Abstract

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 a generic framework for delivering Location Based Services (LBS). The framework is capable of providing the full functionality required for delivering 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 calling user. The main focus of the paper is on the technical specification, the design and the functionality of the framework. However, with the purpose of assessing the proposed architecture, a prototype implementation based on the discussed specifications was built and its performance was evaluated using a series of pilot services.

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, on which both manufacturers and mobile operators have invested a lot [2]. However delivering new services requires developing means and tools that will assist in their creation, provision and maintenance.

This paper presents an extensible cross-platform framework, which facilitates the provision of LBS services with minimum effort from all involved sides (e.g. service provider, mobile operator etc.) and with no change to the wireless Internet infrastructure. Moreover, the framework is based on open standards so that it can accommodate future evolved technologies and be fitted in future telecommunication infrastructures.

The rest of the article is structured as follows. Section 2 lists the requirements for a service provisioning platform. Section 3 provides an overview of the platform and of the involved technologies. Section 4 dives deeper in the architecture of the system. The article concludes with the performance evaluation of the proposed platform, followed by the conclusions.

2 LBS provisioning platforms requirements

The development of a provisioning platform, which would cover all aspects of LBS provisioning is a primary goal of all vendors involved in the world of LBS software systems. The set of requirements and desired characteristics for such platform includes:

- Means for supporting the service creation process without the need for special programming skills.
- Support for service deployment and operation through automatic procedures, which do not require special IT personnel or programming skills.
- Service provisioning through a variety of access protocols so that the service is accessible from different networks.
- Portability over different operating systems and hardware platforms, so that integration to different infrastructures is possible.
- Reusability: The same platform may host a number of different services, with different requirements and functionality. The introduction of new services should not require changes to the platform and changes to the platform should not affect the execution of existing or future services.
- Access to the system should be possible through a variety of devices, using different transport protocols.
- Independence from underlying positioning and GIS technologies: The platform should not bind to any specific networking technologies. GSM/GPRS and WLAN interfaces should be treated in a unified way, while the introduction of UMTS and 3G networks will not require changes to either services or the platform. This also applies to the GIS interface which should be open enough so that integration with different implementations to be possible.
- Security. Interfaces with external entities have to be secure and privacy should be maintained. These characteristics will make the user feel safer and as a result more receptive to using the platform.
- Roaming across different infrastructures. Both indoors and outdoors environments should be supported.
- Scalability. Possibility of hosting large number of services, each capable of serving numerous concurrent requests, is sure to enhance the marketing potential of the platform.

Currently, to the best of our knowledge, no integrated platform, which covers all these aspects, exists. Most of the existing platforms facilitate LBS operations by providing very simple functionality, which fulfils only a subset of these requirements. During the design of the discussed framework the goal was to cover all the aforementioned...
requirements and provide all these features that could maximize the benefit from the adoption of the provided system.

3 Framework overview

The proposed framework 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 and at the same time respect all requirements set in the previous section. The platform focuses on the development, deployment and execution of services and consolidates several technologies.

![Framework Architecture Diagram]

Figure 1. Framework architecture

A modular approach has been followed during the design of the platform resulting to an architecture, which defines a central processing unit: the kernel and independent functional entities on the boundary, each covering a certain operational area in the chain of LBS provisioning. These functional entities, along with the kernel and its interfaces, comprise the integrated middleware platform (Figure 1).

The autonomy granted to each component through this architectural approach makes possible the extension of the platform by adding new components or modifying the existing ones, in order to enhance the overall functionality, which was designed to facilitate the creation of location-based services.

In order to understand the philosophy behind the proposed framework we will provide in the next section an overview of the technologies, involved in delivering location-dependent services. These technologies, more or less, dictate the functional areas that need to cover a platform aiming to provide Location-based services.

3.1 Positioning Technology

Positioning technology [10] is a key point in the LBS context. Research in this area during the last years is impressive. A plethora of solutions that provide accurate estimations of user’s location have emerged, each with distinct characteristics and capabilities and hence applicable in different circumstances. Two main categories of positioning solutions are available today:

- Satellite Based such as the Global Positioning System (GPS) [11], which provide high accuracy with the drawback of needing additional equipment on the terminal side. A newcomer in this category is ESA’s Galileo [9], which promises accuracy similar if not superior to GPS.
- Terrestrial Infrastructure-Based Solutions which are less accurate than the satellite based ones; however despite this drawback they are widely used, as they don’t require expensive add-ons to the terminal devices (i.e., GPS receivers).

A certain shortcoming of the most popular positioning systems is their inability to produce accurate estimates in inside-building environments. However, lately, significant progress has been achieved in the area of WLAN based location technology [1],[12] solving somehow this problem.

3.2 GIS Technology

The Geographic Information Systems (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, by incorporating detailed visual representations (e.g., maps) or specific info that is otherwise difficult to be acquired (e.g., points of interests etc).

The GIS represents a comprehensive database related to positional information with an integrated set of tools for querying, analysing, and displaying information. A standard RDBMS (e.g. Oracle or MS Access) is usually used for storing the spatial info. Numerous data formats have been defined for storing GIS data in these databases (geotiffs, shapefiles, RDFs etc.). Apparently each data format needs specific software in order to be read and further processed, resulting to a GIS market flooded with proprietary GIS software packages. 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 posed towards a GIS system includes:

- Navigation requests
- Proximity requests
- Find location requests
- Geocoding requests

As delay is a crucial factor for such requests, existing GIS software implements algorithms for handling all abovementioned queries with the minimum possible delay.

3.3 Interfaces Technologies

The term “interfaces technologies” refers to the transport technologies used for communicating and transferring data between two end-points. As location-based services are strongly coupled with the mobile communications domain bearers that are used in the latter are of prime importance for transferring data between the platform server and the corresponding client. Until recently the exchange of data between 2 actors of the mobile infrastructure was limited to the SMS and the WAP technology. The former offers a basic transport protocol for textual data while the latter a
much more advanced protocol, which could be considered analogous to the HTTP protocol in the mobile domain. SMS is widely used today but WAP failed to gain wide acceptance mainly due to unavailability of bandwidth and cost-efficiency reasons.

Evolution towards 3G networks benefited greatly both technologies as the EMS and MMS technologies enhanced 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 the HTTP protocol in the mobile domain. In the WLAN environments the HTTP protocol remains the simplest and most promising technology for interfacing with servers and other remote systems.

4 Platform architecture

After this short review of enabling technologies in this section we proceed with the analysis of the architecture of the LBS provisioning system. A brief overview of the main components of the platform (Figure 1) is presented in the following section. Four main components have been identified, namely: a) the Kernel, b) the Interfaces Component, c) the Positioning Component and d) the GIS Component.

4.1 The Kernel

The Kernel is the heart of the platform. The role of the Kernel within the proposed architecture is to serve as the runtime environment for the services offered to users as well as to coordinate the operation of the peripheral components according to the directives of the service logic.

Moreover the kernel provides all the functionality required for deploying new services through automatic procedures. Interfaces for injecting new services in the Kernel have been provided. These interfaces can be invoked either through command line or they can be used to create advanced graphical IDEs that will assist the service creation process.

The internal architecture of the Kernel, which is illustrated in Figure 2 consists of a set of cooperating sub-modules designated to provide the full functionality needed during the life cycle of a service. An overview of these modules and their functionality is presented in the following sections.

4.1.1 The Service Deployer (with the compiler)

It is the entry point for all new services entering the Kernel. The Deployer provides methods needed for the deployment of new services on the platform. The same interface allows external clients to query the status of already deployed services, or to get information about a specific service. The Service Deployer receives the service specification and produces an executable form of the service which can run inside the Service Execution Environment.

4.1.2 The Service Invocation Module (SIM)

The Service Invocation Module (SIM) receives all incoming requests and provides a standard interface towards the Interfaces component. An XML protocol that enables this communication has been designed. This protocol was designed taking into account the efficiency requirements imposed by the real-time communication, therefore it is simple and compact, but at the same time flexible enough in order to accommodate all information for service execution and handling of responses. The DTD of the protocol is presented in Figure 3.

In order to cope with large number of concurrent requests the SIM has been designed as a module, which incorporates a generic, transparent mechanism for invoking all possible services. The general functionality provided by the SIM has as follows:

a) Reception of incoming requests in the form of an XML message
b) Processing of the message to deduce the service to be invoked and its parameters
c) Consultation of the service registry
d) Invocation of the service
e) Packaging of service results according to the XML protocol
f) Dispatching the results back to the Interfaces Component in the form of an XML message

4.1.3 The Service Execution Environment (SEE)

The Service Execution Environment (SEE) acts as the runtime context for the deployed services. It is also responsible for coordinating communication of the deployed
service with components residing externally to the SEE either inside the Kernel (i.e., the Scheduler) or outside (i.e., the Positioning and GIS peripheral components). Specific client modules inside the SEE namely the Component handlers undertake the task of translating the service control logic directives to synchronous method invocations on the aforementioned components. Concept-mechanisms such as configuration variables, variables shared between different instances of the same service, variables that persist between successive invocations of the same service for a specific user are also supported by the SEE through the definition of specific repository areas. These repositories are created per service and can directly be accessed inside the service logic.

4.1.4 The Scheduler

The role of the Scheduler is to support the automatic execution of services within the platform. It features 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.). 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 building block), with the minimum possible delay. This is part of the functionality provided by the OSA/Parlay mobility specification [13] and is also particularly interesting in scenarios of personalized 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 framework is multithreaded which guarantees, that even concurrent events are executed with the minimum possible delay. Particular effort during the design of the Scheduler was focused on the persistency of the scheduled tasks. Hence, aiming to maximum reliability the Scheduler features internal mechanisms for storing information pertaining to each scheduled task in order to cope with potentials crashes of the platform.

4.1.5 The Registry

The Registry is an integral part of the Kernel’s architecture. It is implemented using Directory Services, thus allowing for the swift retrieval of the stored data. The Registry consists of two smaller registries (sub-registries): the Service Registry which contains service-specific data and the User Registry where user-specific data are stored each one represented as a different branch under the root node of the Registry.

When a new service is successfully deployed on the platform, the Service Deployer updates the Service Registry with the respective information. This information is used for the invocation, undeployment and redeployment of a service.

4.2 The Peripheral Components

The peripheral components take their name from their relative position with respect to the Kernel. The functionality of each component features a well-defined API for interfacing with the Kernel. This modular architecture results in autonomous and independent components, which can be plugged out and replaced at any moment with others implementing the same APIs without the Kernel noticing any difference. As shown in Figure 1, three peripheral components have been incorporated, i.e. the Positioning component, the GIS component and the Interfaces component.

4.2.1 The Positioning Component

The positioning component (Figure 4) is responsible for providing location information to the Kernel. It integrates the underlying positioning technologies, providing a unified interface through a well-defined API. It abstracts the differences that are introduced by the diversity of the underlying location tracking techniques. This interface is fully configurable through a variety of parameters intended to cover all possible cases and operation paradigms.

Communication with the underlying network infrastructure is based on open interfaces, such as OSA/Parlay [13], guaranteeing both the independence from the underlying location technology as well as the openness and extensibility of the component. The proposed implementation of the Positioning component includes support for location retrieval from 2G/3G networks and WLAN [4] environments. GPS support also exists through the use of specific software client-modules running on the terminal side for pushing location data towards the corresponding server module residing inside the positioning component. The framework support for various positioning technologies allows the seamless provision of LBS across different infrastructures and facilitates user roaming.

The main design orientations of the Positioning component were extensibility and openness. Therefore a number of core modules, implementing basic functions have been incorporated. Functions, like multiplexing, routing or scheduling are mandatory steps that every positioning request has to go through, depending on its type. Functionality specific to the positioning techniques has been placed in separate modules, which are called wrappers, as their task is to wrap the peculiarities of the different
positioning system in order to provide a single interface towards the Kernel.

4.2.2 The GIS Component

The GIS Component is responsible for the interaction of the Kernel with external GIS repositories. It covers visual and textual information retrieval and preparation in order to create representations (e.g., maps) for a certain geographical area. The GIS component implements algorithms like navigation routing and geo-coding, which render it capable of answering both simple and sophisticated requests. It actually consists of two modules: a client module, which resides within the Kernel and a server module, which interfaces with the GIS repository. All needed functionality is incorporated inside the latter, while the former is merely charged with the task of propagating the requests coming from the SEE (Service Execution Environment) and receiving the answers of the server module.

4.2.3 The Interfaces Component

The Interfaces component is responsible for performing the communication of the Kernel with external entities (i.e., the user’s equipment) irrespective of the underlying transport technology, which is used as bearer for the user’s request. Its current specification supports three established and commonly used protocols (protocol suites), namely SMS, WAP and HTTP. However its design is modular enough so that support for additional protocols can be easily integrated in the future. The Interfaces Component features multiple interfaces with Proxy Servers (e.g. HTTP Server, WAP Gateway or SMS Center (SMSC)) in order to interact with different types of terminal devices. Proxy Servers can be part of the framework infrastructure; although their external placement is not precluded (e.g. the SMSC would normally belong to a mobile network operator).

4.3 The Service Creation Environment (SCE)

The Service Creation Environment (SCE) comprises an Integrated Development Environment (IDE) supported by a GUI, whose goal is to facilitate the creation and deployment of new services. The SCE featuring a graphical interface, and customized support for the LBS-oriented language (SCL), can provide significant support to inexperienced users of the platform for creating, deploying and debugging new services. For more adept users a set of command-line tools is provided. These tools, as already discussed, interface with the web-services interface exported by the Service Deployer and the SEE inside the Kernel, allowing queries about deployed and debugged services or deployment requests to be posted. The IDE part of the SCE is based on the Eclipse software platform [14], an open source, extensible graphical framework, provided by IBM, which enables customized GUIs to be developed. Eclipse supports, also, the Graphical Editor Framework (GEF) [15], for developing graphical (or visual) editors, where service logic is presented to the users as an execution flow diagram. Within this diagram it is possible to add and remove commands corresponding to SCL directives, using a simple drag’n’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 defined, resulting to one-to-one correspondence with an SCL directive. The standard text editor is also available for a conventional development. Switching between the two environments during the creation of the services is also possible. An example of the same service as displayed in the editors can be seen in Figure 5. It is evident that the SCE is an add-on to the platform middleware, and its absence does not restrict the capabilities of the platform. Moreover, the open web-services interface provided by the Kernel allows potential users to develop their own proprietary SCEs for facilitating the service creation process, thus proving the openness of the middleware.

4.4 The Service Control Language (SCL)

The Service logic for applications designed to run using the platform middleware is specified through a new language, SCL (Service Control Language) developed specifically for supporting location-based services. We will provide a brief overview of its grammar and syntax through a simple example.

SCL is an XML-based language, built to facilitate implementations with simple functionality but also flexible enough to have the capabilities of a general-purpose language, similar to other scripting languages. Figure 6 presents the SCL code of a simple service, named “GetMyLocation”. This service defines a single “Main” method, with certain security constraints. Specifically, the method is accessible only to registered users with the specified role (“super”) enabled. The “posdata” variable is defined using the “set” directive. The variable takes the value returned by a method invocation performed 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, similar to most scripting languages, does not provide strong type checking and variables are cast to appropriate types dynamically during execution.
The next invocation is towards the GIS component, where the “getLocationInfo” service is requested. The contents of the “posdata” variable, which contains the result of the previous invocation, are used to define the parameters of the new one. The expected result is assigned in the variable “gisdata” and is next used for defining a new variable with the special name “result”. The “result” variable is reserved by the language 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.

5 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 [6] and the EJB 2.0 specification [3]. The kernel and the peripheral components were developed on top of the JBoss application server [8]. For position retrieval an emulation of an OSA-compliant positioning gateway was developed using Apache Tomcat 5.5 and the Web Services Axis toolkit. Finally, a minimal GIS server, with a simple Find Location was built using the Java map objects software development kit.

5.1 Trial Setup

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 created and tested. For the indoor service no GIS invocation was possible due to unavailability of indoor maps.

5.2 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 service 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 [5], consisting of 10 repetitions of each experiment, was followed. Final results 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.
5.3 Evaluation Results

5.3.1 Indoor Services

In this section, results obtained from indoor services are presented.

Figure 8 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 behavior 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 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 Figure 9, 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.

5.3.2 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 Figure 10, 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 acceptably low levels and increases slowly as the arrival rate increases. As shown in Figure 11, 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 goes 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 perform a GIS invocation. We tested the performance of this service and the results are displayed in the following figures. This service performance is significantly enhanced, and as shown in Figure 12, the success rate is more stable, and degrades slowly while the user arrival rate increases.

As presented in Figure 9, 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.
A more obvious improvement was observed in the service execution time which remained below 20 seconds even for very high arrival rates (Figure 13).

![Figure 13: Average response time for the simplified Outdoor Test service](image)

5.4 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 both indoor and outdoor services is spent in 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.

6 Conclusions

This paper presented issues concerning the design of a generic architecture for delivering location-based services. The focus of the paper is on the principles that drove the design of the proposed framework. Its internal architecture is also presented.

A prototypical implementation of the targeted LBS was developed and used for running a series of pilot services. The tests provided a basis for analyzing the performance of the prototype. The results demonstrated that the behavior of the system is predictable, although its performance seems to degrade 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 improve the performance of the system. The architecture presented in this paper is only an experimental prototype, and as such, can not be compared to similar commercial systems. Our future work includes the development of a more robust version of this prototype, which can be directly comparable to other similar systems. We also plan to develop a navigation service, and use it to perform an exhaustive testing of the prototype using it. The complexity of such a service is expected to yield a better estimation of the system’s capabilities.

7 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)

8 References