Mobile Information Systems 2 (2006) 111–133
IOS Press

A generic framework for Location-Based Services (LBS) provisioning

Ioannis Priggouris, Dimitrios Spanoudakis, Manos Spanoudakis and Stathes Hadjiefthymiades
Department of Informatics and Telecommunications, National Kapodistrian University of Athens, Ilisia, Athens 15784, Greece
Tel.: +302107275327; Fax: +302107275601;
E-mail: {iprigg,d.spanoydakhs,mspan,shadj}@di.uoa.gr

Abstract. Location Based Services can be considered as one of the most rapidly expanding fields of the mobile communications sector, with an impressively large application range. The proliferation of mobile/wireless Internet and mobile computing, and the constantly increasing use of handheld, mobile devices and position tracking technologies prepared the grounds for the introduction of this new type of services. The combination of position fixing mechanisms with location-dependent, geographical information, can offer truly customized personal communication services through the mobile phone or other type of devices. Prompted by the avalanche of technology advances in the aforementioned areas in this paper we present an integrated platform for delivering Location Based Services (LBS). The platform covers the full life cycle of a LBS starting from the specification of the service, covering issues like the deployment and maintenance of services, the service invocation and the final delivery of the produced results to the invoking user. A prototype implementation of the discussed platform was developed and used to perform a series of trial services, with the purpose of demonstrating the pursued functionality.

Keywords: Location Based Services (LBS), positioning, GIS, middleware, software architecture

1. Introduction

The mobile communications market experiences an unprecedented boom in the recent years. New handheld devices with increased capabilities are introduced, while mobile operators are striving to gain a significant portion of the market by delivering new state-of-the-art value added network services that can fully utilize the given technology. Location-Based Services (LBS) is just one such category of services, upon which both manufacturers and mobile operators have invested a lot. However, delivering new services requires developing means and tools that assist in their creation, provision and maintenance.

This paper presents an integrated middleware and service provision platform. The main focus of the platform lies on the LBS domain. It covers the full life cycle of LBS. It provides means that facilitate the creation and provision of such services with minimum effort from all involved parties (e.g., service provider, mobile operator). The platform uses open standards so that it can collaborate with existing network and transport technologies, accommodate future evolved technologies and be accommodated in current and future telecommunication infrastructures.

The rest of the article is structured as follows. An overview of the involved technologies is presented in the initial section. The next sections delve deep into the details of the platform's architecture, discussing its software architecture and the provided functionality. The technical discussion covers in detail the core system and more roughly the Service Creation Environment, which accompanies the platform and

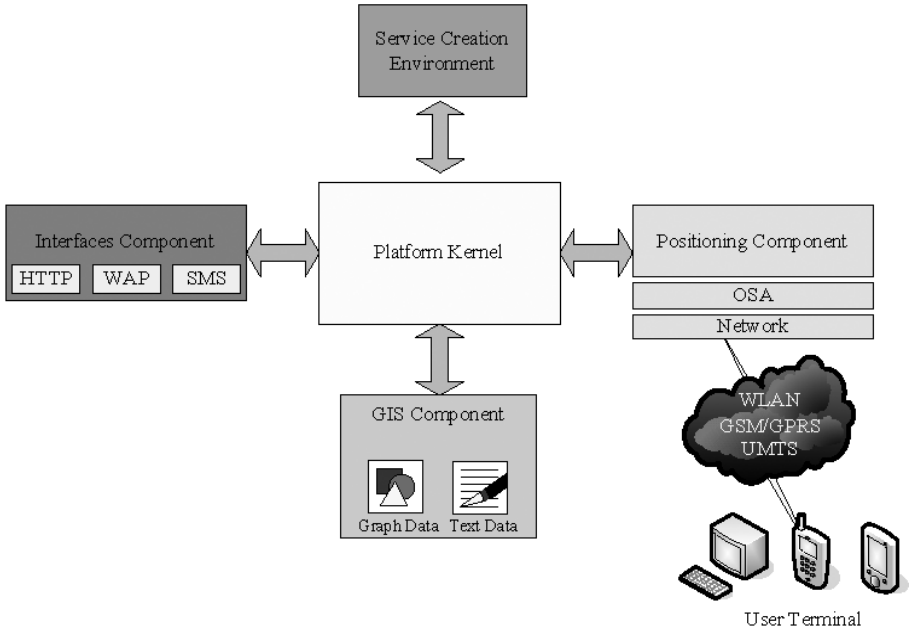

Fig. 1. Middleware and Service Provisioning Platform architecture.

facilitates the service creation and installation process. The article continues with a brief comparison of the pursued system with other platforms available today in the LBS provisioning domain. The article ends with the description of the trials and their results, followed by some conclusions on the discussed work. and some additional thoughts on how to improve the platform in terms of functionality and performance.

2. Platform overview and related technologies

The discussed platform builds upon technical advances in the area of Info-mobility, distributed/mobile computing, and Geographical Information Systems (GIS) and aims to provide all the functionality needed for delivering location-based services in a single platform. The platform focuses on the development, installation and execution of services and consolidates several technologies.

A modular approach has been followed during the design of the platform resulting to an architecture, which defines a core component, the kernel, and independent functional entities on the boundary, each covering a certain operational area in the LBS provisioning chain. These functional entities comprise the integrated middleware platform shown in Fig. 1. The autonomy granted to each component through this architectural approach makes possible the extension of the platform through the addition of new components, modification or even replacement of the existing ones, in order to enhance the overall functionality.

In order to better illustrate the philosophy behind the proposed framework we will provide, in the following section, an overview of the involved LBS technologies.

2.1. Positioning technology

Positioning technology is a key point in the LBS context. Research in this area during the last years is impressive. A plethora of solutions [6], which provide accurate estimations of user's location have emerged, each with distinct characteristics and capabilities applicable in different circumstances. All solutions, however, can be clustered in two main categories:

- Satellite-Based systems, such as the Global Positioning System (GPS) and Galileo [8]. Such systems rely on a set of satellites, which continuously transmit position-related information. Specific terminal devices can be used for receiving the transmitted signals, performing the necessary calculations and produce estimations of the terminal's position. A well-known advantage of satellite-based systems is the high accuracy they can achieve. On the other hand they have certain drawbacks, which include their inability to operate in indoor environments and the need for specialized, usually costly, terminal equipment.
- Terrestrial Infrastructure-Based systems are, generally, less accurate than their satellite counterparts. However, despite this drawback they are widely used, as they do not require expensive add-ons to terminal devices. Terrestrial infrastructure-based positioning reuses the network infrastructure available in 2G and 3G networks, by enhancing them with location-aware hardware and software, capable of performing the complex mathematical calculations needed for achieving accurate location estimation. GSM positioning has been standardized and includes a variety of methods (Cell identifier, Time of Arrival, etc.), each providing a different level of accuracy that can fit almost every need. In addition, significant progress has been achieved in the area of WLAN based location technology. The problem of positioning in these environments has been solved with ad-hoc solutions due to the actual lack of standardization for retrieving location information from WLAN devices.

2.2. GIS technology

The Geographic Information System (GIS) technology, although not strictly required in order to provide location-dependent services, comprises a key technology in this area as its presence can greatly enhance the quality of the delivered service. Incorporating detailed visual representations (e.g., maps) or analytical information about certain points of interest can greatly enhance the market potential of the delivered service.

The GIS consists of a comprehensive database populated with location-specific information, along with an integrated set of tools for querying, analyzing, and displaying this information. A standard database management system (e.g., Oracle, Access) is usually used for storing the spatial information. Numerous data formats have been defined for storing GIS data in these databases (e.g., geotiffs, shapefiles). Apparently, each data format needs specific software in order to be processed, resulting to a GIS market flooded with proprietary GIS software. However, products that perform translation between the most popular formats do exist today; thereupon binding to a specific format does not impose serious problems. A coarse categorization of the requests that can be possibly dispatched to a GIS system includes:

- Navigation requests for acquiring information on how to travel between two distinct locations.
- Proximity requests, where the query aims to retrieve information related to the user's current position (e.g., points of interest).
- Find location requests, where information about a specific location is requested (e.g., a map of the area)
- Geocoding requests, where based on information about the current location (e.g., address, street name), the corresponding coordinates are requested.

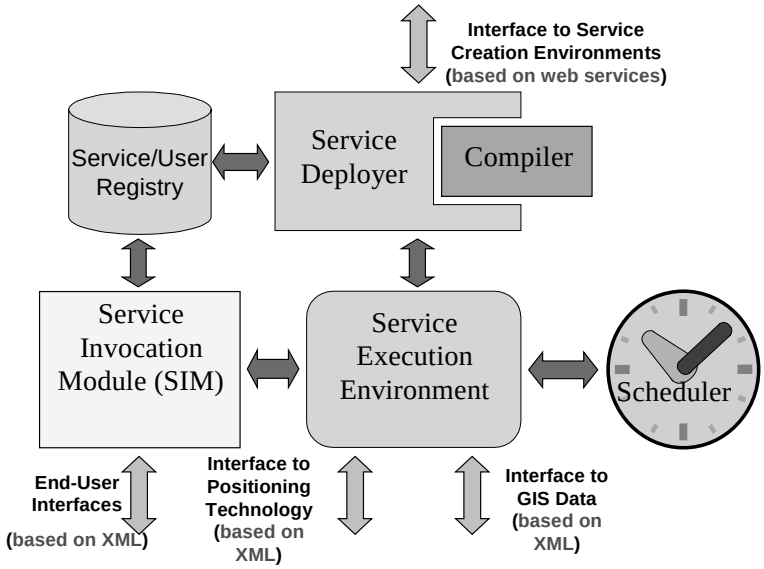

Fig. 2. Kernel Architecture.

2.3. Interfacing technologies

The term "interfacing technologies" refers to the transport technologies used for communicating and transferring data between two end-points. As LBS are strongly coupled with the mobile communications domain bearers that are used in the latter are of prime importance for transferring data between the location services platform and its corresponding clients. Until recently, the exchange of data in mobile networks was limited to the SMS and WAP technologies. The former offered a basic transport protocol for textual data while the latter a more advanced protocol, similar but more restrictive, to HTTP. SMS is widely used today but WAP failed to gain wide acceptance mainly due to cost-efficiency reasons, resulted from the connection-oriented nature of the GSM technology.

Evolution towards 3G networks provided significant benefits to both technologies. The Enhanced Messaging Service (EMS) and Multimedia Messaging Service (MMS) technologies augmented the SMS protocol with advanced multimedia capabilities while the GPRS technology not only revived the WAP technology, but together with the evolution of handheld devices made possible the use of HTTP in the mobile domain. In WLAN environments the HTTP remains the simplest and most promising technology for interfacing with servers and other remote systems.

3. Platform architecture

After this short introduction on enabling technologies we proceed with the analysis of the architecture of the LBS provisioning and middleware platform. As shown in Fig. 2, four main entities have been defined in the proposed architecture. These entities are:

- The Kernel,
- The Interfaces Component (IFS),
- The Positioning Component (POS), and,

– The GIS Component (GIS).

The platform is complemented with a Service Creation Environment (SCE), which is not required for the operation of the rest of the system. However, its presence can greatly enhance the service provisioning process. An extensive presentation of the SCE will take place in Section 4.

3.1. The kernel

The role of the Kernel in the proposed architecture is to serve as the runtime environment for the offered services as well as to coordinate the operation of the peripheral components according to the service logic. Communication of the kernel with the peripheral components is based on XML protocols, thus, leaving the corresponding interfaces open and adaptable to future updates or replacements of the respective peripheral component. Moreover, Web Service (WS) interfaces that allow easy and secure invocation from everywhere have been built for the communication with external entities (e.g., the Service Creation Environment). The overall functionality of the kernel is summarized in the following basic tasks:

– Service Installation & Management
– Service Provision

The internal architecture of the Kernel is depicted in Fig. 2, and as shown it consists of several smaller entities, which cooperate in order to perform the aforementioned tasks. It is evident that the activities mentioned above, cover the full life cycle of a service from its initial installation on the system to its final removal.

As seen in Fig. 2, many of the involved modules act as mediators for interfacing with the technologies presented in Section 2.

3.1.1. Service installation & management

We will start our tour of the kernel from the Service Deployer module, which is the entity responsible for receiving requests for the installation of new services. Such requests may originate from Service Creators that operate the SCE, which accompanies the platform. In this perspective, the Service Deployer can be considered as the server side part for the SCE. The featured WS interface provides to potential clients (e.g., supported SCEs), methods for receiving new services, for querying the status of already installed services, for acquiring information about specific services that have been previously installed on the platform, etc. Clients that need to install new services in the platform should produce the service specification document and then use the methods provided by the discussed interface to upload that document to the kernel. The service specification document should comply with the Service Control Language (SCL) syntax and grammar. The SCL is the language understood by the Service Deployer and comprises a convenient tool for quickly developing LBS. More details about the SCL and its features are provided with the description of the SCE, in Section 4.

Integrated with the Service Deployer is the **Compiler**, which translates the service specification from SCL to Java code and generates the executable component of the service in the form of an Enterprise Java Bean (EJB). The EJB format can be directly loaded and executed inside the **Service Execution Environment** (**SEE**)**.** The SEE reproduces the functionality of a standard java-enabled application server, which has been augmented with functionality that enables the bi-directional communication with the POS and the GIS peripheral components. This communication is handled by specific modules residing inside the SEE, which are called **Component Handlers**. Each Component Handler translates the service

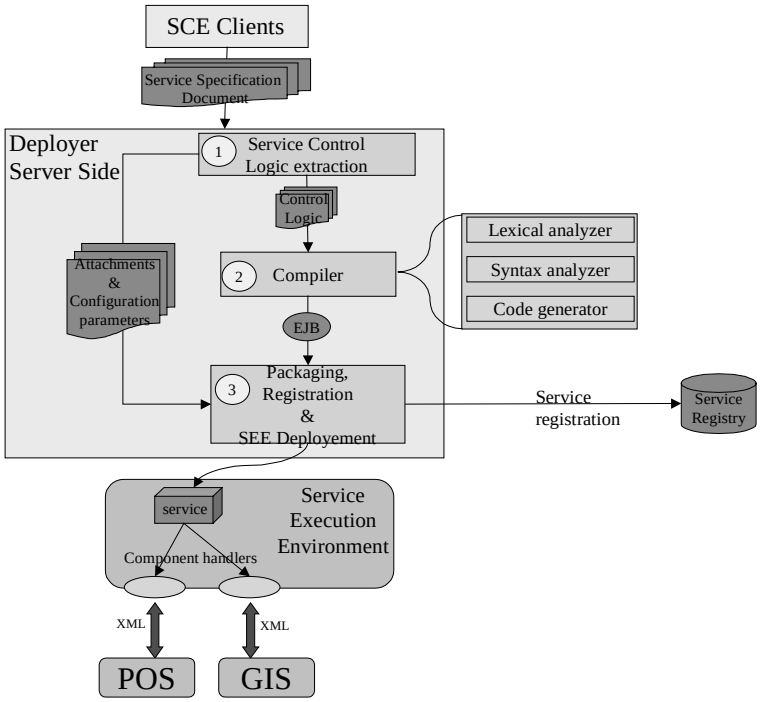

Fig. 3. Service Installation process.

control logic directives to synchronous method calls on the peripheral component. Data exchanged between the handlers and the corresponding peripheral component is XML-encoded. Using XML in this interface enables the kernel to adapt easily to potential enhancements of the involved technologies. Parsing of the communication parameters is left to the service, rather than being hard-coded in the kernel (e.g., inside the Component Handlers). For instance, imagine that a new positioning technology is devised. The new technology may engross new parameters and provide results formatted in a completely new way. Integrating the new technology to the platform will require minimum effort and will necessitate a simple modification to the XML schema that is used for the messaging between the SEE and the POS component. The toll we have to pay for this openness is a small delay in the communication (messaging protocols cannot compare to direct invocation calls), as well as the transfer of some complexity to the service creator, which needs to parse XML messages in the service control logic.

The SEE also incorporates concept-mechanisms such as repositories for configuration variables, repositories for variables shared between different instances of the same service, for variables that persist between successive invocations of the same service for the same user. These repositories are created and populated during the service installation process. The overall installation process for new services is depicted in Fig. 3.

Each service, following its successful deployment in the kernel is registered with the Service Registry. The Service Registry is part of the Kernel's **Registry** (see Fig. 2). The Registry consists of two parts: the Service Registry and the User Registry. The Service Registry is used for storing data related to the service. The data is stored there upon the installation of the service and is used during the initialization of the platform (i.e., bootstrapping), for reloading the service contexts, but also during the invocation of the service in order to locate the appropriate executable component that needs to be invoked. The User Registry contains the accounts of the users that have registered with the platform. Its presence in

the system, allows for the adoption of different business models in the service provision process (e.g., pre-paid model, per-invocation charging, etc.). The Registry is additionally used for managing services after their initial installation on the platform (e.g., configuring service parameters, changing invocation rights, uninstalling services).

3.1.2. Service provision

When a service has been installed in the kernel it becomes available to the users. In this section we will discuss the process of service provision from the kernel's perspective. In subsequent sections we will explain how the peripheral components cooperate in this process. We have already discussed that when service installation completes, the executable form of the service resides inside the SEE. Now, in order for the service to "run", a certain trigger is required. Such triggers arrive either from an end user and are received through the Interfaces component or they are generated internally by the Scheduler. In both cases the Service Invocation Module (SIM) is the final interceptor for all triggers.

The role of the Scheduler in the platform is to support the automatic execution of services. It has built-in support for both event- and time-triggered service execution and a variety of scheduling paradigms are catered for (one time scheduling, infinite periodic scheduling, periodic scheduling with fixed deadlines, etc.). Time-triggered scheduling is performed through the Scheduler's management interface, and requires defining the service name and the execution timeframe. The Scheduler also provides means for delivering location events coming from the Positioning component to the service (e.g., the user entered a mall or a theater), with the minimum possible delay. This is part of the functionality provided by the OSA/Parlay mobility specification [2] and could prove particularly useful for implementing Location Based advertising. No priorities have been integrated in its architecture, which means that all events are treated equally. However, this approach does not impose any restrictions, as the scheduler is a scalable component, which guarantees, that even concurrent events are executed with the minimum possible delay. Particular effort, during the design of the Scheduler, was spent on achieving persistency of the scheduled tasks. Hence, aiming to maximum reliability, the Scheduler features mechanisms for storing information pertaining to each scheduled task in order to cope with potentials crashes of the platform. A detailed discussion on the Scheduler's architecture and capabilities can be found in [10].

The Service Invocation Module (SIM) receives user requests coming from the Interfaces component. The SIM receives requests in the form of XML messages, which means that any client that wants to invoke a service directly from the SIM, will need to formulate such a message. Corresponding answers are also dispatched in the same form. The XML messaging protocol (Fig. 4) was designed by taking into account the efficiency requirements of real-time communication. In this perspective, it is fairly simple and compact, but at the same time flexible enough in order to accommodate all information needed for executing different types of services and handling the produced responses. The general processing rules that the SIM applies on each incoming message are as follows:

1. The incoming request (in the form of an XML document) is received.
2. The document is validated against the XML DTD.
3. If the validation succeeds, the document elements are processed in order to deduce the service to be invoked and its parameters.
4. The service registry is consulted, in order to retrieve the context of the requested service.
5. The service is invoked.
6. The service results are packaged in a new XML document.
7. The results, encapsulated into the XML document, are returned to the client that performed the invocation.

```
<!ELEMENT      interf-krn-message  (request-to-kernel | response-from-kernel)  >
<!ELEMENT      request-to-kernel   (user-data, service-invoc-data, security-data?)  >
<!ATTLIST      request-to-kernel
               msg-id    CDATA     #REQUIRED
               date-time CDATA     #REQUIRED>
<!ELEMENT      user-data (term-cap?)>
<!ATTLIST      user-data
               user-id             CDATA               #REQUIRED
               user-address                  CDATA               #REQUIRED
               second-response-type          (sms | wap | http)  #IMPLIED>
<!ELEMENT      term-cap  EMPTY>
<!ELEMENT      service-invoc-data  (service-data, cookies*)>
<!ATTLIST      service-invoc-data
               service-id          CDATA               #REQUIRED
               service-type        (sms | wap | http)  #REQUIRED
               rqst-method         CDATA               #REQUIRED>
<!ELEMENT      service-data        CDATA>
<!ELEMENT      cookies   EMPTY>
<!ATTLIST      cookies
               session-id          CDATA     #IMPLIED
               name                CDATA     #REQUIRED
               value               CDATA     #REQUIRED
               description         CDATA     #REQUIRED>
<!ELEMENT      security-data       EMPTY  >
<!ATTLIST      security-data
               url-proxy CDATA     #IMPLIED
               url-referer         CDATA     #IMPLIED>
<!ELEMENT      response-from-kernel          (xslt-data, service-invoc-data, user-data)
<!ATTLIST      response-from-kernel
               msg-id              CDATA               #REQUIRED
               cservlet-name       CDATA               #IMPLIED
               date-time           CDATA               #REQUIRED
               priority            (high | medium | low)          medium >
<!ELEMENT      xslt-data EMPTY  >
<!ATTLIST      xslt-data
               type      (generic | specific)           generic
               name      CDATA                          #IMPLIED>
```

Fig. 4. SIM-Interfaces communication protocol (DTD).

3.2. Peripheral components

The peripheral components are named after their relative position around the Kernel. Each component features a well-defined API for interfacing with the Kernel, which is based on XML. This modular architecture, along with the use of XML results in autonomous and independent components, which can be swapped-out and replaced, at any moment, by other entities implementing the same APIs without obstructing the Kernel's operation. As shown in Fig. 1, three peripheral components have been incorporated, i.e. the Positioning component, the GIS component and the Interfaces component.

3.2.1. The positioning component

The positioning component is responsible for providing location information to the Kernel. It provides the means for interacting with the various location systems that are available today. The interface towards the Kernel is based on XML while the interface towards Location systems adopts the OSA/Parlay specification implemented over WS [2].

The main role of this component is to integrate all underlying positioning technologies, abstracting from their differences, under a unified interface, which is used by the service. The interface is fully configurable through a variety of parameters intended to cover all possible cases and operation paradigms.

The current implementation of the Positioning component supports location retrieval from 2G/3G networks and WLANs [5]. GPS support also exists through the use of a specific software client module running on the terminal side. This client pushes the GPS location data (piggybacked on the request) towards the corresponding server module residing inside the positioning component (GPS Wrapper). The framework's support for various positioning technologies allows the seamless provision of LBS across different infrastructures and facilitates user roaming.

The positioning component consists of several modules that carry out the common functionality needed for position retrieval. Functions, like multiplexing, routing or scheduling are mandatory steps that every positioning request has to go through, subject to its type. Functionality specific to the positioning technology has been placed in separate modules, which are called wrappers, as their task is to wrap the peculiarities of the underlying positioning system. The positioning component supports four types of positioning requests, namely those listed below:

1. Request – Response (RR). This is the most common positioning request, where a service requests the user's position providing his identity as input.
2. Periodic Request (PR), which is, actually, a periodically activated RR request registered to the Kernel's scheduler.
3. Event-driven Request (ER) where an initial request which defines a boundary area is registered to the underlying positioning mechanism. After the registration completes, the positioning mechanism returns events on the user's location in relation to the defined area (e.g., alerts every time the user is entering or leaving the specified area).
4. Generic Request (GR). This is an RR kind of request where all underlying networks are searched for locating a specific user.

The internal architecture of the Positioning component includes the Dispatcher, the Router, the QoS Scheduler, the Multiplexer and the Wrappers (see Fig. 5). Location requests from the Kernel are routed by the Dispatcher to the appropriate sub-module for further processing. If the request is of type ER it is processed by the Multiplexer. The latter multiplexes requests coming from the same user terminal in order to end up with a single request registration within a certain network for each user. Such multiplexing reduces the load at the network, as it ensures that only one request per user remains pending, inside the network, at any given time. The three other types of requests are handled by the Scheduler. The Router performs address resolution for the addresses (e.g., IP or E.164 addresses) of the traced terminals. Resolved information is used for routing requests to the appropriate wrapper (i.e., the wrapper handling the communication with the network that the mobile device is connected to). The router entries are simple mappings of address prefixes to wrapper IDs; they are static and are updated manually by the platform administrator.

The wrapper is the actual client of the underlying positioning mechanism. Each wrapper instance has a unique identifier and is bound to a specific positioning gateway. It embodies a set of message queues, for incoming and outgoing traffic and communicates with the underlying network using an OSA-compliant WS interface. We should note that the wrapper's architecture depends exclusively on the underlying positioning technology. Currently, three different wrappers have been integrated to the platform for covering GSM, WLAN and GPS positioning. Supporting additional positioning mechanisms would simply require developing new wrappers and integrating them to the middleware. Further discussion on the positioning component can be found in [4].

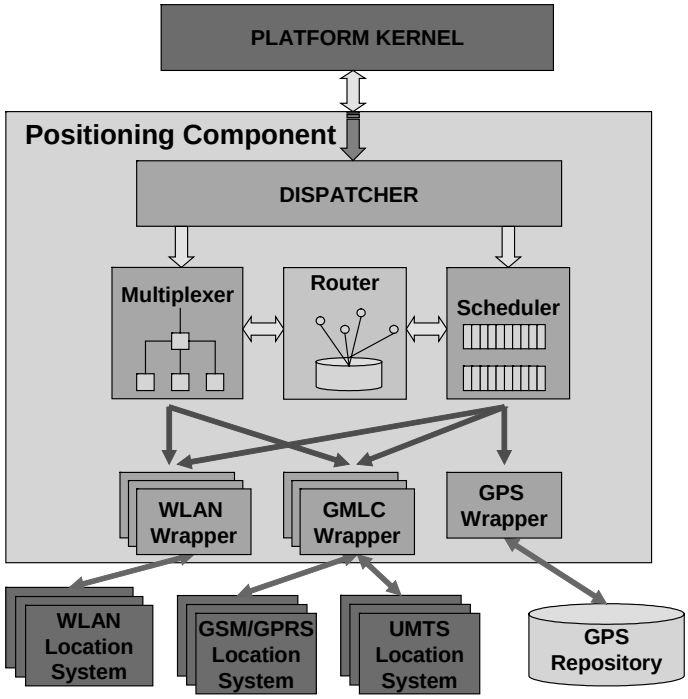

Fig. 5. Positioning Component architecture.

3.2.2. The GIS Component

The GIS Component is the mediator of the platform with the GIS repository. The component covers the process of spatial information retrieval and provides means for creating visual representations (e.g., maps) of the spatial data. The component implements algorithms like navigation routing and geo-coding, which render it capable of answering both simple and advanced requests. It consists of two modules: a client module, which interfaces with the Kernel and a server module, which interfaces with the GIS repository. Communication between the two parts is achieved through a WS-enabled interface. Such an approach allows the two involved parties to be technology independent. This is very important, since, in most cases, GIS repositories are operated by external entities (e.g., content providers) and may adopt a different implementation technology. Currently, the GIS component supports the ESRI shapefiles data format and implements the following services:

- Indoor and Outdoor localization service.
- Indoor and Outdoor navigation service.
- Outdoor proximity service for finding POIs.
- Outdoor geocoding and reverse geocoding service for finding street details based on provided coordinates or the coordinates of a given place accordingly.

Supporting one specific data format is not restrictive as there are several GIS data conversion/transformation tools available today. Such tools eliminate the need for creating different implementations of the same GIS algorithms as the spatial data format can be changed instead.

3.2.3. The interfaces component

The Interfaces component is the gateway of the platform to the external world. Its current implementation supports three established and commonly used protocols (protocol suites), namely SMS, WAP

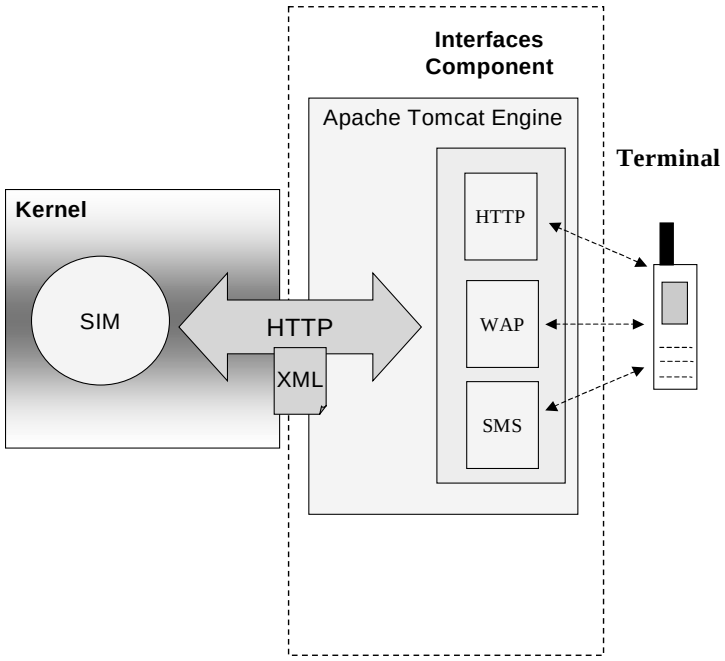

Fig. 6. Interfaces Component Design.

and HTTP. Its design, however, is modular enough so that support for additional protocols can be easily added in the future. The Interfaces Component can be connected with various proxy servers (e.g., HTTP Servers, WAP Gateways or SMS Centers) in order to receive the end users' requests.

The internal design of the component is depicted in Fig. 6 and shows the existence of the special front-end modules, dedicated to the various supported protocols. The main task of these modules is to receive the requests arriving in the respected protocols, process them and form the XML document that is needed for the actual service invocation. The document is next passed to the SIM module, which processes it further according to the rules discussed in earlier paragraphs.

4. The service creation environment

The Service Creation Environment (SCE) comprises an Integrated Development Environment (IDE) supported by a GUI, whose goal is to facilitate the creation and installation of new services on the platform. The SCE features a graphical interface, and customized support for the LBS-oriented language (SCL), which is supported by the platform. Development using the SCE can provide significant assistance to inexperienced users, enabling them to create, deploy and debug services easily. For more adept users a set of command-line tools is provided.

4.1. The development environment

The SCE is based on the Eclipse software platform (http://www.eclipse.org), an open source, extensible, graphical framework, which enables the development of customized GUIs. The SCE supports two development environments. New services can be developed using the standard text-based editor but also

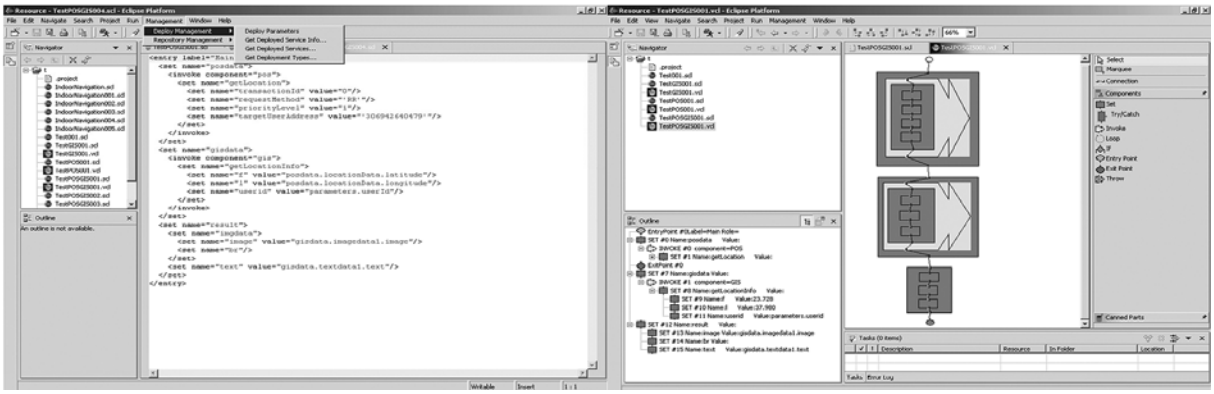

Fig. 7. Textual (left) and Visual (right) editors in action.

a visual environment has been provided. Within the latter it is possible to add and remove commands corresponding to SCL directives, using a simple drag-and-drop mechanism, and to define the flow of execution between the various commands. Commands in the visual editor are represented as boxes. Each box has a set of attributes that need to be filled, resulting to a one-to-one correspondence with the SCL directives. Development in visual mode generates, behind the scene, the complete SCL specification for the new service. Switching between the two environments during development time is possible. Figure 7 displays the same service as it looks in each editing environment.

It is evident that the SCE is an add-on to the platform, and its absence does not restrict its capabilities. Moreover, the web-services interface provided by the Deployer module allows potential users to use their own, possibly different SCEs for the service creation process.

4.2. The Service Control Language (SCL)

The service logic for applications designed to run by the platform is specified through a new language the, SCL (Service Control Language) developed specifically for the LBS domain. The language is based on the XML syntax and resembles other similar scripting languages that have been developed and used for similar purposes (i.e., service specification).

Reviewing other similar languages we can find CPL [9] and SCML [7] as the most notable examples. Both are used in providing next generation telephony services. The use of XML for expressing procedural languages has both benefits and drawbacks. Element nesting used in procedural languages is directly supported by XML. However, serial interpretation is not a core feature of XML and implicit rules must be established, such as ordering of tags and content. Fortunately, some XML processing tools (e.g., DOM, SAX), can provide the XML elements in the order of their appearance in the document. Another feature of the XML representation is that it can easily be presented in a visual, instead of the traditional textual form. This allows the easy development of visual service building tools, like the visual environment that has been developed for supporting the service creation process and which is displayed in Fig. 7. Special syntactical constructs were used to access middleware components (e.g., Positioning, GIS), while the generality of the language allows the creation of complex services.

4.2.1. Language specification

A complete service specification includes the following elements:

1. Service Control Language part: The Service Control Language (SCL part) constitutes the business logic description of the Service Specification Language (SSL). It does not include input processing and filtering, nor support for specific output presentation elements. The language is also designed to be independent from specific interface characteristics, so that potentially diverse functionality is accessible in a unified way.
2. Configuration Options: This section includes configuration settings for the service, such as pre-requisites (e.g. dependency on other services or installed libraries) and parameters (e.g. connection strings to remote databases). Several other elements also reside in this area, such as service descriptions, visualization data (used by the Visual Editor) and potential security constraints.
3. Additional Elements: This area allows the inclusion of files, which may be used directly or indirectly by the service. Such files may provide additional functionality to the service logic (e.g., class libraries, customized HTML/WAP front-end, applets/midlets that can be downloaded to the user terminal, images, etc.).

The SCL constitutes the most complex and interesting part of service specification. During system design it was decided that the language should provide sufficient facilities for simple service description, without the need for external, native components. By examining a number of possible services, the following functionality levels were identified:

1. Very simple services, which do not need any control logic other than the activation of system components in a predetermined order (e.g. a service retrieving the requestor's location)
2. Low processing requirement services, which require some data manipulation, coordination and making simple decisions (e.g. a service retrieving someone else's location)
3. Medium processing services, which require support for manipulating data quantities of varying sizes (e.g. a service for finding nearby restaurants)
4. High processing services, which require arbitrary control capabilities (e.g. a service performing fleet management)

Services belonging to levels (1)–(2) may be easily expressed as a series of simple instructions. For service level (3), in many cases, though not always, processing may be provided by pre-existing platform components, so as to require the same level of language support as (1) and (2). Service level (4) requires the presence of a general-purpose language and the support normally associated with full-featured languages, such as C/C++ and Java, and is not really fit for direct implementation in a scripting language. To ensure sufficient language flexibility, it was determined that at least level (3) services should be supported, and the final decision was to provide a general-purpose scripting language, supporting all functionality levels.

As a mediator for the platform components, SCL needs to communicate and exchange information with peripheral components. In order to support service aggregation, services should also be capable of communicating between them. Services, which support advanced functionality, may also require storage of data across multiple transactions. It is therefore important that SCL provides such facilities to services, which require them. Relational databases, LDAP enabled directory servers or other mechanisms could provide the data persistence facilities. However, exposing such mechanisms to the SCL should not increase the syntactical load of the language. Complex service logic that is not easily expressible using the SCL should be delegated to native platform modules, which are coded in a native language and can, therefore, execute more efficiently. Such logic should be embeddable/invocable from SCL descriptions so that the service execution model remains unchanged, whether the service is implemented using SCL or implemented as built-in platform functionality.

```
<service date="adate" name="GetMyLocation" lang="SCL">
  <entry label="Main" roles="super">

    <set name="posdata">
        <invoke component="POS">
            <set name="getLocation">
                <set name="transactionId" value="0"/>
                <set name="requestMethod" value="'RR'"/>
                <set name="priorityLevel" value="1"/>
                <set name="targetUserAddress" value="'0049932910729'"/>
            </set>
        </invoke>
    </set>
    <set name="gisdata">
        <invoke component="GIS">
            <set name="getLocationInfo">
                <set name="f" value="posdata.longtitude"/>
                <set name="l" value="posdata.latitude"/>
                <set name="userid" value="'10302020210'"/>
            </set>
        </invoke>
    </set>
    <set name="result">
        <set name="image" value="gisdata.imagedata1.image"/>
        <set name="text" value="gisdata.textdata1.text"/>
    </set>

  </entry>
</service>
```

Fig. 8. An SCL example.

The potential of the SCL to be used as a language for expressing arbitrarily complex control logic, affirms that SCL should provide support for general-purpose language constructs. Typical constructs to achieve this are iterations, conditional execution and execution flow control in general. These constructs are meaningful only if execution state can be monitored, such as through the use of variables, as well as the ability to make calculations based upon the state, which implies that SCL needs a sufficiently strong expression mechanism. Finally, SCL should provide a compact description of a service. This directly relates to readability and ease-of-use. Services that require little or no data processing should be simpler and more intuitively expressed by SCL than equivalent descriptions in general-purpose high-level languages.

4.2.2. SCL example

We provide a brief overview of the SCL's grammar and syntax through the simple example service that is listed in Fig. 8.

The service specification comes in the form of an XML document. Only characters from the standard 7-bit ASCII set can be incorporated (multilingual support through UTF-8 encoding). Binary data (classes/midlets, etc.) are described using Base64 encoding. The service presented is a simple "GetMyLocation" service. It defines a single "Main" method, with certain security constraints. Specifically, the method is accessible only by registered users with the specified role ("super") assigned. The "posdata"

variable is defined using the "set" directive. The variable takes the value returned by a method invocation on the Positioning component. Component invocations in the SCL are defined with the "invoke" command, followed by a sequence of variables. The first variable in this sequence defines the service/method to be invoked on the component, whereas the rest define the parameters to be passed to it. In our example, the result of the invocation is returned in the variable "posdata" in the expected format. Note that the SCL, similarly to most scripting languages, does not provide strong type checking and variables are cast to appropriate types dynamically at run-time.

The next invocation is towards the GIS component, where the "getLocationInfo" service is requested. The contents of the "posdata" variable (i.e., the location coordinates), which contains the result of the previous invocation, are used to define the parameters of the new invocation. The result is assigned to the variable "gisdata" and is used for defining a new variable with the special name "result". The "result" variable is reserved for specifying the return value of a method. Hence, the content of the "result" variable will be returned upon invoking the "Main" method of the "GetMyLocation" service. More detailed analysis of SCL can be found in [1].

5. Comparison with other platforms

Location-based solutions comprise an area of major interest and activity in the wireless domain. Many corporate vendors have developed software tools and middleware platforms for handling the delivery of such services. A list of most location-based platforms, available today, is presented in Table 1.

Many of these LBS platforms are general-purpose allowing development of proprietary services, while others are targeted to specific application domains (e.g., fleet management). Almost all of the reviewed platforms are based on JAVA/J2EE.

Table 1 lists some key characteristics for each platform. What can be observed from this list is that although many solutions have capabilities similar to that of the proposed framework none of them integrates the full functionality offered by the proposed platform. For example, there are many platforms, which support 2G/3G networks or WLAN environments but not a single one with support for both of them. Also, none of the general-purpose platforms has built-in GPS support.

Summing up this comparison, it seems that despite the variety of platforms that exist today there is no integrated solution that covers all aspects concerning LBS provision, i.e. from new service specification to its actual deployment and delivery to end users. Most of the available platforms focus on the process of deploying and delivering the service and neglect issues such as creation and specification of the service logic. Others do consider those latter issues but restrict their capabilities on a limited subset of the full LBS spectrum.

6. Performance evaluation

In order to perform the evaluation of the proposed architecture, a prototypical implementation was developed. The prototype system was based on Java technologies and more specifically on the J2EE framework (http://java.sun.com/j2ee) and specifically the EJB 2.0 specification. The kernel and the peripheral components were developed using the JBoss application server (http://www.jboss.org) as basis. For position retrieval, an emulation of an OSA-compliant positioning gateway was developed using the Apache Tomcat 5.5 jsp/servlet engine (http://www.apache.org) and the Web Services Axis toolkit. Finally, a simple GIS server was built using the Java map objects software development kit.

Table 1
Location Based Service platforms

LBS Platform	Provided by	Key characteristics	Information URL
ArcLocation Solutions	ESRI	WAP/SMS/HTTP connectivity, GMLC connectivity using MLP	http://www.esri.com/industries/locationservices/business/developer.html
Autodesk Location Services	AutoDesk	Service deployment through java or web services APIs	**http://locationservices**.**autodesk**.com
Canvas Location-Enabling Server	Telenity	SCE provided, Services specified in SCML, OSA/Parlay support	http://www.telenity.com
Cellocate	Cell-Loc Inc.	Uses proprietary positioning hardware for delivering LBS	http://www.cell-loc.com/
Celltick Platform	Celltick Technologies	Supports GSM and GPRS networks using SMS or WAP	http://www.**celltick**.com
Location Engine	Kivera Inc.	No interface to positioning infrastructure	http://www.**kivera**.com
LocationAgent	Mapflow	Service deployment over 2G/3G networks	http://www.**mapflow**.com
MapInfo MapXtreme Java Edition	MapInfo	Java middleware for LBS but without positioning interface	http://www.**mapinfo**.com
Mobile Positioning System	Ericsson	LBS for 2G/3G networks	http://www.ericsson.com/mobilityworld/sub/open/technologies/mobile positioning/about/mps system overview
SpatialFX	ObjectFX Corporation	Java enabled software for performing spatial queries	http://www.objectfx.com
Xypoint Location Platform (XLP)	TeleCommunication Systems, TCS	GSM, CDMA, TDMA and 3G support	http://www.telecomsys.com
Webraska Products	Webraska Mobile Technologies	GMLC positioning interface, SOAP HTTP/XML APIs for service development	http://www.**webraska**.com
The Cellpoint MLS/MLB architecture	cellpoint	2G/3G networks support	http://www.**cellpoint**.com

6.1. Trial setup

The trial setup involved the network topology displayed in Fig. 9.

The LBS middleware runs on an AMD Athlon XP 2000+ PC (768MB RAM), whereas the Positioning Gateway and the GIS server run on another AMD Athlon XP 1800+ (512 RAM) PC. Both PCs are located in the same local area network and use the Microsoft Windows 2000 Server operating system. Service invocations are performed from a remote computer (Pentium 4 1800 MHz with 256 MB RAM, running Microsoft Windows XP Professional Edition), which accesses the network via an ADSL 384Kbps line. The aforementioned topology is chosen because it fits well a real invocation scenario, where high data rates are experienced in the server domain, while the link between the end user and the invoker is significantly slower.

One indoor and one outdoor service were created and used for the tests. Both services performed an invocation to the positioning gateway to obtain the user's position. A second outdoor service which performed an additional request towards the GIS to retrieve the map of the surrounding area was also

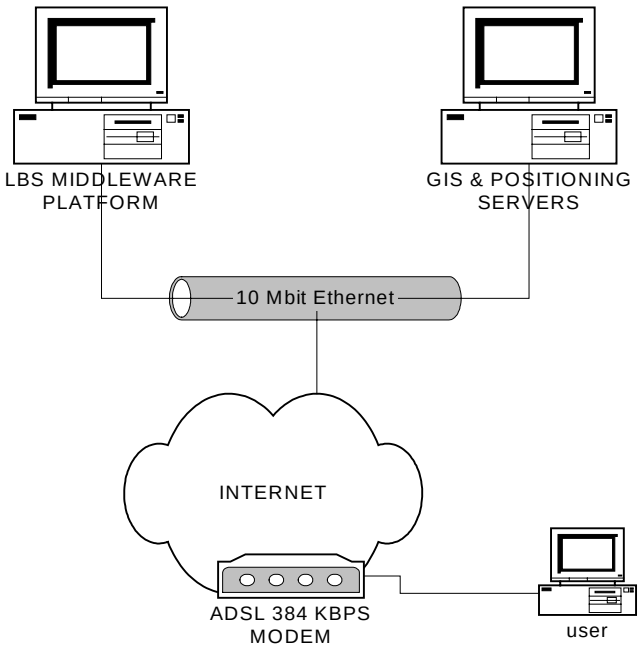

Fig. 9. Trial Network Setup.

created and tested (Fig. 10). For the indoor case no GIS invocation was possible, due to unavailability of indoor maps. Navigation services, although created, were not tested as their invocation is not deterministic but strongly depends on the two end points. After all, the performance analysis, was oriented mainly in the testing of the core system (i.e., the kernel and its interfaces towards external entities); so we did not want to incur in our tests the overhead produced by external systems, such as the GIS server.

6.2. Evaluation metrics

In order to study the system's performance, a series of trial scenarios were developed. Each scenario used a different type of service and aimed to assess the performance of our prototype, under different load conditions. In this perspective, each scenario consisted of a series of experiments, with an increasing arrival rate for incoming requests. The general scenario is as follows: A total of N users is assumed. Each user performs 1 to 3 services invocations. The user arrival rate (L) follows the exponential distribution. 60% of the users perform exactly one request, while the rest remain connected and perform a maximum of 3 requests (the number is randomly chosen for each individual user) before they leave the system. Time between two subsequent user requests follows the basic pareto distribution with the shape parameter set to 0.9 ($\alpha = 0.9$), thus modeling the thinking time of the user. Starting with a small number of users ($N = 10$) and a very low inter-arrival rate L (6 arrivals/min), the values of both N and L are increased until the platform fails to respond to a significant number of incoming requests. In order to be statistically correct, the Monte Carlo methodology [3], consisting of 10 repetitions of each experiment, was followed. The evaluation results, discussed in the following sections, are the statistical average of all 10 experiments.

Evaluation metrics that were used include: a) the success rate (% of total requests that were served successfully), and b) the average service execution time.

Fig. 10. Example service output produced during service invocation: (a) FindLocation service, (b) navigation service.

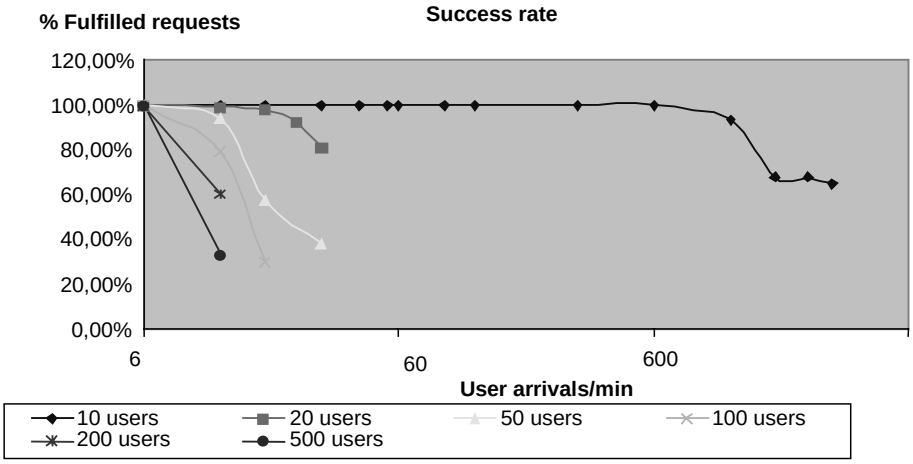

Fig. 11. Indoor location system success rate.

6.3. Evaluation results

6.3.1. Indoor services

In this section, results obtained from indoor services are presented. Figure 11 depicts the success rate for the indoor location service as a function of the number of users in the system (N) and the user arrival rate (L users/min). For $N = 10$ the success rate decreases when the users arrival rate (L) increases to over 1200 users per minute, while for greater values of N (20, 50, 100, 200 and 500 users), the performance of the system degrades faster (for values of L between 120 and 240 users/min).

The overall behaviour of the system for the indoor location service scenario is uniform, progressive and predictable. Degradation of performance is gradual and depends strongly on the system load. The

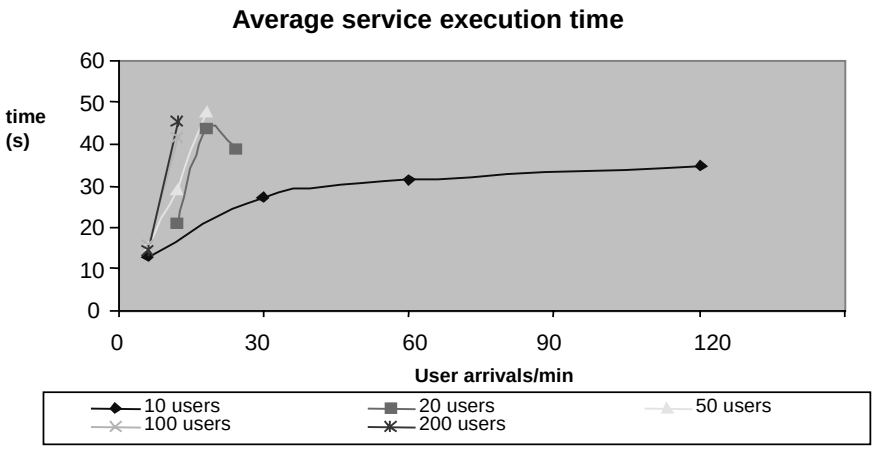

Fig. 12. Average service execution time.

statistical analysis of the results proved the existence of very small confidence intervals, which imply predictable behaviour as well as high reliability and precision for our measurements.

As presented in Fig. 12, the system's response time (the time from the moment the request is received until the moment the user receives the response) is increased gradually when the rate of incoming requests is increased.

6.3.2. Outdoor services

The results presented in this section, determine the performance of the outdoor services tested.

Outdoor Location Service (*with GIS invocation*)

The service retrieves the user's position and displays it on a map. As shown in Fig. 13, the service performs reasonably well, and the performance decreases gradually as the user arrival rate increases. It is remarkable though, that the system remains stable even when the user arrival rate exceeds the 100 users/min threshold.

The average service execution time, for this service, remains in acceptable levels and increases slowly as the arrival rate increases. As shown in Fig. 14, when the user arrival rate is low, the system responds in a few seconds (maximum response time in the order of 2–3 sec), which is an excellent performance. The response time degrades to over 20 seconds only for very high user arrival rates ($>$120 users/min) and a large pool of users ($N = 1000$).

Outdoor Location service (*without GIS invocation*)

In order to determine the possible cause of the delays experienced in the service invocation, we performed a series of profiling sessions. The results showed that the GIS component imposed a bottleneck for the system. To prove this, we developed a simplified service that did not performed a GIS invocation. We tested the performance of this service and the results are displayed in Figs 15 and 16. This service performance is significantly enhanced, and as shown in Fig. 15, the success rate is more stable, and degrades slowly while the user arrival rate increases.

A more obvious improvement was observed in the service execution time, which remained below 20 seconds even for very high arrival rates (Fig. 16).

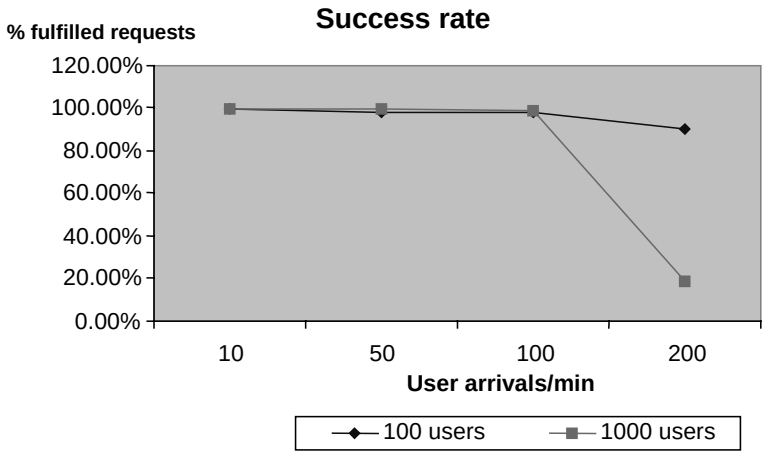

Fig. 13. Success rate for Outdoor Location Service.

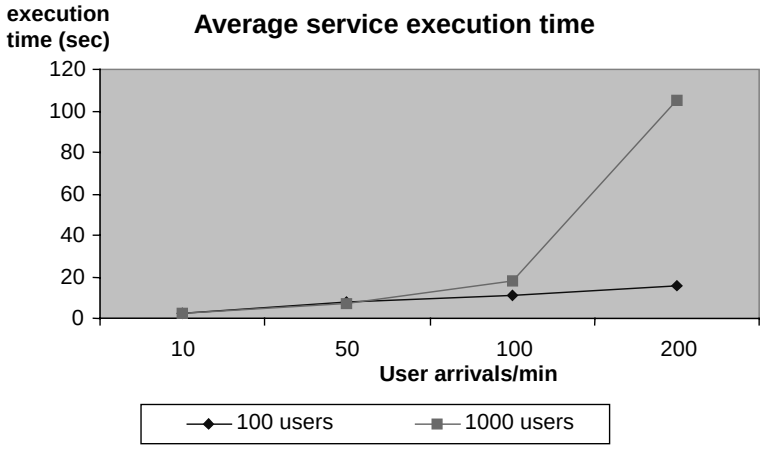

Fig. 14. Average Outdoor Location Service execution time.

6.4. Evaluation conclusions

In general, all tested services exhibited similar behavior, with slight variations. An issue that should be stressed here is that a lot of time in the outdoor service case is spent to the GIS calls. This results from the fact, that most of the processing that is required for providing the results of the service is performed by the GIS server (e.g. routing algorithms etc.) Consequently, in certain cases the LBS architecture has to wait for the answer of the GIS server, thus, resulting to significant delays in the service execution. Removing the GIS calls from the tested services showed significant improvement in both the success rate and the mean service execution time.

7. Conclusions

Service provisioning platforms comprise an area of major interest in contemporary telecommunication infrastructures. Mobile operators offer a continuously growing range of services towards their customers

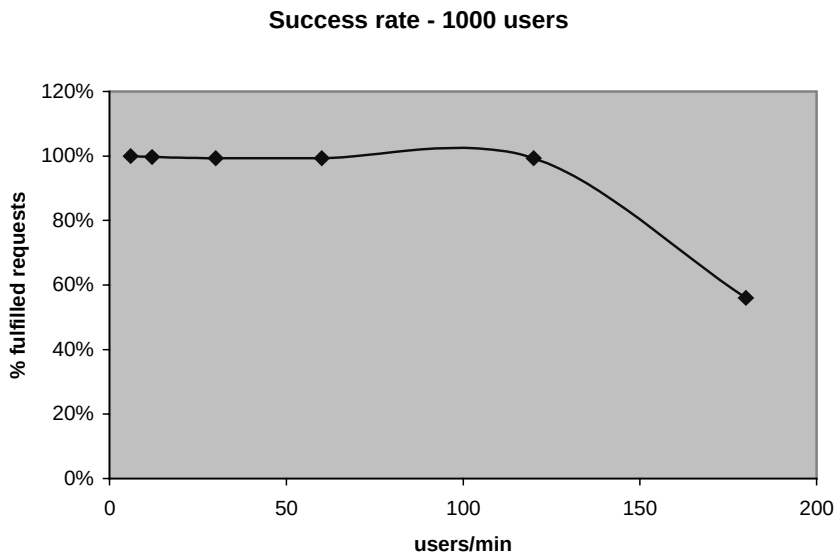

Fig. 15. Success rate for the simplified Outdoor test service.

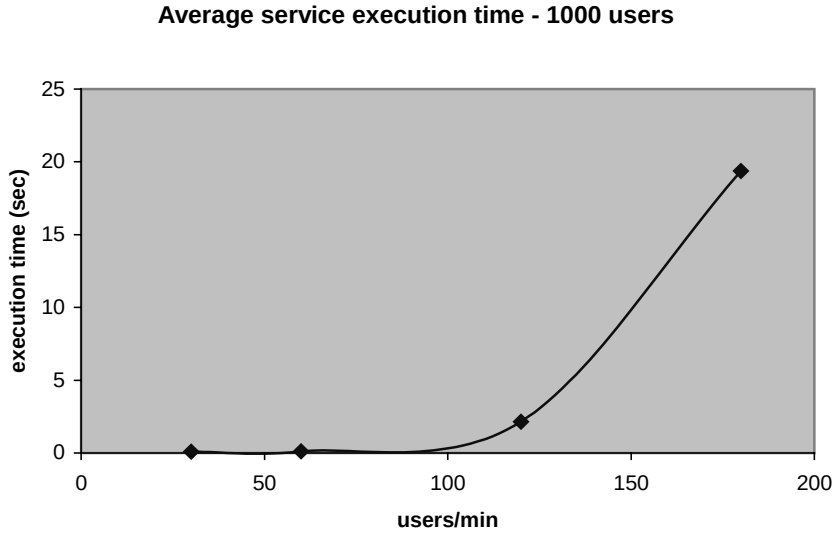

Fig. 16. Average response time for the simplified Outdoor Test service.

trying to attract a significant portion of the market. In this service-centric landscape, platforms capable of supporting the full life cycle (creation, provision, management) of a service, like the one presented in this article, will play a critical role. The presented platform offers important tools to its owners (actors like mobile operators and application service providers), allowing them to develop, test and provide LBS with little effort and high reliability. The platform was built using open standards in both its software components (e.g., Java, XML), and interfaces towards third parties (i.e., service creators, positioning systems, end users). The platform adopts a modular and easily maintainable architecture, which enables effortless adaptation to the new positioning technologies and application protocols, without affecting the

core mechanisms. The architecture of the platform allows fully distributed installation and operation, by different actors, that have to collaborate in order to deliver a complete LBS. Mobile operators, service developers, service providers and content providers can use the platform as a coordinating entity that will enable them to pool their resources together, and deliver state of the art services to the user.

A prototypical implementation of the proposed framework was developed and used for running a series of pilot services. The tests provided the basis for analyzing the performance of the prototype. The results demonstrated that the behavior of the system is predictable, although its performance degrades severely when high arrival rates are experienced. However we believe that the conclusions produced by the evaluation of the system will further assist our research in order to identify the bottlenecks of the system and improve its performance. An already identified bottleneck point is the GIS subsystem, whose presence is unavoidable for delivering the full range of LBS applications. Out future intent is to create a new version of the prototype capable of running in a clustered environment. This will surely improve the performance of the system but we also hope that it will reveal potential new problems or inefficiencies that were not initially considered and will assist us in the enhancement of the system's functionality.

Acknowledgments

This work is supported by the PYTHAGORAS programme of the Greek Ministry of National Education and Religious Affairs (University of Athens Research Project No 70/3/7411). The project is co-funded by the European Social Fund and National Resources (EPEAEK II).

References

[1] A. Ioannidis, M. Spanoudakis, P. Sianas, I. Priggouris, S. Hadjiefthymiades and L. Merakos, *Using XML and related standards to support Location Based Services*, in the proceedings of SAC'2004, 19th ACM Symposium on Applied Computing, Web Technologies and Applications Special Track, Nicosia, Cyprus, March 2004.

[2] ETSI ES 202 915-6 v1.1.1: Open Service Access (OSA); Application Programming Interface (API); Part 6: Mobility SCF, 2002.

[3] G.S. Fishman, *Monte Carlo Concepts*, Algorithms and Applications, Springer Verlag, 1996.

[4] G. Marias, N. Prigouris, G. Papazafeiropoulos, S. Hadjiefthymiades and L. Merakos, *Brokering Positioning Data From Heterogeneous Infrastructures*, (Vol. 30), in Kluwer Wireless Personal Communication (Special Issue on Location Based Services and Technologies), Issue: 2–4, September 2004.

[5] G. Papazafeiropoulos, N. Prigouris, I. Marias, S. Hadjiefthymiades and L. Merakos, *Retrieving Position From Indoor WLANs Through GWLC*, proceedings of the IST Mobile & Wireless Communications Summit, Portugal, June 2003.

[6] I. Priggouris, S. Hadjiefthymiades and G. Marias, in: *Chapter 14: Location-based services in Emerging wireless multimedia services and technologies*, N. Passas and A. Salkintzis, eds, Wiley, West Sussex, England, 2005.

[7] J. Bakker, *Next Generation Service Creation Using XML Scripting Languages*, Proc. IEEE ICC, April/May, 2002.

[8] J. Benedicto, S.E. Dinwiddy, G. Gatti, R. Lucas and M. Lugert, *GALILEO: Satellite System Design and Technology Developments*, European Space Agency, November 2000.

[9] J. Lennox and H. Schulzrinne, *Call Processing Language Framework and Requirements*, RFC 2824, May 2000.

[10] V. Tsetsos, O. Sekkas, I. Priggouris and S. Hadjiefthymiades, *A Scheduling Framework for Enterprise Services*, accepted for publication in the Journal of Systems and Software (JSS), Elsevier Science, 2005

Ioannis Priggouris received his B.Sc. in Informatics from the Department of Informatics & Telecommunications at the University of Athens in 1997 and his M.Sc. in Communication Systems and Data Networks from the same Department in 2000. Over the last years he has been a PhD candidate in the department. Since 1999, he is a member of the Communication Networks Laboratory (CNL) of the University of Athens. As a member of the CNL he has been involved in several research projects, implemented in the context of IST (e.g., EUROCITI, PoLoS, etc.) as well as many National projects. His research interests

are in the areas of mobile computing, QoS and mobility support for IP networks. He is the author of many publications in the aforementioned areas.

Dimitris Spanoudakis received his B.Sc. in Electronic and Computer Engineering from the Department of Electronic and Computer Engineering, Technical University of Crete, Chania, Greece, in 2002, and his M.Sc. in Communication Systems & Networking, from the Department of Informatics and Telecommunications, University of Athens, Athens, Greece, in 2005. He has performed research on network traffic analysis and modelling, as well as on architectures for mobile and wireless systems. He has been member and administrator of the Information System Center, Technical University of Crete (ISC-TUC) from 1997 till 2000, and member of the Telecommunications Laboratory and the of Technical University of Crete, from 1999 till 2002. Since 2004 he has been a member of the he has been a member of the Communication Networks Laboratory of the University of Athens (UoA-CNL). His current research interests are in the areas of network modelling, mobile computing, and autonomic communications.

Manos Spanoudakis received his B.Sc. in Computer Science and Telecommunications from the Department of Informatics and Telecommunication, University of Athens, Athens, Greece in 2000 and his M.Sc. (with honours) in Computer Systems Technology from the same Department in 2003. He is currently a Ph. D candidate at the same Department, specializing on Content Distribution Networks. He has performed research on WWW architectures for wireless/mobile systems and caching. Since 2000, he has been a member of the Communication Networks Laboratory of the University of Athens (UoA-CNL). He was involved in the IST projects EURO-CITI (EUROpean CITIes platform for on-line transaction services) and PoLoS (integrated Platform for Location-based Services).

Stathes Hadjiefthymiades received his B.Sc., M.Sc., Ph.D. degrees (in Computer Science) from the University of Athens (UoA), Athens, Greece and a Joint Engineering-Economics M.Sc. from the National Technical University of Athens. Since 1992, he was with the consulting firm Advanced Services Group. He has been a member of the Communication Networks Laboratory of the UoA. He has participated in numerous EU-funded and national projects. He served as visiting Assistant Professor at the University of Aegean, Dept. of Information and Communication Systems Engineering. He joined the faculty of Hellenic Open University (Patras, Greece) as an Assistant Professor. Since December 2003 he belongs to the faculty of the Department of Informatics and Telecommunications, UoA, where he is an Assistant Professor. His research interests are in the area of mobile/pervasive computing and networked multimedia applications. He is the author of over 100 publications in the above areas.