A CORBA infrastructure to provide distributed GPS data in real time to GIS applications

P. R. Muro-Medrano
D. Infante
J. Guillo
J. Zarazaga
J. A. Bañares

Computer Science and System Engineering Department
Centro Politécnico Superior
Universidad de Zaragoza
María de Luna 3
50015 Zaragoza, SPAIN

{prmuro,dinfante,jguillo,javy,banares}@posta.unizar.es
http://diana.cps.unizar.es/iaaa

Abstract

This paper shows a distributed object-oriented architecture to provide GPS data to GIS applications. Data captured in real time by GPS units are sent via radio to the computer. Several sources of this real data in addition to simulated GPS data are distributed in a computer network. These servers of GPS data can be accessed by distributed client applications. A CORBA based infrastructure provides the integration and distribution mechanisms. Client applications range from simple GUIs for GPS remote control to GIS applications used for automatic vehicle monitoring.. The basic components to provide the radio communications and GPS data servers are presented in detail.

1.Introduction

The reduction in price of radio communications together with steps forward in the field of data captured in digital format (using handheld computers in the fields, and GPS units), and the development of the distributed object-oriented technology are the technologies that have revolutionised the possibilities of GIS applications functionality. Automatic vehicle monitoring (AVM) is a prototypical application that requires the integration of mentioned technologies: Radio communication that provides wireless interaction between different devices, communication software, Global Positioning Systems (GPS) [5], GIS, access to databases to integrate data with non-spatial data, etc. [6]. These technologies enable the acquisition of vehicle locations in real time and the visualisation of vehicles in a map.

The need to optimise service costs, the increase of rivalry in the transport sector, and the interest of public institutions to promote public transport have persuaded many enterprises to integrate tracking and fleet control systems with their information systems. Fleet management aid systems that incorporate these technologies permits the recompilation of real operation parameters. They offer users a real time information that allows incident control and a valuable information for end-users of public transport [1].

Although AVM applications require a set of common services (acquisition, communications, GIS, ...), requirements and needs of each client are very different in resources and functionality, and therefore, they need "ad hoc" solutions. The work in this domain with new software technologies is just beginning [3]. However, the cost of these "ad hoc" solutions may only be affordable by the design of a flexible architecture that makes use of new software technologies, such as distributed object-oriented systems, and where features as interoperability and reuse are appropriately considered. In this sense, it is required that Geographic Information Systems (GIS) provide much more than a map to provide the framework for all the major functionality of an application. Applications must be able to transform themselves intelligently by reusing useful parts and incorporating new technologies to extend their capabilities [2].

This paper shows the basic ideas of OODISMAL; an object oriented distributed information system for mobile automatic location. OODISMAL provides the basic components to integrate radio communications and real-time data captured by GPS units with GIS components. These components may be easily adapted to any kind of radio or sensor to capture data. Firstly, the technological approach to develop the OODISMAL architecture and its basic components is explained. Following, design decisions and a detailed explanation of modules and operation of main components is given. In §3 the radio component server that provides to all computers in the local net the functionality of a trunking radio connected to a PC. In §4 the data acquisition component that uses the radio to extract from received messages the data captured by different GPS units. The integration of the radio and data acquisition components with GIS to develop final applications is detailed in §5. Finally, in §6 the conclusions are presented.

2. OODISMAL Architecture

2.1 Technological approach

The technological approach that has been adopted in this work is based on the new technologies of distributed object-oriented systems. The object model makes easier the co-operative work and the reuse by means of encapsulation. However, it is not sufficient. The fast evolution of new technologies suppose the cohabitation of a great diversity of machines, operating systems and programming languages that makes difficult the maintenance of systems. It is important to make compatible

new applications with applications that use old and heterogeneous technologies. The present tendency to afford these difficulties is the development of distributed client-server applications using component software technology [7], [10].

Component Software technology makes possible the development of light applications where it is solved the specific problematic of the application, whereas the basic functionality is provided by reusable components. A distributed client-server application, where clients and servers may be resident in the same or different machines, makes possible to increase the system functionality without the modification of reused parts. Servers are used only when it is necessary, and may be shared by several light remote or local clients.

Our research group has therefore been considering international computer industry standards for developing distributed object-oriented systems. In particular the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) [9]. CORBA has been used as the middleware infrastructure to provide interoperability and to distribute functionality. In CORBA, applications are viewed as objects, and their functionality is provided through their object interface, no matter they are programmed in an object-oriented language [VIN95]. From the implementation point of view, clients of a remote CORBA component deal with them as local objects. CORBA components may be running in the same or in different computers in a transparent way.

It has been necessary to consider other component technologies because many commercial components often use the Microsoft's Distributed Component Object Model (DCOM), which is the de facto "other standard". Finally, we have also considered Java, the third component technology to be considered, because it supposes the perfect complement to CORBA. CORBA offers an infrastructure that provides interoperability, and Java provides an infrastructure for mobile code. In this way we may do CORBA ubiquitous in the net [8]. The main advantage of this language is that a Java program may be easily transformed in a Java applet that may be downloaded in any computer connected to Internet. It makes Java the best option to develop light client that provides portable GUI to access the functionality of server components.

2.2 OODISMAL Overview

OODISMAL is an information system that is composed of a set of distributed components in a LAN. These components interoperate between them using the CORBA infrastructure, and provide the basic services for location data acquisition, radio telecommunications, and the storing and processing of acquired data. If every object were a CORBA object it would suppose an unnecessary overhead of communications and would produce coupled components. Consequently, each component interface is defined clearly by a reduced number of CORBA objects. Each component works as a server of its interface CORBA objects and uses, as a client, the CORBA objects that needs from other components.

In object-oriented systems, components typically interact with each other by explicitly invoking their services. OODISMAL also uses an event based, or implicit invocation mechanism. The idea is that instead of invoking a procedure directly, a component can announce one or more events. Other components in the system can register an interest in the reception of an event. When the event is announced, the registered component is notified and executes some service.

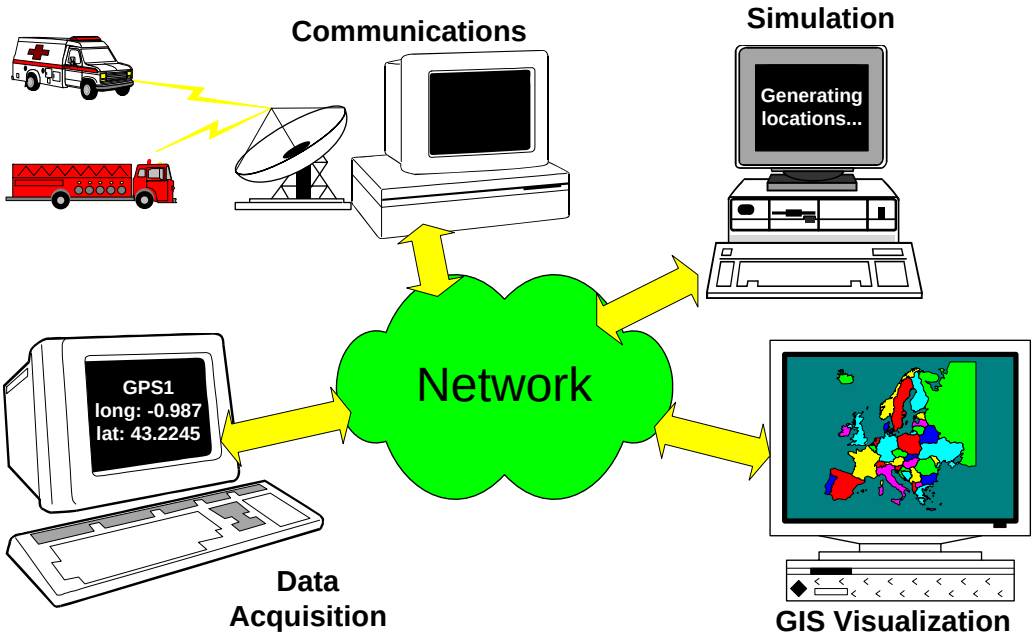

Figure 1. OODISMAL system

Figure 1 shows OODISMAL as a distributed system with the basic components that interoperate using CORBA to visualise in digital maps real-time location of vehicles. The working of these components is orchestrated by the reception of radio call with GPS data information, and their responsibilities are the following:

- The *radio component* works as a CORBA server that offers to the CORBA bus the messages received by the

radio connected to a PC in the LAN. It also offers the functionality to make different kind of call through the radio.

- The *data acquisition component* works as a client of the radio component to receive data from remote GPS, and as a server that offers location information to the CORBA bus.
- The *simulation component* allows the developer to tune the system simulating the reception of messages with GPS information. It has the same interface that the data acquisition component. In this way its clients do not distinguish real form simulated data. The only difference is that simulated location data are taken form a database through the persistence component.

Functionality of these components is offered to the CORBA bus. So, any new client application may be introduced in the system simply by registering it for the events provided by components. CORBA supports interoperability, and its client/server style provides strong support for reuse. For example, the simulation component and the acquisition component may be interchanged without affecting its clients because they offer the same interface. Enhanced versions of these components may be developed without affecting client applications. When a vehicle tracking application initiates, it may look for CORBA servers that provides vehicle locations in the LAN, and can choose any of them.

Previous functionality may be easily integrated in a GIS framework. It allows the visualisation in digital maps of real-time vehicle locations provided by the data acquisition component, the access to the radio of visualised vehicles, or the visualisation of analysis results of routes recorded in a database.

Finally, computers in which components reside may offer light Java clients that may be downloaded from any computer connected to Internet. It allows the user the supervision of components. For example, it is possible to access data acquisition components and show in a window the last data received from each GPS. A simple version of the tracking application has been developed in Java. It allows Internet users the visualisation in digital maps of real-time locations.

3. Radio Component

This component has the responsibility of handling and making accessible, to several clients, all radio typical features using a narrow and well-defined interface of allowed operations. Those features include communicating information and establishing different kinds of calls using the radio physic communication channels, such voice channel, data channel and control channel. The flow of information between the clients of this component and the radio device is in two senses. On one hand, some clients may ask to send a message, which is coded and sent through the serial port to the radio device. On the other hand, other clients may register in the component to be notified when a message arrives, and to receive it decoded.
The choice between different radio devices depends on different factors such as price and functionality. We consider trunking radiotelephony the best option due to its fixed cost (it is not charged by the number of calls, the working time, or the number of transmitted data). The component has been developed for a T500 voice terminal with GPS option developed by Teltronic. The integration of the GPS inside the radiotelephone reduces the dimensions and prize, and offers a better portability. In any case, different kind of radios provides different kind of capabilities, but a great number of this features are common to all radio devices and these are the operations offered by the radio component. This allows creating different instances of the radio component using different kind of radio devices, but maintaining the interface, so the same capabilities are offered device independent.

3.1 Radio CORBA Interface

The radio component has been built as a CORBA server. In this case, it is so important to reuse the radio device as the software component. By providing a CORBA server that controls the radio, it is possible to access to the same radio from every computer that is connected to the network. The entire interaction with the radio component is made through several CORBA interface objects, as illustrated in figure 2.

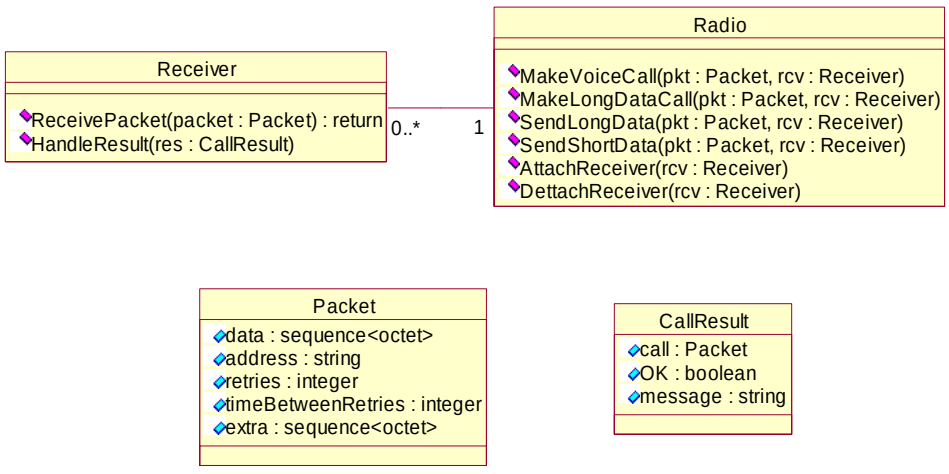

Figure 2. Radio Interface Objects

The *Radio* CORBA object is the interface to the radio component. It has services for making calls and registering clients for receiving incoming calls. The *Receiver* CORBA object is an interface through which the component will communicate with the client. A client must implement a Receiver object in order to use the radio component. The client can use a Receiver to interact with the radio component in two different ways. On one hand, the client may specify a Receiver when it makes a call, in order to get the result of that call through the Receiver. On the other hand, a client may register a Receiver in the component to receive any incoming message. The client can implement the Receiver to do whatever is necessary when this information arrives. In this way, any component or application which implements a Receiver can be a client of the radio component.

Both *Packet* and *CallResult* are CORBA data objects; that is, they contain data but no services. They are used to send packed data between the Radio and the Receivers. The *Packet* object contains data for either incoming or outgoing calls. Therefore, it is used to specify a call that a client wants to make and it is also used to send incoming messages to the Receivers. The *CallResult* objects contains information about whether a call has been sent successfully or not.

3.2 Modules and Operation of the Radio Component

Figure 3 shows the modules and operation of the radio component. The *Clients Managing module* offers the *Radio* CORBA object, which has been explained above.

The process for making a call is the following. A client requests a radio service through the Radio object. The client must send to the radio a Packet containing the data and destination address of the call. A Receiver object may also be specified if the client wants to receive the result of the call. As the *kernel module* receives the user request, it builds the corresponding radio call. Then, the built call is sent to the radio device by a COM port using the low-level radio communication protocol and the device makes the call. The *low-level protocol module* implements the access to the serial port and the low-level communication protocol of the specific radio device. At the present, two protocols are implemented, one for trunking radios and another one for 40 MHz radios.

This simple operation lacks for robustness when used intensively, because some calls may not have been made due to different problems such as channel occupation, or the disconnection of the called device. If this happens, the call would be lost. In many situations, a call loss can not be afforded. To solve this problem a scheduling system has been incorporated into the *kernel* module. The client can specify, in the Packet object, how many times the component is going to try to send the call, and the time between these retries. The *scheduler* has the responsibility for queuing the calls, scheduling them according to the timings specified for each call, and trying to send each one as many times as the client programmed. When either the call has been successfully sent or it has used up all its retries, the component will send a CallResult object to the Receiver specifying the result of the call.

On the other hand, if a client wants to receive any incoming call, it must register a Receiver in the radio component. This behaviour is based on the Subject-Observer pattern [4]. This mechanism defines a one-to-many subscription relationship between components, so that when one component changes state, all its subscribed components are notified and updated.

Following this pattern, a client registers a Receiver in the component through the Radio object. From that moment on, the Receiver will get every incoming message. When a message is received from the *low-level protocol module*, the *kernel* extracts its data and source address. The *kernel module* will build a *Packet* object with all this information and send it to all the registered Receivers. This way, the client does not have to worry about asking the component for messages. Whenever a message arrives, the client will be notified through its Receiver object. Note that every Receiver will get every message, so clients must filter those messages they are interested in.

In order to allow users to monitor the component working, a basic *GUI* is provided. It shows the incoming and outgoing messages in a text window, along with other relevant information. In the following section it is shown how to use this component to develop specialised data acquisition components, such as real time location of several vehicles that incorporate a radiotelephone with an integrated GPS.

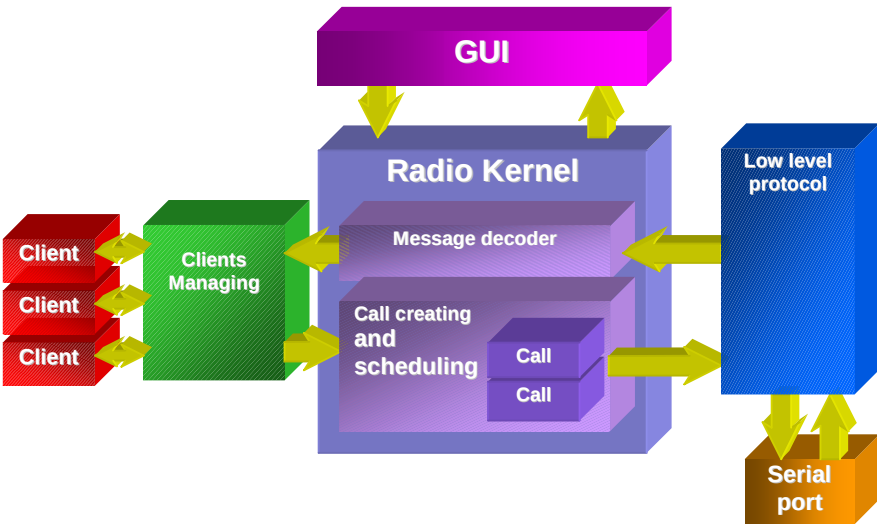

Figure 3. Modules of the Radio Component

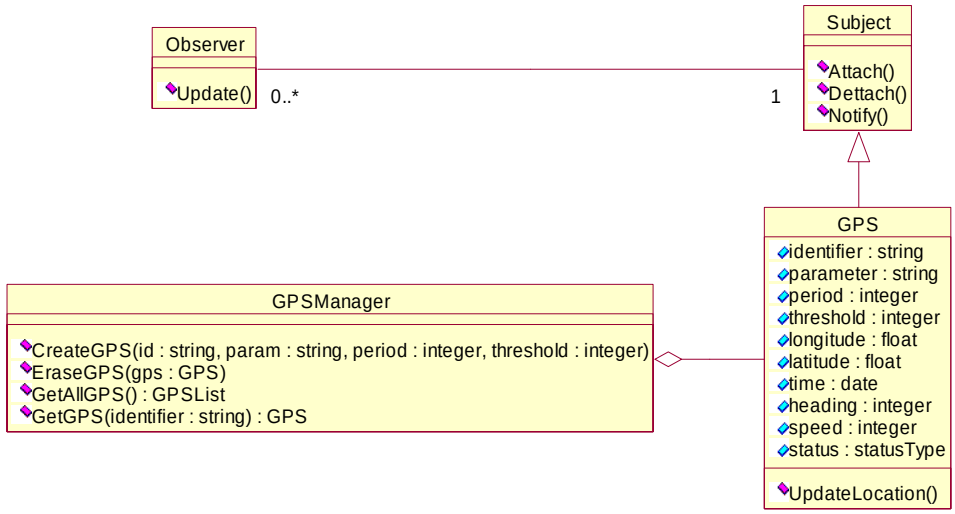

Figure 4. GPS Interface Objects

4. Data Acquisition Component

The GPS component has the responsibility to offer location information of several vehicles, in which a radio and GPS device has been installed, using a real time communication mechanism. The GPS component uses the radio component described in the previous section to make and receive the radio calls. These physical radio calls are transformed in location and time information that can be used by the system. The basic information is composed by position, given by latitude and longitude values. More complete information can be offered. This information can include velocity, bearing, and GPS signal status, as required by clients.

4.1 Acquisition data CORBA Interface

The GPS blocks are the logical representation of the GPS devices. GPS blocks are composed by the CORBA interface objects illustrated in figure 4. Every GPS device that needs to be tracked has a counterpart *GPS object* in the component. This way, clients of the data acquisition component can use all the functionality of the remote GPS devices through these objects. There is also a *GPS manager* which allows clients to access all the GPS objects in the component, create new objects to represent other GPS devices and delete existing objects which do not need to be used any more.

GPS objects contain the last location received from the physical device. Clients can register in any GPS object so that they are notified when a new location is received, following the subject-observer pattern. To achieve this, clients must implement one or more *Observer* objects. The generic *Subject* object provides the mechanisms to register Observers in an object. The GPS object inherits this behaviour from Subject. The client can retrieve a complete list of the GPS objects in the component from the *GPS manager*. Once the Observer is registered in the GPS, the pattern starts to work. Each time the GPS's position changes, the Observer is notified, so it can perform the task it is programmed to. It can be built a lot of kinds of clients, clients that prints textually the information on a screen, clients that displays the position graphically, or even clients that makes calculus or stores information treating it in some way.

Additionally, some clients may be interested in accessing to the last location of a vehicle when they are interested, therefore they may not require a subscription to the event notification mechanism provided by the GPS component. Of course, these clients do not need to implement any Observer object.

4.2 Modules and Operation of the Data Acquisition Component

Figure 5 provides a complete overview of the component's work and general organisation. GPS devices can be configured to send their location every time a given period is elapsed or its location changes in a given distance, which is called threshold. The GPS objects have services to access this functionality. The configuration of every GPS object in the component is stored in a database. When the component is initiated, this configuration is retrieved from the database and sent to the GPS device, so that its behaviour is always the same.

The GPS objects need to communicate with the physical devices using the radio component for configuring them and receiving their locations. This is achieved through the *message manager* module, whose structure is illustrated in Figure 6.

The *message manager* has the responsibility for dealing with all the messages which go to and come from the radio component. The message manager inherits the behaviour of the *Receiver* interface (explained in the Radio Component). In this way, it is able to register itself as a client of the radio component to receive incoming calls using the mechanisms explained earlier.

Note that a Receiver registered in the radio component will receive every call that is sent to the radio device, and these may include unwanted messages. By centralising the reception of messages in the message manager, the messages can be easily filtered and then dispatched to the appropriate GPS object if they are true GPS messages.

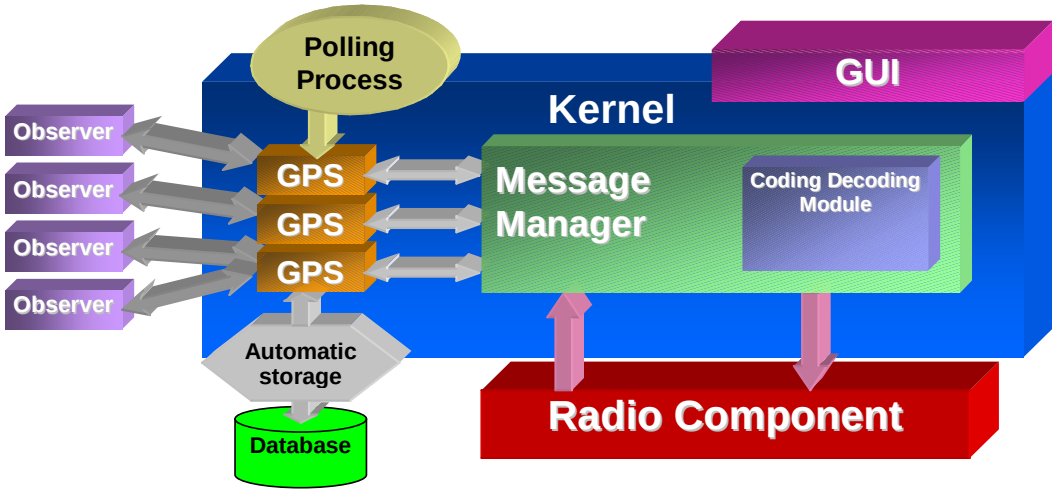

Figure 5. Data Acquisition Component

When a GPS message is received, the message manager verifies whether the source address matches any recorded GPS in the component. If so, the *coding/decoding module* extracts the location information from the binary message and updates the information contained in the GPS object. Then, the GPS notifies its observers that its information has changed, so they can get its new location.

The message manager is also used to build and send any message that the component needs to send to a GPS device, for configuring it or asking its position. First, the message manager asks the coding/decoding module to build the necessary message, and then uses the radio component to send it to the GPS device. Because the message managing is centralised it is very easy to adapt the component to another kind of GPS devices that may understand a different set of messages. Most times, it is only necessary to change the coding/decoding module. Even if the message manager had to be modified, the rest of the component would remain unchanged because it interacts through a *generic* message manager.

The above explanation covers the kernel capabilities of the data acquisition component, but there are other features that give the component a greater functionality. There are three additional features in the component: the *automatic location storage*, the *stop detection* and the *polling process*.

The two first features have been implemented as small modules inside the kernel, expanding its functionality. The *automatic storage module* is integrated within the GPS objects and the GPS manager and stores in a relational database all the locations received by every GPS. Each object has an associated table in the database, where it stores each position it receives. The associated table is changed everyday, so the module generates a table for each GPS and day. Besides, the tables have a limited lifetime, so they are deleted after a specific number of days. This way, the database is not saturated, and there is no unnecessary waste of hard disk space. The user can configure this lifetime, as well as the hour for table change.

The *stop detection module* detects when a vehicle has not moved during a given time and activates a flag in the GPS information so this can be showed in a GIS using a special colour. It is also stored in the database so that it can be checked later.

The *polling module* is separated from the kernel and implements a special process for obtaining the locations from the GPSs. Instead of configuring the GPSs to automatically send their location, the polling module asks the location of each GPS in turn. This way, there is no saturation in the radio network and no collisions between radio calls. The polling process can be configured for every GPS in the component, or only for a few.

Finally, the *GUI module* allows an external user to monitor and configure the different features of the component through a graphical interface. The GUI module is a Java light client that may be downloaded to any computer connected to Internet.

As it has been seen, the acquisition component is a CORBA server that makes the location information accessible through the CORBA bus. Several final applications may use location information offered by this component for different purposes, such as fleet control, offering users the locations of vehicles in a map through Internet, etc. It may also be possible to integrate directly this component in final applications without a distributed infrastructure. This may be decided due to security and efficiency. There are a number of linked references between this component and its clients, raising the number and length of network communications, making it a little less efficient.

This component is only an example of a radio client, specialised in dealing with position calls. However, there could be other components that would act as radio clients that could observe and filter other kind of calls, like alarms, input/output systems and so, in addition to the use of the radio for voice calls.

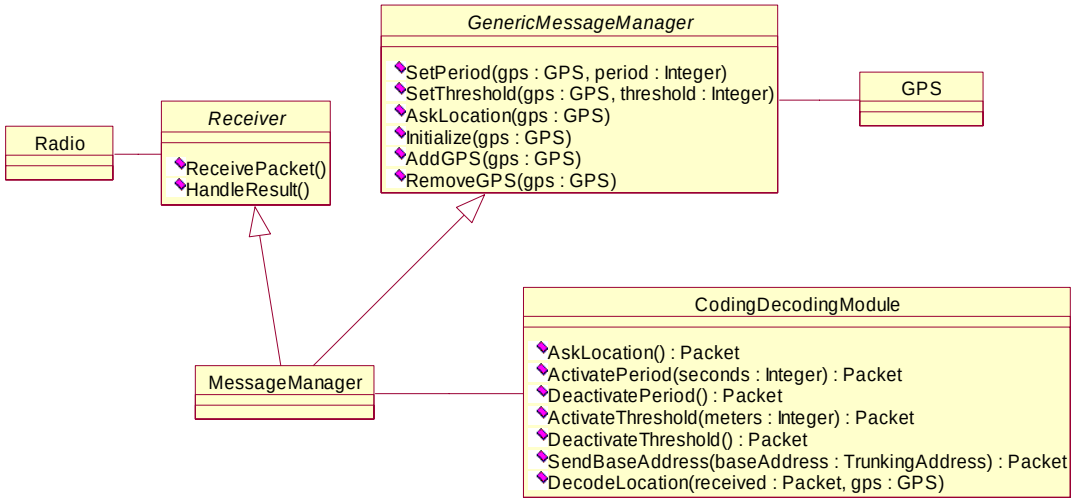

Figure 6 Messaging Objects

5. Final Applications

The first domain component developed to provide integration of basic services according to more specific clients has been an AVM application. This component integrates the functionality of all previous basic components to perform vehicle tracking and to show the locations of vehicles in real-time. The integration of functionality allow us to perform a radio voice call selecting a vehicle on the map, to show the location of the vehicle in real time, or to graphically visualise the result of any route analysis. A lot of final applications can be built using the same hardware and software set-up.

In order to show how previous components are orchestrated to provide required functionality let us show how they are incorporated in a final application, and how it may be configured. Firstly, the best framework to provide all the major functionality of the kind of applications we are dealing with is a geographic information system (GIS). For example, the graphic visualisation of a fleet of vehicles' locations and the capability of interact with it, is one of the most obvious and attractive functionality of a final application.

Therefore, the visualisation component has two main responsibilities: On one hand it has to provide GIS capabilities in order to visualise vehicles over maps. On the other hand, it has to provide mechanisms to access fleet information, manage it and exploit its capabilities to the maximum, as a usual information system but enhanced with GIS capabilities. The fleet interaction capabilities, like path analysis or making voice calls to the vehicles are accessed through the GIS capabilities, being able to select vehicles by clicking on the map and displaying results in a geographical way, over a geo-referenced map.

Figure 7 shows the graphical windows to visualise a fleet. The way to incorporate into the GIS the real-time locations of vehicles has two possibilities: The first one, to use the subject-observer pattern. In this way, the data acquisition component would notify us whenever a new position received from the vehicle is processed. The second one, to periodically query the server.

A tracking vehicle component has been developed increasing the functionality of the data acquisition component. The tracking vehicle component tracks vehicles that may follow a predetermined route. Locations received from the GPS component are adjusted to the route if this route is provided. In this way, a simple correction to the GPS error is provided. More functionality may be easily incorporated. For example, some public services have an associated itinerary pattern with temporal information that may be used to make better corrections and estimations in the case GPS locations are not being received.

The interface of any enhanced component is the same of the data acquisition component explained above. In this way, they may be interchanged. Moreover, the applications may locate and connect as clients of different CORBA servers in the LAN that provide real-time locations. So, it is possible to visualise in the same map vehicles whose location is provided by the same or different radios and data acquisition components.

The rest of the functionality of the radio and data acquisition component may also be accessed trough the GIS framework. As mentioned, it allows us to perform radio voice call, or configure a GPS device selecting a vehicle on the map.
Additionally, it has been incorporated to the GIS framework a more specific functionality, which is required in applications of AVM. It is worth to point out the possibility of storing historical locations (routes) that may be analysed by an off-line route analysis component. The results of queries about these routes may be graphically displayed.

Finally, the web component offers light Java clients to configure components and to visualise any information. It is worth to note a simple Java version of the visualisation component. It only allows the visualisation in real-time the location of vehicles in digital map. Neither GIS nor the rest of previous functionality is incorporated. It uses the same mechanism to incorporate real-time locations of vehicles. In fact the component is an applet that can be loaded by any Internet browser with a Java plug-in. The CORBA infrastructure also is downloaded, therefore once it is downloaded it may attach as client of any data acquisition component.

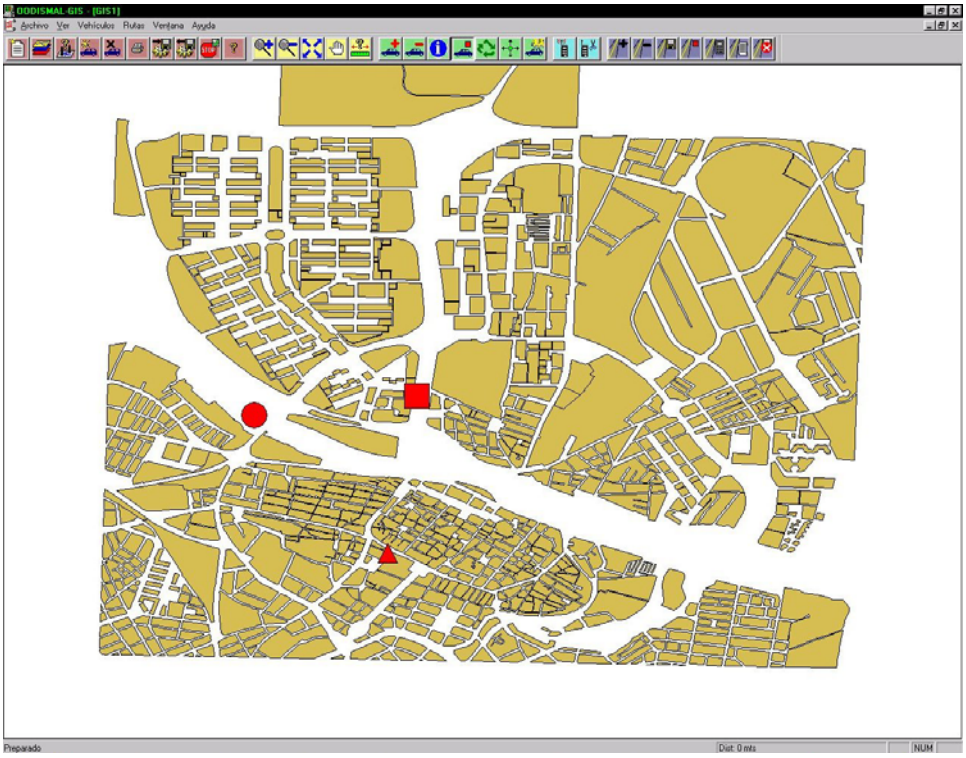

Figure 7. GIS Visualisation of a vehicle fleet

The distributed configuration of these components may be adapted to the requirements of clients. All components may be installed in a single computer, or it may be installed in different working areas. The operation of software components is independent from the chosen configuration.

6. Conclusions

In this paper, we have presented an architecture that provides the basic components to integrate the functionality of Geographic Information Systems (GIS), with the functionality of radio telecommunications. Over the radio component it is possible to build different components to capture data, such as vehicle's location acquisition in real time using GPS units. The integration of functionality of the radio and data acquisition component with GIS capabilities provide mechanisms to access fleet information, manage it and exploit its capabilities to the maximum in a synergic way. These components are the kernel of any application for vehicle tracking.

CORBA has been used as the middleware infrastructure to provide interoperability and to distribute functionality. Any application may look for different servers in the LAN that provide a predetermined CORBA interface, and use them. Moreover, it may be notified of any change of state in servers simply by registering itself for the events that they provide. For example, the simulation component and the acquisition component may be interchanged without affecting its clients; and a GIS viewer may visualise the location provided by different data acquisition components.

Java allows downloading the CORBA infrastructure. In this way Java has been chosen to develop light clients that may be downloaded in any computer connected to Internet. A simple map Java viewer that shows real-time location has been developed.

The advantages of the proposed architecture are:

- A GIS viewer provides the framework to offer all the major functionality. In this way, the functionality of other components may be graphically integrated with a digital map.
- It allows a flexible development. All components may be easily interchanged by new components with enhanced functionality (or cohabit with them). In this way it is possible to increase the functionality without the modification or previous parts.
- Components may be reused in several applications. For example, the radio component may be used by data acquisition components that filter alarm messages or data received of any kind of sensor.
- Applications may be configured in different ways according to the requirements of the client. It is not necessary any modification of components to configure different final applications. The proposed architecture is scalable according to the client's requirements.

A final application that integrates OODISMAL components to AVM has been illustrated. In the future, new data acquisition components that integrate information coming from different sensors will be developed. The OODISMAL infrastructure may be enhanced with new specialised domain server such as public urban and interurban bus transport. It implies the development of new components specialised in tickets, user information, fleet management, etc.

Acknowledgements

This work was partially supported by the Comisión Interministerial de Ciencia y Tecnología (CICYT) of Spain

through the project TIC98-0587 and by the project P-18/96 of the CONSYD de la Diputación General de Aragón.

References

[1] B. de Saint-Laurent and K. Bourée. "Eurobus Project. Final Report". DRIVE PROGRAMME, Eurobus Project V2025.

[2] L. Hecht. "GIS Helps Utilities Thrive in A Deregulated Environment". *ESRI ArcUser Magazine*. Jul-Sept.1998. (http://www.esri.com/news/arcuser/798/utilities.html)

[3] Ted Foster and Liping Zhao . "Structuring the Network Model to Service More Functions". Report to assess the impact on Transmodel Version 4.1 of some of the change request raised within the Titan Data Model Management Group. 1998.

[4]. E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Desing Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley Publishing Company. 1994.

[5] E.D. Kaplan (ed.). *Understanding GPS Principles and Applications*. Artech House Publishers. 1996.

[6] E.J. Krakiwsky. "Tracking the Worldwide Development of IVHS Navigation Systems". *GPS World*, V 4, N 10, pp. 40-47. October, 1993.

[7] R. Orfali, D. Harkey and J. Edwards. *The Essential Client/Server Survival Guide CORBA*. Wiley Computer Publishing. 1997.

[8] R. Orfali, D. Harkey. *Client/Server Programming with Java and CORBA*. Wiley Computer Publishing. 1998.

[9] Object Management Group. Object Management Architecture Guide, Version 3.0. Framingham, MA: Object Management Group, 1995.

[10] C. Szyperski. Component Software. Beyond Object-Oriented Programming. Addisson Wesley, 1997.

[11] S. Vinoski. *CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments*. *IEEE Communications Magazine*, pp 46-55. Feb. 1997.