadata EJB finds out dynamically that an OpenGIS Corba object called Miami Land Info can provide the polygon. This mechanism helps resolve the semantic variability of GIS data sets. The Miami Land Info object is an instance of the QueryableContainerFeature Collection Corba class (see the "Example IDL Descriptions for Corba Interfaces" sidebar, on page 74).

3. After the authentication process involving the Security Corba objects (see [4] in Figure A), the client requests the parcel polygon from the Miami Land Info Corba object by issuing a method call `get_geometry` of the Corba object, supplying the parcel's extent as a rectangle and the spatial reference system as a string for input parameters (see [5] in Figure A). Note the client is unaware of the location or the name of the server where the Corba object is deployed.

4. The Miami Land Info Corba object has an attribute (`ImplementorEjbName`) indicating that the GIS EJB ArcViewCorbaConnector implements the Corba object. Another attribute (`OpenVendorCorbaObjName`) in the Miami Land Info Corba object indicates it's associated with the Miami Land OpenVendor Corba object. The Miami Land Info Corba object relays the get_geometry request to the ArcViewCorbaConnector EJB (see [6] in Figure A) along with the name of the OpenVendor Corba object (Miami Land).

5. The ArcViewCorbaConnector EJB obtains a reference to the Miami Land OpenVendor Corba object and passes the `get_geometry` request to it (see [7] in Figure A).

6. The Miami Land object relays the `get_geometry` request to the C++ class ArcViewObjectWrapper, along with the name of the ArcView project (Miami_Land_Project, stored in the `ArcViewProjectName` attribute) with which the Corba object is associated (see [8] in Figure A).

7. The C++ object ArcViewObjectWrapper has a `get_geometry` method that calls the ArcView project (Miami_Land_Project) through AVExec, supplying the parcel's extent and the spatial reference system (see [9] in Figure A).

8. The ArcView application understands the statements from the ArcViewObjectWrapper carried by AVExec, finds the parcel, transforms the parcel polygon into the specified spatial reference system, and returns the parcel polygon as a string containing the coordinates of the polygon's nodes.

9. ArcViewObjectWrapper repackages the returned polygon string into a WKSGeometry object.[2] The WKSGeometry object is eventually returned to the Java client through the Miami Land OpenVendor Corba object, the ArcViewCorbaConnector EJB, and the Miami_Land_Info OpenGIS Corba object.

10. The Java client integrates the WKSGeometry object into the MapInfo application by, possibly, converting the WKSGeometry parcel object into a polygon in MapInfo format. The user sees the parcel overlaid on other GIS layers. The user can perform a geospatial operation on the parcel polygon object like any local geographical object, such as computing the polygon's area.

References

1. *Using Avenue*, Environmental Systems Research Institute, 1996.
2. *OpenGIS Simple Features Specification for Corba,* revision 1.0, Open GIS Consortium, http://www.opengis.org/techno/specs.htm, 1998.

The Security Corba subsystem acts as a gatekeeper for the Corba objects in other subsystems. Client applications must be authenticated by security objects before they can interact with GIS Corba objects.

OpenVendor Corba subsystem. The OpenVendor Corba subsystem enables vendor-neutral use of vendor GIS applications and legacy geospatial data that don't comply with the OpenGIS specifications. The GIS EJBs in the OpenGIS object server (see [5] in Figure 3), along with other clients, can take advantage of the functionality provided in vendor GIS packages through OpenVendor Corba objects. Vendor neutrality is achieved because clients only communicate with the Corba interfaces. Clients don't need to be concerned about the data models and structures of vendor applications used to implement these interfaces.

Some vendor GIS applications provide geospatial functionality unavailable in the current OpenGIS specifications such as 3D mapping. OpenVendor Corba objects can complement the current OpenGIS specifications by providing clients the special geospatial functionality through Corba interfaces. When OpenGIS specification for the functionality is available, we can create new OpenGIS Corba objects and link them to the OpenVendor Corba objects to provide the functionality through standard interfaces.

Server-side implementation of the OpenVendor Corba objects involves using object wrappers written in Java, C, C++, or other languages that are mapped to Corba. The object wrappers are objects that access the functionality in vendor applications through APIs the vendors provide.

Placing the OpenVendor Corba objects between clients and vendor APIs makes it possible for most vendor applications to plug into the GIS. As a result, the GIS can maintain a heterogeneous vendor environment to fill functionality gaps that might exist in the OpenGIS Corba subsystem. Through the OpenVendor Corba interfaces, a vendor application can implement the business logic of some GIS EJBs in the OpenGIS Corba subsystem. This approach promotes reusing functionality and increases the system's responsiveness to new commercial methodologies. (See the sidebar "Sample Mechanism" for an example that illustrates accessing a proprietary GIS through this subsystem.)

Client applications

Client applications perform one or both of these tasks: creating a user interface to interact with the user or requesting services from Corba objects or HTTP servers on the user's behalf.

Three types of client applications are Web, Corba, and Web/Corba clients. A Web client (see [1] in Figure 3) follows guidelines in the OpenGIS Web Map Server Interface Implementation Specification,[15] a product of the OpenGIS Web Mapping Testbed (see http://www.opengis.org/wmt/index.htm), to access geospatial data from multiple servers in different sites. It uses HTML pages for the user interface. Java servlets dynamically create these HTML pages (see [4] in Figure 3). These Java servlets are actually Corba clients because they obtain services from GIS objects in the Corba subsystems, which the dotted line between the Java servlets and the ORB object bus in Figure 3 indicates. The Java servlets generate and render display elements from these GIS objects. The Web client may also connect to Web servers operated by value-adding vendors (see [11] in Figure 3).

A Corba client implements a customized user interface using a computer language with mapping to Corba interfaces. Typical Corba clients include Java servlets and customized client applications written in C, C++, Java, and so on (see [4] and [3] in Figure 3). Java servlets request services from Corba objects on behalf of Web clients and return results in HTML format to Web clients. Customized client applications can create flexible and powerful user interfaces because they are only limited by the capacity of the computer languages used for client-side implementation. A customized client application enables client–server integration of GIS data. In addition to being able to incorporate geospatial information from Corba objects into local data sets, an end user can create GIS layers from local data sets and publish the GIS layers through the OpenGIS Corba subsystem. The published GIS objects encapsulate data from the end user's local data sets and contain a set of geoprocessing methods. This information integration mechanism is a powerful tool by which a GIS application can spatially enable end users' data sets, enhance the data with processing methods, and facilitate data sharing.

The Corba client/server (see [11] in Figure 3) is a Corba client and a server that provides services beyond those offered by Corba objects of which it's a client. To provide the extended services to its clients, the value-adding Corba client/server might let users connect to it either through its Web server or its Corba interfaces. The Corba client/server provides extensibility to the architecture on the client side. In fact, under this setting, any GIS can be a Corba client/server to another GIS. Services offered by objects from different GISs complement one another regardless of the software and hardware platforms involved in their implementations.

A Web/Corba client runs in a Corba-enabled Web browser (see [2] in Figure 3). It loads Java applets from a Web server. These Java applets can interact with the user through its graphical user interface (GUI) and communicate with Corba objects to obtain services. Similar to a customized client application, an end user of a Web/Corba client application can integrate GIS Corba objects into local data sets and create new GIS layers from local data sets. However, because of restrictions the Java security model imposes, this is relatively difficult to accomplish because it involves running trusted applets signed by "trusted" servers.[16]

Implementation

We carried out a prototype implementation of our architecture to demonstrate its feasibility and validity. The implementation focuses on the middleware in which we implemented a subset of OpenGIS for Corba[5] using Java, EJBs, and a DBMS. Implemented objects include GIS Corba objects that can be published on the Internet or an intranet for clients to use as well as GIS EJBs that implement the business logic of the Corba interfaces and access spatial data from databases compliant with the OpenGIS for SQL specification. On the client side, we implemented a simple customized Java application to access and query published GIS Corba objects.

We used real-world data on oceanography for this implementation. The oceanographic data sets are a subset of data from the study of Florida Bay conducted by the Southeast Fisheries Science Center of the National Oceanic and Atmospheric Administration (NOAA) Fisheries.

We created these four GIS layers from the four data sets using methods we explain next:

- a point GIS layer (Station), corresponding to locations of water property sampling stations;
- a polygon layer (Sub-basin), corresponding to division of Florida Bay into sub-basins;
- a line layer (Tidal Amp), corresponding to contours of the bay's tidal amplitudes; and
- a line layer (Effort), corresponding to routes helicopters use for surveying wading-birds and large-fish survey.

Figure 4 (next page) shows both the logical and physical configuration of the prototype implementation.

Server-side implementation

In this implementation, we retrieved GIS data from an Oracle database using an Oracle spatial option[8] that conforms to the OpenGIS Simple Features Specification for SQL.[7] According to the SQL schema defined in the specification, each GIS layer is stored as a feature table. Oracle spatial provides utilities for creating, loading, and indexing spatial feature tables (GIS layers) from the real-world oceanographic data sets. We can query all tables for a GIS layer like any table in a relational database management system. Spatial queries on a feature table can be performed by the SQL functions that have been implemented using the OpenGIS Simple Features Specification for SQL.

GIS EJBs that implement GIS Corba objects access GIS feature tables and spatial SQL functions in the Oracle database through JDBC and the OpenGIS Simple Features Specification for SQL.

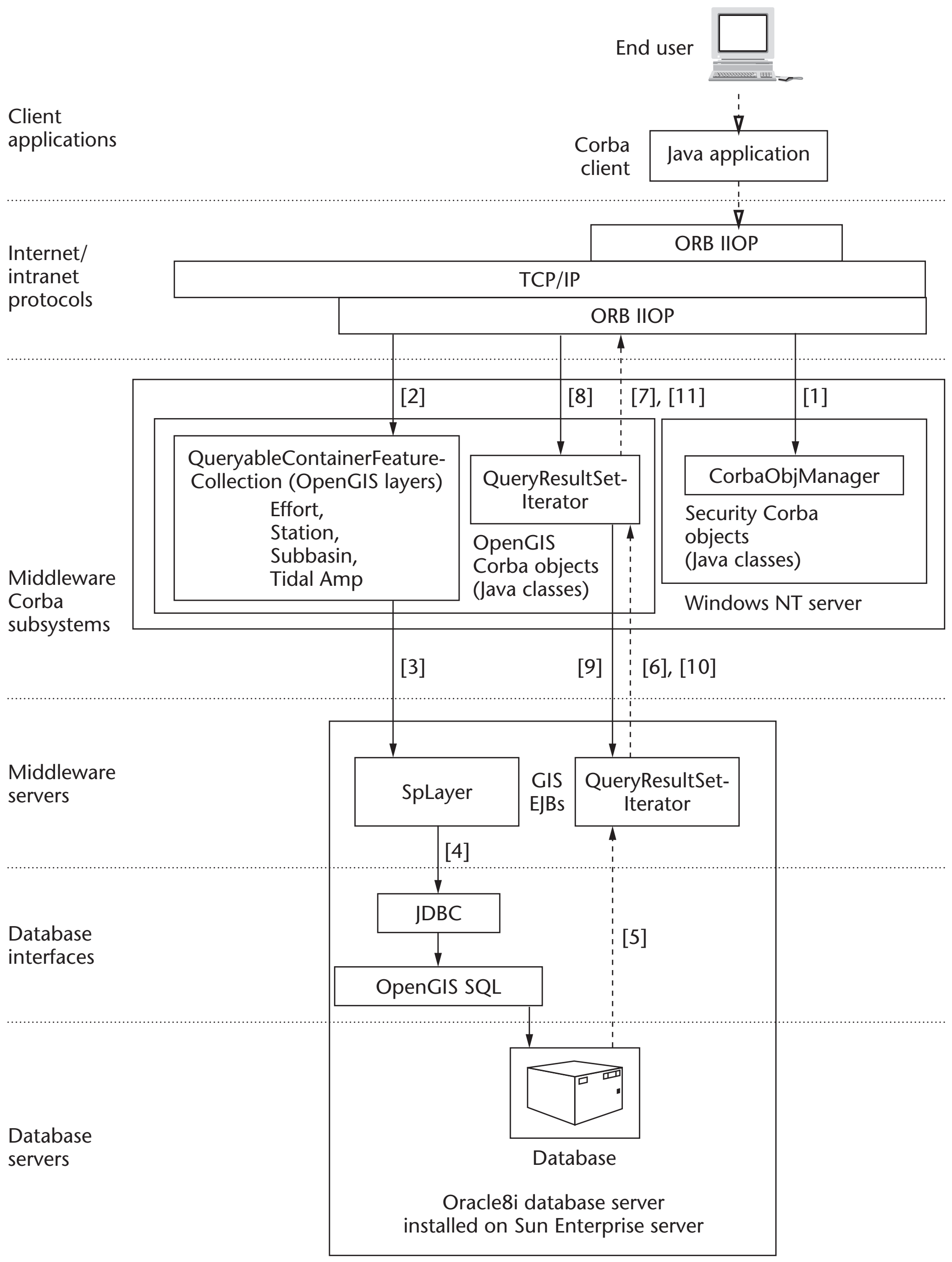
We have implemented several necessary Corba interfaces for client applications to access GIS layers in the Internet/intranet environment. These interfaces include QueryableContainerFeatureCollection, QueryResultSetIterator, and CorbaObjManager. All Corba interfaces are in Java classes.

CorbaObjManager is a new Corba interface we defined to function as a gatekeeper. All client applications must go through this interface to obtain handles for other Corba objects. Its `getGisLayer` method takes the user's name and password and the name of a GIS layer as strings and returns a reference to a QueryableContainerFeatureCollection object to the client. If data for the GIS layer don't exist, the method throws a LayerNotCreated exception. CorbaObjManager is part of the security Corba subsystem.

QueryableContainerFeatureCollection and QueryResultSetIterator Corba interfaces are defined in OpenGIS for Corba. We implement methods of these Corba objects with EJBs. A Queryable ContainerFeatureCollection object represents a GIS layer. Given a query string and other parameters to define a query's scope, a client application can invoke the `evaluate` method of the GIS layer to obtain information in the GIS layer. The `evaluate` method returns a QueryResultSetIterator object for the client to navigate the query's result set. We can deploy these Corba objects (in the form of Java classes) to any server that has a Java virtual machine. Figure 4 shows the deployment of Corba objects in a Windows NT server.

SpLayer is a GIS EJB that we create to implement the business logic of the QueryableContainerFeatureCollection Corba object, which delegates all requests from client applications to methods in the SpLayer. SpLayer can communicate with other GIS EJBs to fulfill requests from QueryableContainerFeatureCollection objects. The QueryResultSet-

Figure 4. Logical and physical configuration of the prototype implementation. Dotted lines with hollow arrows denote a request for services. Solid lines with solid arrows denote implementation of services. Dotted lines with solid arrows denote returning objects once a service is rendered.



Iterator EJB implements methods in QueryResultSetIterator Corba interface. The EJB holds and navigates the result set for a spatial query.

We implemented two methods with this GIS bean: **`advance`** and **`get_geometry_by_name`**. The advance method advances the iterator to the next record in the result set. Initially, the iterator is positioned before the first record. Calling the **`get_geometry_by_name`** method of the QueryResultSetIterator EJB with the geometry field's name as the input parameter returns a WKSGeometry object. The WKSGeometry object defined in the Open GIS Consortium's specification[5] is used as a basic object representing geometry of simple features (points, lines, and polygons, and so on) and is exchanged between client and server objects. Methods in the SpLayer GIS bean perform these tasks:

- process requests from clients and carried by the QueryableContainerFeatureCollection Corba object,
- fetch data from an Oracle server using spatial functionality in the Oracle spatial option, and

- return results to the QueryableContainer-FeatureCollection Corba object through QueryResultSetIterator EJB and QueryResult-SetIterator Corba object.

Corba objects and GIS EJBs communicate through the JNDIs protocol. We used Oracle8i Enterprise Database Server as an EJB server and a container in which GIS EJBs are deployed (see Figure 4).

Client-side implementation

We have developed a customized Java application (see [3] of Figure 3) with a minimal GUI. It can connect to GIS Corba objects and invoke a subset of methods in these objects. The purpose in this implementation phase is to ensure middleware components of the architecture work in a coordinated manner. Our future implementation will include significant improvements in both the user interface and interactions with the Corba subsystems.

The GUI has a window to display GIS maps and tool buttons for the user to interact with these maps. The tool buttons perform these tasks:

- zoom in and out,
- pan,
- overlay GIS layers from different sources,
- display the cursor's coordinates in a status bar as the cursor moves over the layers on the screen, and
- move a layer upward or downward in relation to other layers.

In a real query, the Corba client interacts with the middleware Corba interfaces, and GIS EJBs assist the Corba objects to access a GIS layer stored in a feature table in the database (see Figure 4). To access GIS Corba objects, the Java client first contacts the CorbaObjManager Corba object that is published on the Internet or an intranet. The Corba ORB IIOP facilitates this communication. The client requests a reference to a GIS layer, such as the Tidal Amp, by sending a GetGisLayer call defined in the CorbaObj-Manager interface (see [1] in Figure 4 and the "Example IDL Desciptions for Corba Interfaces" sidebar, next page). The client presents a user name, a password, and the name of requested GIS layer (Tidal Amp) as input parameters.

In the current implementation, the Corba-ObjManager stores a set of valid user names and passwords in its Java class. A future implementation will let the CorbaObjManager access user information from a database through JDBC. Once the validity of a user is confirmed by Corba-ObjManager, the client obtains a reference to the Tidal Amp Corba object, an instance of the QueryableContainerFeatureCollection OpenGIS Corba interface. The client then invokes the `evaluate` method of the GIS layer Tidal Amp (see [2] in Figure 4), starting a query. The Tidal Amp Corba object delegates the query task to the `evaluate` method of a SpLayer GIS EJB, along with the necessary information (for example, the `FeatureTableName` attribute) for the EJB to locate the feature table for the Tidal Amp GIS layer in the Oracle database (see [3] in Figure 4 and the "Example IDL Desciptions for Corba Interfaces" sidebar). This is done through the JNDI protocol, without the client's awareness. The SpLayer EJB queries the feature table through JDBC and OpenGIS Simple Features Specification for SQL interfaces (see [4] in Figure 4). The Query-ResultSetIterator EJB that holds the result set of the query (see [5] in Figure 4) is wrapped inside a QueryResultSetIterator Corba object (see [6] in Figure 4) and returns to the client (see [7] in Figure 4). This lets the client navigate all spatial features of the GIS layer Tidal Amp, using the combination of the `advance` and `get_geometry_by_name` methods of the QueryResultSetIterator Corba interface (see [8] in Figure 4). Keeping transparent to the client, the QueryResultSetIterator Corba object delegates the method calls to QueryResultSetIterator EJB (see [9] in Figure 4), which holds the result set of the `evaluate` query. The `advance` method iterates through the records in the result set. The `get_geometry_by_name` method returns a WKSGeometry object (see [10] and [11] in Figure 4). The WKSGeometry Java objects encapsulate spatial features of the Tidal Amp GIS layer that can be used to generate display elements to be passed to the Java client's GUI for rendering. Other GIS layers (Effort, Station, and Sub-basin) can also be accessed in the same way and overlaid on one another.

In the next implementation phase, we propose to use WKSGeometry objects to wrap spatial data on the end user's computer. These WKSGeometry objects can be passed to a QueryableContainerFeatureCollectionFactory Corba object[5] that then creates new GIS layers on the server side for publication on the network.

Example IDL Descriptions for Corba Interfaces

We only define some of the attributes and methods we mention in this article in the interfaces here. In this sidebar, we provide an explanation of Figure B and then give two other examples (see Figures C and D) for further reference. For detailed documentation on the standard Corba IDL, see the Open GIS Consortium's specification.[5]

Figure B gives the QueyableContainerFeatureCollection Corba interface. We added the three `readonly` attributes to the interface for our architecture:

- `readonly attribute string ImplementorEjbName;` is the name of the EJB that implements this Corba object.
- `readonly attribute string OpnVendorCorbaOjbName;` is the name of the OpenVendor Corba object from which this Corba object can obtain services through the EJB. It can be an empty string meaning no OpenVendor Corba object is needed for service.
- `readonly attribute string FeatureTableName;` is the name of the feature table in DBMS that stores data for the GIS layer. It can be an empty string if this Corba object depends totally on the OpenVendor Corba object for services.

The subsequent operation (`Geometry get_geometry(in NVPairSeq geometry_context) raises (InvalidParams);`) is inherited from Feature interface that is inherited by FeatureCollection. Geometry context can be information on a spatial reference system, a bounding polygon, and so on. This final operation in Figure B is inherited from QueryEvaluator.

```
module OGIS {

interface QueyableContainerFeatureCollection :
FeatureCollection, QueryEvaluator {
  readonly attribute string ImplementorEjbName;
  readonly attribute string OpnVendorCorbaOjbName;
  readonly attribute string FeatureTableName;

  Geometry get_geometry(in NVPairSeq geometry_context) raises
  (InvalidParams);
  QueryResultSetIterator evaluate(in string query,in QLType
    q1_type, in
  NVPairSeq params) raises (QueryLanguageTypeNotSupported,
  InvalidQuery, QueryProcessingError);
};

};
```

Figure B. The QueyableContainerFeatureCollection Corba interface.

```
#include "OGIS.idl"

module GisCorba {

interface CorbaOjbManager {
  exception LayerNotCreated { string why; };
  OGIS::QueryableContainerFeatureCollection getGisLayer(
  in string username, in string password, in string
  GisLayerName ) raises (LayerNotCreated);
};

};
```

Figure C. The CorbaOjbManager Corba interface.

```
#include "OGIS.idl"

module OpenVendorCorba {

interface ArcViewCorba {
  readonly attribute string ArcViewProjectName;
  OGIS::WKSGeometry get_geometry ( in OGIS::NVPairSeq
  geometry_context );
};

};
```

Figure D. The ArcViewCorba Corba interface.

We can convert the Java application into a Java applet, so that users can use a Corba-enabled Web browser (such as Netscape Navigator 4.x, see [2] in Figure 3) to access and interact with GIS Corba objects. Future implementations on the client side will improve the client GUI and include more types of clients such as a Web/Corba or C++ client. Client applications will provide users reliable and timely access to a single, virtual GIS with spatial and nonspatial information from a network of geographically distributed, physically separated servers.

The proposed system architecture is the foundation for the design and implementation of an interoperable GIS: the Spatially Enabled Fisheries and Environmental Database System (SEFEDS) at the Southeast Fisheries Science Center.[1]

Performance issues

We ran several tests to study a prototype system's performance. In one test, we deployed all Corba objects, EJBs, and spatial databases in an Oracle8i database server running on a Sun Enterprise 250 server with a single CPU at 400 MHz and a 256-Mbyte main memory. On our LAN T1 line (100 Mbits per second), it took an average of 4 seconds for the Java client application running on a number of Intel desktops to initialize a connection to the Oracle database and the CorbaObjManager Corba object deployed in the Oracle8i database. It took about 2 seconds for the CorbaObjManager object to connect to a QueryableContainerFeatureCollection Corba object that initializes its corresponding SpLayer EJB. After that, the CorbaObjManager object returned a reference to the QueryableContainerFeatureCollection Corba object to the client that can then invoke methods of the latter Corba object. The method calls to query and iterate geospatial features of a relatively small GIS layer (less than 1-Mbyte file size) completed within a second. The performance for connections degraded moderately in the Internet environment, adding extra seconds depending on the network setup. The connections are only necessary once for each client per processing session.

In another test, we deployed Corba objects in one Windows NT server. We deployed EJBs and spatial databases in the Oracle8i database server we just mentioned, as Figure 4 shows. We only ran this test on the LAN environment, and we can compare its performance with the first test.

Our study results reveal that the prototype system's performance is adversely affected by establishing connections between the client and Corba objects, between Corba objects and EJBs in the Oracle database server, and between EJBs inside the EJB container (in our case, the Oracle database server). However, once connected, a client can invoke methods in Corba objects and obtain returned results quickly. We believe that the following measures will improve the system performance:

- Upgrade the main memory and increase the capacity and number of CPUs for the Sun Enterprise server.
- For each Corba object, create a pool of connections to EJBs in the Oracle server to reduce the need to establish a database connection on the fly.
- For each EJB, create a pool of connections to the Oracle database that stores GIS layers in feature tables to avoid frequently establishing the expensive database connection.
- Explore cache technology to reduce the frequency of accessing the physical databases.

Conclusion

GIS applications based on our architecture in its current form are interoperable syntactically. At this level of interoperability, before using a GIS layer correctly, the user must acquire knowledge on the semantics of attributes of the underlying GIS layer[12] through studying the metadata of the GIS layer. Researchers have conducted much work toward solving the GIS semantic heterogeneity.[17,18] We envision that the future OpenGIS catalog service[14] will improve our ability to resolve semantic heterogeneity in distributed GISs by enabling metadata servers to interact with one another dynamically. **MM**

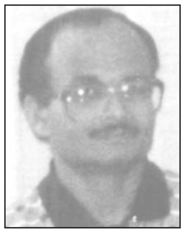
Acknowledgments

We thank Chris Snyder for his help with our Java GUI and his comments on this article. We also appreciate Marinell Davis' assistance with Oracle8i. We are fortunate to have Geoff Sutcliffe providing valuable feedback on this study. We thank the reviewers whose constructive suggestions have improved the presentation and literature survey. This research was carried out in part under the auspices of the Cooperative Institute for Marine and Atmospheric Studies (CIMAS), a Joint Institute of the University of Miami and the NOAA, cooperative agreement #NA67RJ0149.

References

1. S.H. Wong, *A Corba-Based Middleware Architecture for Building Open and Interoperable Geographic Information Systems*, master's thesis, Dept. of Computer Science, Univ. of Miami, 2000.
2. M.J. Egenhofer et al., "Progress in Computational Methods for Representing Geographical Concepts," *Int'l J. Geographical Information Science*, vol. 13, no. 8, Dec. 1999, pp. 775-796.
3. *The Complete Corba/IIOP 2.5 Specification*, Object Management Group, http://www.omg.org/technology/documents/formal/corba_iiop.htm, 1998.
4. R. Orfali and D. Harkey, *Client/Server Programming with Java and Corba*, 2nd ed., John Wiley & Sons, New York, 1998.
5. *OpenGIS Simple Features Specification for Corba,* revision 1.0, Open GIS Consortium, http://www.opengis.org/techno/specs.htm, 1998.
6. *Enterprise JavaBeansTM Specification*, version 1.1, Sun Microsystems, http://java.sun.com/products/ejb/docs.html, 1999.
7. *OpenGIS Simple Feature Specification for SQL,* revision 1.1, Open GIS Consortium, http://www.opengis.org/techno/specs.htm, 1999.
8. *Oracle8i Spatial User's Guide and Reference*, release 8.1.5, Oracle, 1998.
9. J. Gaskill and D. Brooks, "Understanding ArcSDE," http://www.esri.com/software/arcinfo/arcsde/understanding_arcsde_uc20000.ppt, 2000.
10. *VisiBroker for Java Product Documentation*, Borland, http://www.borland.com/techpubs/books/vbj/vbj33/pdf_index.html, 2000.
11. Federal Geographic Data Committee, *Content Standard for Digital Geospatial Metadata*, http://www.fgdc.gov/metadata/contstan.html.
12. *An Overview of the Web Mapping Testbed*, Open GIS Consortium, http://opengis.opengis.org/wmt/wmtinsert.pdf, 2000.
13. Y. Bishr, "Overcoming the Semantic and Other Barriers to GIS interoprability," *Int'l J. Geographical Information Science*, vol. 12, no. 4, June 1998, pp. 299-314.
14. *OpenGIS Catalog Interface Implementation Specification*, revision 1.0, Open GIS Consortium, http://www.opengis.org/techno/specs/99-051.pdf, 2000.
15. *OpenGIS Web Map Server Interfaces Implementation Specification*, revision 1.0.0, Open GIS Consortium, http://www.opengis.org/techno/specs/00-028.pdf, 2000.
16. C.S. Horstmann and G.Cornell, *Core Java*, Sun Microsystems Press, Palo Alto, Calif., 1998.
17. T. Devogele, C. Parent, and S. Spaccapietra, "On Spatial Database Integration," *Int'l J. Geographical Information Science*, vol. 12, no. 4, June 1998, pp. 335-352.
18. R. Laurini, "Spatial Multi-Database Topological Continuity and Indexing: A Step Towards Seamless GIS Data Interoprability," *Int'l J. Geographical Information Science*, vol. 12, no. 4, June 1998, pp. 373-402.

Steven H. Wong is a physical scientist at the Southeast Fisheries Science Center of NOAA Fisheries. His research interests include object systems and architecture and applying geographic information systems and remote sensing to the study of marine fishery ecology. He has a BS in marine geology from the Shandong College of Oceanography, China; an MS in marine geology from the University of Miami; and an MS in computer science from the University of Miami, Florida.



Dilip Sarkar is an associate professor of computer science at the University of Miami, Coral Gables. His research interests include design and analysis of algorithms, parallel and distributed processing, middleware and Web computing, multimedia communication over broadband and wireless networks, fuzzy systems, and neural networks. He has a BTech in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur; an MS in computer science from the Indian Institute of Science, Bangalore; and a PhD in computer science from the University of Central Florida.

Steven L. Swartz is the director for the Protected Species and Biodiversity Division, Southeast Fisheries Science Center of NOAA Fisheries. His current research interests include marine mammal studies and applying information technology to marine fisheries studies. He has a BA in biology from the University of California, Santa Barbara, and a PhD in marine science from the University of California, Santa Cruz.

Readers may contact Wong at the Southeast Fisheries Science Center, NOAA Fisheries, 75 Virginia Beach Dr., Miami, FL 33149, email Steven.Wong@noaa.gov.