

A Client-server Architecture for Accessing Multimedia and Geographic Databases within Embedded Systems

Alain Bouju, Arunas Stockus, Frédéric Bertrand, Patrice Boursier
University of La Rochelle - LSI - Avenue Marillac - 17042 La Rochelle - France
{alain.bouju, ...}@univ-lr.fr

Abstract
In this paper we present a software architecture that is dedicated to the development of an information system embedded in a vehicle allowing access to multimedia and geographic databases. This system provides the functionalities of a navigation system and uses a Global Positioning System for recognizing the position of the vehicle. The client-server architecture allows communication between the components of the system, as well as access to various information sources. Communications are based on the use of a cellular phone and Internet connection. Web browser and Java applets are used for information presentation and processing. The combined use of embedded local server and distant servers is one of the main characteristics of the system.

1. Introduction

An information system embedded in a vehicle can be considered as a mobile system intended to help navigation. It is important that such a system should be able to access in real time to various information sources: position of the vehicle, information about traffic, meteorological conditions, car crashes, etc. Some of these data may be stored locally. Others can come from external databases located on distant servers and must be transmitted either continuously or on request from users.

Embedded navigation systems are on the market, but most of the time, they are autonomous systems that only make use of information stored on CD-ROMs or on the hard disk of the onboard computer. They have no connection with external information sources or data servers. Some other navigation systems use specific communication and data exchange modes between a vehicle and a data server, which needs the installation of specific infrastructures.

In this paper, we present the architecture of an embedded system that allows the access to information coming from various external sources using standard technologies. Two main aspects are considered. Firstly, the architecture is a client-server one, and the system components may access in real time to distant information servers. Secondly, the system is based on widely used existing technologies. We use a satellite based Global Positioning System (GPS) to localize a vehicle. The connection between embedded systems and distant data servers is based on the use of cellular phones. At the software level, an Internet connection allows to communicate data between the system components. Web browser and Java applet visualize and process information within embedded system.

In the following section, we will present the framework of our studies and the technological choice that have been made. We will also indicate the main problems, which must be solved. In section 3 we will review various solutions to access distributed data, and in particular spatially referenced data. In section 4 we will describe the components of our system, its architecture, functioning, implementation and some results. We will finally present future developments of ongoing work in section 5.

2. Framework

2.1. Constraints and technological choices

We consider an information system embedded in a vehicle as an usual information system with some specific characteristics due to its mobility.

First, it is a navigation, positioning and guidance system, which helps the user to locate himself. Thus, spatially referenced information play an important role as they are required for the user to know the position of the vehicle on a map.

In addition, at least in our mind, an embedded information system makes possible the presentation of various information accessed from various sources in real time. This means that the system not only makes use of “local” embedded data (e.g. maps or multimedia data), but also of data available on distant information servers. Dynamic data (e.g. roadwork, traffic or meteorological conditions) have to be transmitted to the vehicle and integrated within static data (maps).

The last constraint concerns the means required for bringing the system into play. We actually plan to have a system easily workable and widely accessible.

So, we consider that the main functionalities of an
embedded information system are (i) the positioning, (ii) the possibility to access external data sources and (iii) the presentation of various types of information. The choice of means that enables us to implement those requirements is based on widely used and easily accessible technologies.

We use GPS device to localize the mobile. Cellular phone is used to establish connection between mobile part of the system and distant data servers. Data are transferred between different part of the system by Internet based connections. Finally, we use Web browser with Java applet to visualize and process information of various types.

Increasing popularity and development of Internet, Web browsers and Java programming language lead in the use of those means in many various domains. Internet now is the main medium of diffusion and access to the information almost everywhere in the world. Web browsers become real presentation environments for all kinds of multimedia data. Java language allows to develop portable applications - applets - that can be easily and safely downloaded from distant servers and then executed by various browsers on various operating systems. It facilitates the development of applications that can be diffused to a very large number of users.

2.2. Problematic

The means chosen are already available and applicable in many domains. They can be easily used in development of our system. But some problems still remain, namely: (i) the low speed of communication between the components of the system, (ii) the diversity of information sources, (iii) the diversity of information types, especially for spatial data, and (iv) the diversity and large number of users. Our studies presented in this paper cover the first two points.

Diversity of data sources is problematic when they are accessed from Java applet. Java security model imposes restrictions on an applet executed by a browser: it can only gather data coming from the same source as its code. In our case, we have at least two data sources. In fact, GPS data are acquired by a specific device connected to the computer and they need to be displayed on a map loaded from a server. Note that there are no similar restrictions to usual Java application.

Moreover, the use of communications based on cellular phones seriously limits the speed of data transfer. Multimedia information tends to take up large amounts of memory. So the usual means of transferring information on Internet (by HTTP or FTP) must be optimized or new means of communication need to be studied.

In the following sections we present the architecture of the embedded system we have been working on. This architecture allows us to solve those problems. But first, we will review the access to distributed geospatial data.

3. Distant access to geographical databases

A considerable amount of work has been carried out on the sharing spatial data between different applications and users. Various formats, interfaces and specifications are developed for this purpose, like Open Geodata Interoperability Specification [4] or Open Geospatial Datastore Interface [5]. Several commercial and noncommercial applications and libraries of software components as Alexandria Digital Library [10] or GeoToolKit [2] have been developed to process distributed spatial data.

Access to spatial information using Web browsers is another aspect of current research in this sector. Two main approaches are being used: server-side and client-side oriented, depending on which side data processing and calculations are done.

Common Gateway Interface (CGI) scripts are typically used when data are processed on the server side. CGI script is a special program executed by a Web server to process data sent by a browser. In the case of spatial data, we usually collect the parameters for a spatial query send them to the server and receive the HTML page that displays the produced map. GRASSLinks, a WWW interface for geographic information systems developed at University of California, Berkley, could be an example of such a processing of geospatial data (www.regis.berkeley.edu/grasslinks/).

This method has some drawbacks. It is not possible to establish a permanent connection between the client and the server for a work session. This complicates the re-use of calculation results. But if a solution to this problem can be found [8], another drawback still remains: all calculations are made on the server, and the client only sees its results. It considerably increases the workload of the server.

Such drawbacks can be avoided in using the client-side oriented methods. Their are based on software components executed by a Web browser or operating system: browser plug-ins, ActiveX components or Java applets. They offer more flexibility than CGI scripts. As modules containing an executable code, they allow local calculations and more advanced exchange of data with a server (permanent connections, progressive data loading). They also offer higher level of interactivity with a user.

Note that the use of Java applets offers the highest portability of components (they can be executed on any Java enabled browser), while plug-in must be developed for concrete browser and operating system. ActiveX components are limited to Microsoft Windows operating systems.

Advantages of client-side oriented methods lead in the development and the use of very large number of such software components. Numerous examples can be found: Autodesk MapGuide plug-in [1], GeoLib ActiveX component [7], Java based Lava [3] prototype, GAEA Java applet [6] or Java applet presented in [9].
4. System architecture

The architecture of the system that we present is a client-server one. The quantity of information (geographical or multimedia) can be very considerable and the dynamic information (traffic, position of vehicles) requires frequent updating. It is thus preferable to have a centralized source of information and to send progressively all the necessary data to the users, according to their needs.

4.1. Integrating geographical data

We have considered several solutions to access different data sources from an applet.

One solution (figure 1) would be to create a specific application (program executed by the operating system), which can access the GPS data and send it to the server. Then, the applet can receive data from the server in the “usual manner”.

Another solution would be to use an electronically signed (“trusted”) applet (this approach to access distributed spatial data is used in [11]). The browser can no longer impose any security restrictions on such an applet. In this way it can access directly various data sources, including local sources. But this solution also has its drawbacks. Access to the GPS is made by operating system’s native code. Experience acquired during the tests showed that the applet code should be as simple as possible, if we aim to run it on different browsers and operating systems without making any prior modifications. Access to the physical resources of a computer would make it very dependent on the concrete operating system.

The solution we have adopted derives from the first one. We use a local mini-server installed on the client’s computer. The principle of this solution is that the applet accesses all the data via this server (figure 2). This server receives requests from the applet, redirects them and sends an answer. For the applet, it is always the same source of data, and thus it does not need to violate the security restrictions.

The main disadvantage of this solution is that the user has to install additional software in order to run the system. It is not sufficient to download and execute an applet. But there are some advantages.

Communication between server side and client side of the system can be carried out by the servers and hidden from its other parts. This allows us to find some methods to optimize connection, at the same time staying with the same mode of communication in regard to the other components. For example, the reply to a request of the browser or an applet can always be in HTTP format. The possibility to optimize communications presents a considerable interest in the context of an embedded system, as this system limits the speed of data transfer.

4.2. Organization of the system

The organization of the system is shown in figure 2. The browser with the applet and the local server are the software components on the client side.

On the server side, HTML pages and applet’s code are data requested by a Web browser (we will call corresponding requests as source requests). Another data are processed by an applet. It can be spatial data (vector or raster images) as well as associated alphanumeric or multimedia information stored in a database or a file system (we will call its requests as data requests).

A HTTP server and a specific system server are the software components on the server side. The system server acts as entry point for all clients. It only processes data requests. The others requests (source requests) are redirected towards the HTTP server. There are several reasons for separating the tasks between the two servers.

By redirecting the source requests to the Web server we ensure their correct processing. A system server can in turn process the data requests more efficiently. It can access directly to a database (the Web server in this case has to use CGI scripts). It can process the data of a map locally and send it in parts according to the needs of the
client (the Web server can only send whole files). The system server increases security of data, because there is no direct access to the databases, nor to the image files.

### 4.3. Functioning

The figure 2 shows also the functioning principle of the system (the arrows show the data flows).

A Web browser connects to the local server and loads an HTML page and the applet’s code. Once loaded, the applet starts execution: sends data requests to the local server, loads and visualizes the required data (configuration files, raster and/or vector images of a map). It can also send a request to obtain GPS data, receive them and visualize the vehicle position on the map. The functioning of the applet can be changed to obtain the positions of vehicles stored in the database.

All requests sent by the applet (and the browser) are formulated as HTTP queries. Their destination is always the same as the origin of the applet, i.e. local server.

Local server analyses a request and decides whatever it has to be redirected to distant server or processed locally. In the case of GPS data request it recovers the received GPS data by a serial port and sends it to the applet. It can also send GPS data to the system server to store them in the database.

The operating principle of the system server is similar to that of the local server, and it depends on the type of request. It can establish a connection to the Web server and send back the answer to a client. The system server can read and send raster- or vector-based data stored within files. It can also make a connection to a database to process a query: update or select data.

Note that local and system server act like a very specialized proxy server, widely used on the web.

### 4.4. Implementation

All parts of the system are implemented as Java applications. Consequently, they are portable applications and can be executed on any kind of operating system with Java Runtime Environment (JRE).

The GPS being connected to the serial port, the local server accesses it using a javax package, extension of the Java Development Toolkit (JDK). The system server establishes a connection to a database using Java Database Connectivity (JDBC) functions. Moreover, the system server is independent of the concrete HTTP server, since it communicates with it only by HTTP queries.

The visible part of the client is a Java applet (figure 3). It allows visualization of maps made up of different layers: vector-based layers and raster images. They constitute static information, that do not change during a work session, and that do not need to be reloaded. A map can also contain dynamic layers that represent information on the position of the vehicle(s). In the case of the visualization of dynamic layers, the applet regularly queries the local server by sending a request of new GPS data.

It also allows basic operations on maps: scale change, zooming and panning.

### 4.5. Tests

Several tests of our system have been performed in real conditions. On the client side (in vehicle) we used a portable computer with connected differential GPS device and GSM cellular phone. Microsoft Windows 95 operating system was installed and Netscape Navigator was used to execute the Java applet. On the server side we used Postgres database to store information about vehicle position. This information was sent from the client to the server as User Datagram Protocol (UDP) packets.

As expected, connection has very limited speed (~1.5Kb/s). We encountered also some problems of stability of connection (possibly due to changes of GSM cells). Thus, it is difficult to practically use such connection to transfer large amounts of multimedia data in real time. Contrarily, it is quite acceptable to send small packets of information. It can be information about vehicle position sent to the server. It can also be some parts or layers of a map, or raster images of reduced resolution.

### 5. Conclusion and future work

A software architecture for an embedded system has been presented. Our developments and tests have pointed out that a system based on such an architecture can be easily implemented. But its practical use requires the resolution of some problems, mainly caused by the low speed of data exchange between clients and servers.

The optimization of communications between servers and clients is the first objective of our future work. We anticipate two main directions:
• The use of a data cache - a local copy of the data that have been downloaded by the applet. The synchronisation of updated data must be implemented, as it was done in [8].
• The definition of new modes or protocols of communication between servers, in order to reduce the amount of information to be transmitted. Such an optimization can be based on data compression or on its serialization. In the first case, we reduce the size of information, which has to be transmitted. In the second case, we reduce information itself.

We can also point out that the proposed solutions can be used not only in the context of an embedded system. This architecture can be adopted, with or without any modification, for “immobile” applications, and especially for those, which use Internet to exchange the information. In this case several configurations are possible:
• The removal of the local server. This modification is possible, if we do not need to access local information sources, and if the communication speed is sufficient.
• The removal of the system server. This situation is similar to the previous one, as the local server can take care of some of the functionalities of the system server (for example the connection to distant databases).
• The removal of both servers. This case can be most acceptable for a simple user, as the client (the applet) directly accesses to data through a Web server.
• The introduction of other data sources. Data can be preloaded or stored on a CD and must be integrated with data loaded from a server.

Acknowledgements

Presented research has been partly funded by the region Poitou-Charentes and the city of La Rochelle, France.

References