Designing of Multi-tier GIS Architecture for Distributed Network Environment

†‡YOSHINORI NIE, †‡MORIKAZU NAKAMURA
†‡HAYAO MIYAGI and †KENJI ONAGA
†Okinawa Research Center, Telecommunications Advancement Organization
1 Asahi-machi, Naha, Okinawa, 900-0029, Japan
Phone +81-098-862-3986 Fax +81-098-862-3989
E-mail nie@naha.tao.go.jp
‡University of the Ryukyus
1 Senbaru, Nishihara, Okinawa, 903-0213, Japan

Abstract: This paper presents a multi-tier GIS architecture to adapt to large-scale distributed networks and to improve data transfer performance with intelligent caching technique. We design this system using UML based on object-oriented analysis. We show some advantages in our proposed system against the ordinary GIS, in special, suitability to distributed networks.

1 Introduction

As Geographical Information Systems (GIS) have been growing in popularity, the needs for sharing spatial data in a distributed network computing has become necessary recently [1].

On the other hand, weakness of traditional GISs has also been highlighted: Most GIS softwares are not good at sharing spatial data in the distributed network since ordinary softwares treat the data as a collection of flat files whose size is relatively large to be transferred. Besides, because such a file is arranged based on (2-dimensional) mesh it is difficult to rearrange dynamically contents of a file according to the user's request. Therefore, the file-based transfer is not flexible.

Our approach to this problem includes the followings: (1) Defining the smallest transfer unit called "feature" which includes the information of the earth point in the real world. (2) Designing 3 parts of layer: the presentation, function and data layer, which is suitable for the distributed network environment. (3) Developing a traffic reduction technique among layers.

Instead of transferring flat files, utilizing feature object format allow us to treat spatial data flexibility. In a prototype of this system, each layer employs the same cache function, which is specialized and optimized for geospatial data.

We design this system using UML (Unified Modeling Language) [3] by object-oriented analysis. The UML is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems which was led by Rational Software's researcher.

Let us now explain our system characteristics.

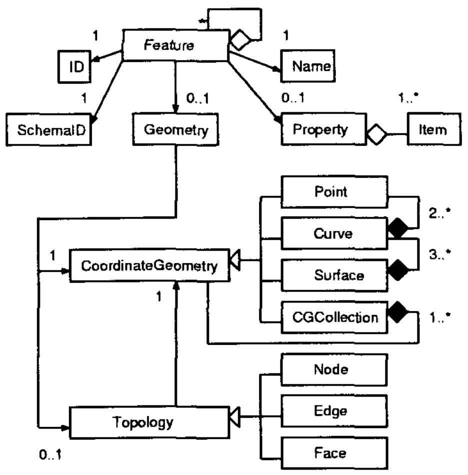


Figure 1: Feature Class

2 Feature Model

By referring to the object design of Open Geodata Model (OGM) proposed by Open GIS Consortium [2], we define the feature object, which is the fundamental unit presenting geospatial information. It is composed of the coordinate geometry object and some properties. A feature may be defined recursively, this means that it can be a composite of other features. Figure 1 illustrates simply the feature architecture.

A *geometry* expresses the geospatial information and is represented by a set of Point, Curve, Surface objects. A Surface is composed of some curve objects and a curve is constructed by some point objects. Namely, a point object is the fundamental one of the coordinate geometry. Figure 2 illustrates the geometry class and 3 depicts the coordinate geometry class.

A *property* is a set of item objects which are composed of a pair of key and value. In most cases, the key and value are expressed by character string.

In addition, a feature has the ability to move among layers through network transparently. This concept provides us with more flexibility for transfer control of both queries and size. Moreover, the feature is designed as an

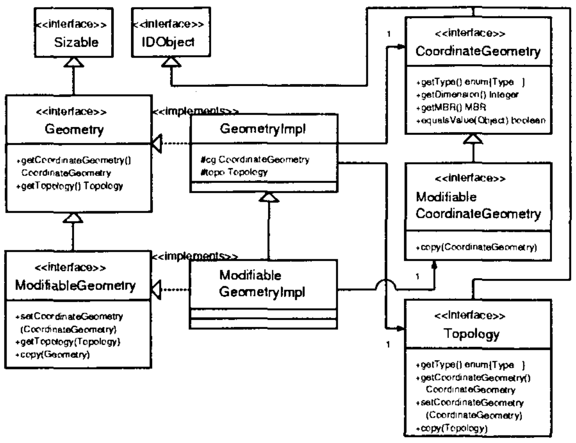


Figure 2: Geometry Class

immutable object in order to process a lot of requests from connected upper layers. An immutable object is the object that has no method to change its attribute, that is, the only chance to set geometry and property data of the feature is the time to initiate it.

This characteristic seems to be inconvenient, but it allows us to neglect complicated mechanism for the object synchronization when the multiple requests to modify attributes of an object are taken place in distributed network environment. Instead of providing methods to modify feature attribute, we introduce the modifiable feature object, which can be constructed by the corresponding user's request.

3 Multi Layer Architecture

Our GIS adopts the multi layer architecture in the base system. Figure 4 illustrates the component of this system.

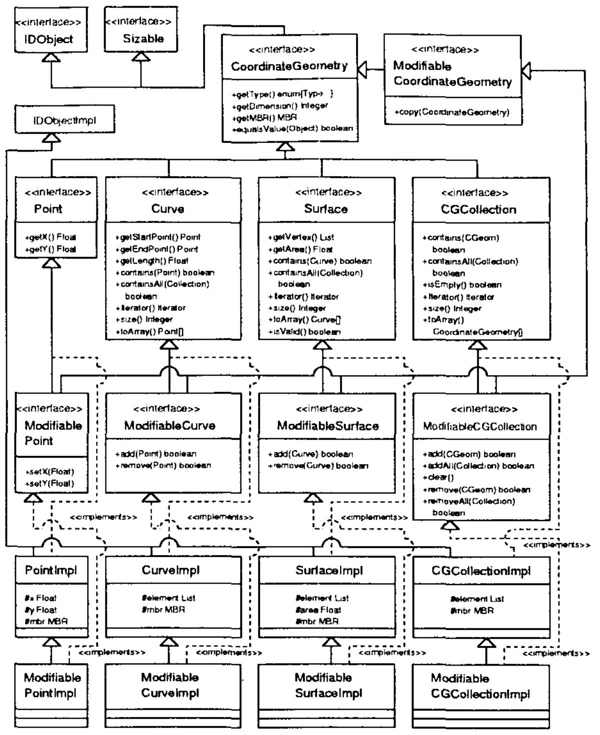
We provide a simple geospatial data viewer as a presentation layer's application. The Feature Transfer Service component enables to connect recursively another Feature Transfer Service. And it has its own DBManager component. Figure 5 depicts the deployment of this system.

Each layer's role is as follows:

Presentation Layer The presentation layer has a role of communicating with users, that is, displaying geospatial objects and waiting user's requests.

Function Layer The function layer, playing roles of various geospatial processing as the middle layer of the system, is constructed by connecting recursively several function sub-layers. Each function layer has the pointer to the following lower layer and its own database system. Figure 6 illustrates the function layer class.

Data Layer The database system generally works as a cache manager, but when no following layer exists, this function layer functions as the data layer in this system. This cache manager stores the feature object in the cache space to reduce the network traffic. Figure 7 depicts the data layer class.



<<interface>> IDObject
<<interface>> Sizable
<<interface>> CoordinateGeometry
<<interface>> Modifiable CoordinateGeometry
IDObjectImpl
<<interface>> Point
<<interface>> Curve
<<interface>> Surface
<<interface>> CGCollection
<<interface>> Modifiable Point
<<interface>> ModifiableCurve
<<interface>> ModifiableSurface
<<interface>> ModifiableCGCollection
<<implements>>
PointImpl
CurveImpl
SurfaceImpl
CGCollectionImpl
Modifiable PointImpl
Modifiable CurveImpl
Modifiable SurfaceImpl
Modifiable CGCollectionImpl

Figure 3: Coordinate Geometry Class

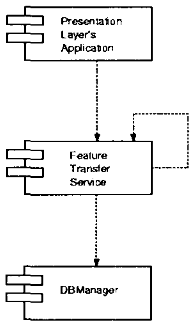


Figure 4: Component of the system

The Feature Transfer Service component refers to the lower layer of another Feature Transfer Service component. And, each one has the DBManager component. Especially, the DBManager component managed by the Presentation layer and the Function layer works as the cache system of the feature objects. The layer with no connection to lower layer of the Feature Transfer Service component is defined as the Data layer.

When the Feature Transfer Service received the request message from the user, it querys its own DBManager to get the requested feature objects. In case of finding no objects, the request message is sent to the

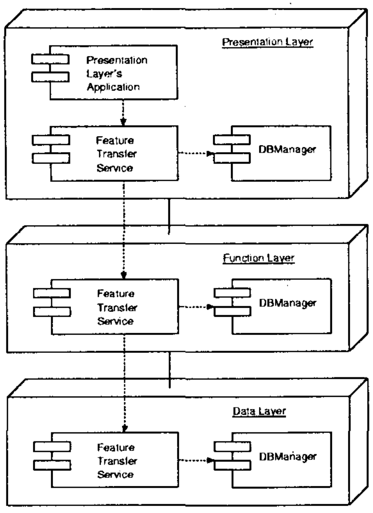


Figure 5: Deployment of the system

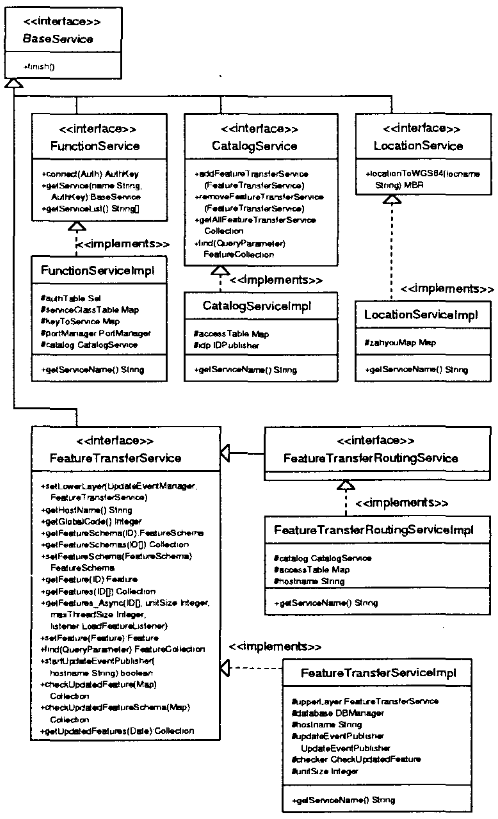

<<interface>>
BaseService
+finish()
<<interface>>
FunctionService
+connect(Auth) AuthKey
+getService(name String, AuthKey) BaseService
+getServiceList() String[]
<<implements>>
FunctionServiceImpl
#authTable Set
#serviceClassTable Map
#keyToService Map
#portManager PortManager
#catalog CatalogService
+getServiceName() String
<<interface>>
CatalogService
+addFeatureTransferService (FeatureTransferService)
+removeFeatureTransferService (FeatureTransferService)
+getAllFeatureTransferService Collection
+find(QueryParameter) FeatureCollection
<<implements>>
CatalogServiceImpl
#accessTable Map
#idp IDPublisher
+getServiceName() String
<<interface>>
LocationService
+locationToWGS84(locname String) MBR
<<implements>>
LocationServiceImpl
#zahyouMap Map
+getServiceName() String
<<interface>>
FeatureTransferService
+setLowerLayer(UpdateEventManager, FeatureTransferService)
+getHostName() String
+getGlobalCode() Integer
+getFeatureSchema(ID) FeatureSchema
+getFeatureSchemas(ID[]) Collection
+setFeatureSchema(FeatureSchema) FeatureSchema
+getFeature(ID) Feature
+getFeatures(ID[]) Collection
+getFeatures_Async(ID[], unitSize Integer, maxThreadSize Integer, listener LoadFeatureListener)
+setFeature(Feature) Feature
+find(QueryParameter) FeatureCollection
+startUpdateEventPublisher(hostname String) boolean
+checkUpdatedFeature(Map) Collection
+checkUpdatedFeatureSchema(Map) Collection
+getUpdatedFeatures(Date) Collection
<<interface>>
FeatureTransferRoutingService
<<implements>>
FeatureTransferRoutingServiceImpl
#catalog CatalogService
#accessTable Map
#hostname String
+getServiceName() String
<<implements>>
FeatureTransferServiceImpl
#upperLayer FeatureTransferService
#database DBManager
#hostname String
#updateEventPublisher UpdateEventPublisher
#checker CheckUpdatedFeature
#unitSize Integer
+getServiceName() String

Figure 6: The function layer class

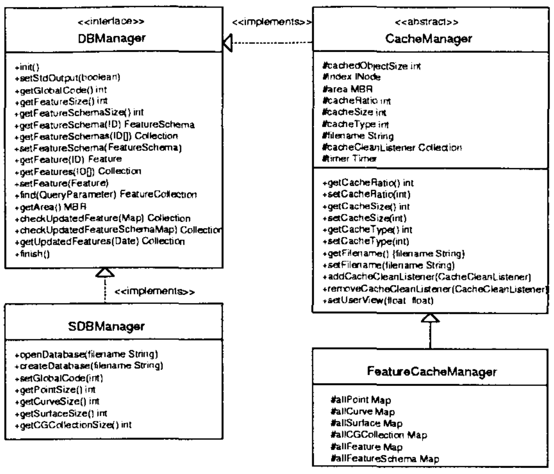


Figure 7: The data layer class

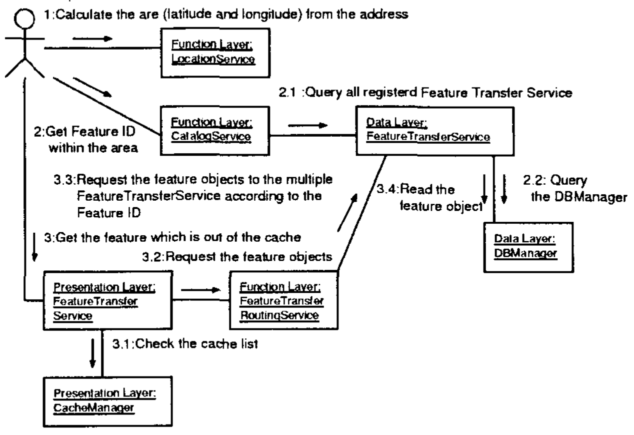

1:Calculate the are (latitude and longitude) from the address
Function Layer: LocationService
2.1 :Query all registerd Feature Transfer Service
2:Get Feature ID within the area
Function Layer: CatalogService
Data Layer: FeatureTransferService
3.3:Request the feature objects to the multiple FeatureTransferService according to the Feature ID
3.4:Read the feature object
2.2: Query the DBManager
3:Get the feature which is out of the cache
3.2:Request the feature objects
Data Layer: DBManager
Presentation Layer: FeatureTransfer Service
Function Layer: FeatureTransfer RoutingService
3.1:Check the cache list
Presentation Layer: CacheManager

Figure 8: Collaboration of reading feature

lower Feature Transfer Service until the requested feature is found.

4 Update Services

We equip the feature update service in this system.

When someone (authorized to modify certain features) changes some attributes of the feature and publishes update transaction to this system, the updated event of the feature spreads over multi-tier network systems, and transfer to all presentation layer certainly. The presentation layer that receives the updated event can repaint the geospatial data viewer to maintain the latest information. To realize the above procedure, each layer has to be ready to receive the update event from another layer via the network using special utility that provides server function to the remote machine.

However, from the security point of view, such a server-side function may be undesirable. Therefore, a polling type method is also provided. This polling function accesses the lower layer regularly to check the exis-

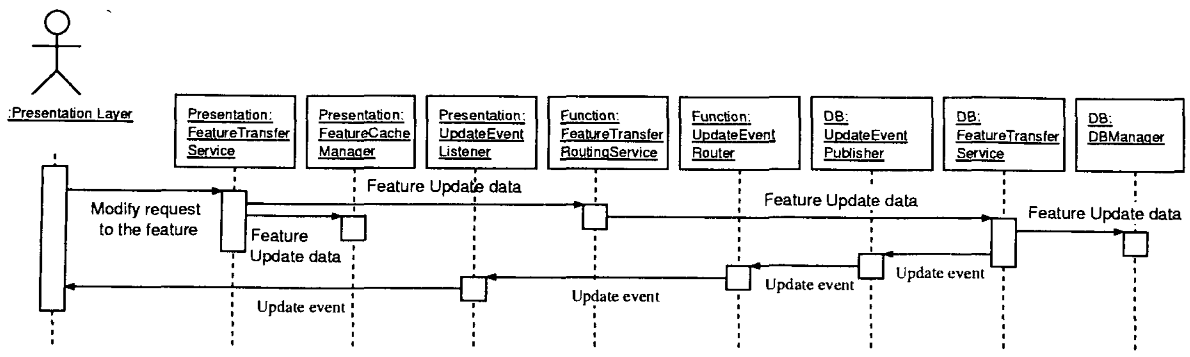

:Presentation Layer
Presentation: FeatureTransfer Service
Presentation: FeatureCache Manager
Presentation: UpdateEvent Listener
Function: FeatureTransfer RoutingService
Function: UpdateEvent Router
DB: UpdateEvent Publisher
DB: FeatureTransfer Service
DB: DBManager
Modify request to the feature
Feature Update data
Feature Update data
Feature Update data
Feature Update data
Update event
Update event
Update event
Update event

Figure 9: Update event type sequence

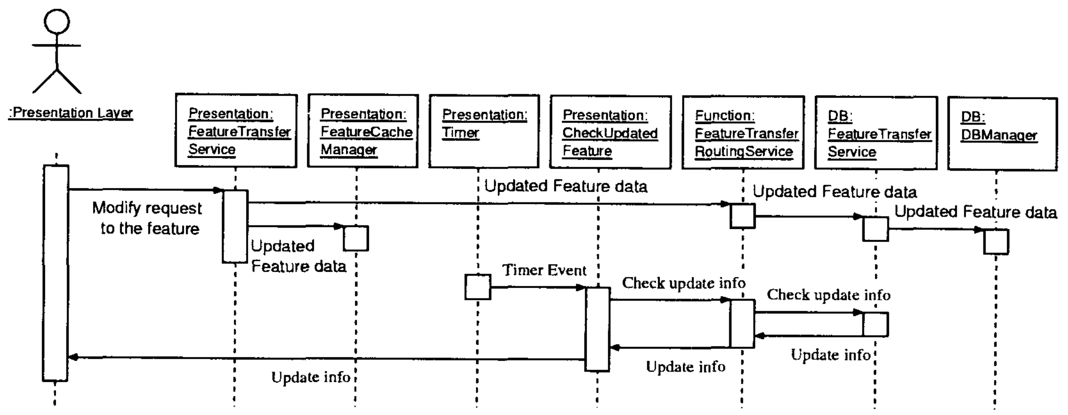


Figure 10: Polling type sequence

tence of the updated feature.

The polling type refreshes its own cached feature objects regularly, but it cannot get the new feature object which is not contained in the cache. Nevertheless, the update event type can receive any feature objects immediately, but there is a possibility of missing the update event when a layer is down for some reason or other. By employing the two methods concurrently for updating, the reliability is increased.

5 Conclusion

In this paper, we have proposed a network centric GIS architecture with the object-oriented technology to share effectively the spatial data in distributed networks.

The proposed system is implemented with Java language. Each layer interface is available by the technology of Java RMI (Remote Method Invocation)[4] included in the Java Development Kit (JDK). In Java RMI, a network transparent object is easily defined by adding a few steps into the original object definition. The database including the function layer can store feature objects as persistent objects with the simple hash table structure. This approach makes development cost decreased.

We showed some advantages in our proposed system against the ordinary GIS, in special, suitability to distributed networks. The UML diagram helps us to understand class structure and message sequence clearly.

References

[1] NSDIPA Web Site, http://www.nsdipa.gr.jp/, May. 2000.

[2] OGC Web Site, http://www.opengis.org/, May. 1999.

[3] Object Management Group, "OMG UML Ver 1.3 specification", http://www.omg.org/, March, 2000.

[4] Sun Microsystems, "The Source for Java Technology", http://www.java.sun.com/, March, 2000.