The “Le” interface: Performance Evaluation of 2-tier and 3-tier 3GPP compliant realizations

S. Kapellaki, N. Tselikas, E. Chaniotakis, E. Koutsoloukas, G. Prezerakos, I. S. Venieris
Intelligent Communications and Broadband Networks Laboratory
School of Electrical and Computer Engineering
National Technical University of Athens
9, Heroon Polytechniou str, Zografou, Athens, Greece
e-mail: {sofiak, ntsel, emchan, lefterisk, prezerak}@telecom.ntua.gr, venieris@cs.ntua.gr

Abstract - Location-based services (LBS) constitute one of the most attractive services’ categories for cellular networks. 3GPP defines the “Le” interface for external LBS provision and access to the Core Network of mobile operators. Multiple proprietary and commercial protocols and APIs can be used to implement this interface. The paper examines the realization of “Le” interface according to the proposed 3GPP standards, namely the Mobile Location Protocol (MLP) over HTTP and the Mobility API of OSA/Parlay using middleware technologies. The performance of both approaches is evaluated. Furthermore, with respect to the Mobility API of OSA/Parlay, three implementations, based on respective middleware technologies (CORBA, RMI and Web Services over SOAP), are examined, for evaluating the performance of each OSA/Parlay Gateway. The results of the experiments provide useful performance indicators concerning the efficiency of the internal architecture of each implementation as well as the anticipated speed of the user-service interaction.

Keywords - Location Based Services (LBS); “Le” interface; middleware; OSA/Parlay; protocols; Application Programming Interfaces (APIs)

I. INTRODUCTION

Location Based Services (LBS) provision is usually based on the transfer of user location information to external service providers that cooperate with one or more cellular networks. User location information can be estimated from the core entities of the cellular network by applying numerous methods and techniques (e.g. cell-ID, TOA, E-OTD etc) [1]. More precise estimation can be achieved directly from the mobile terminal by using Global Positioning System (GPS), provided that the terminal is GPS enabled. Although positioning based on GPS is very powerful outdoors and assures enhanced location accuracy – down to a few meters – it is ineffective indoors, because buildings block GPS transmissions [2]. This phenomenon renders necessary the existence of the location estimation based on the core network entities of cellular networks.

Independently of the underlying positioning technique used in such a network, external LBS have to interact with the core network’s entities each time a user’s location is required. Section 2 of the paper presents the different approaches that are able to realize this kind of interaction. In Section 3, the proposed middleware-based implementation is cited and analyzed, while Section 4 evaluates three different distributed technologies able to support the implemented middleware solution, as well as the main differences regarding performance between the different middleware implementations and the plain protocols utilization. Conclusions of the paper are summarized in Section 5.

II. PROTOCOLS AND APIs REALIZING THE “LE” INTERFACE

The core network’s entity, with which external LBS interact regarding user’s location, is the Gateway Mobile Location Center (GMLC). GMLC is also known as the Location Server, which exposes the information of users’ location to the external LBS, while the Radio Network Controller (RNC) in conjunction with Mobile services Switching Centre (MSC) in circuit-switched networks and the Serving GPRS Support Node (SGSN) in packet-switched networks are the responsible entities which trace a user’s location back to the Location Server. External LBS and the Location Server interact via the “Le” interface as defined by 3GPP [3]. Thus, the “Le” interface allows the external LBS to query a Cellular Wireless Network (2G, 2.5G or 3G) for user location estimation and provides the LBS with the corresponding results.

Different approaches can be followed for realizing the “Le” interface. Proprietary and vendor specific protocols and APIs are frequently used. The Mobile Positioning Protocol (MPP) is proposed by Ericsson. It is an XML-based protocol implemented over HTTP [4]. Another proprietary solution is the Alternis API for the exposure of GMLC functionality to a LBS platform provided by Alternis [5]. Proprietary implementations result to incompatibility in LBS provision across multiple networks, since each cellular network is obliged to follow the approach of the GMLC provider regarding “Le”. For compatibility and open service provisioning reasons, 3GPP proposes two solutions for the realization of “Le” interface. These are the Mobile Location Protocol (MLP) of Location Interoperability Forum (LIF) and an open API, the Mobility API of Open Service Access/Parlay (OSA/Parlay) forum.

MLP is an application level XML-based protocol that is used by LBS clients to query the location of mobile stations (cellular phones, wireless personal digital assistants, laptops, etc.) [6]. The MLP has been designed with interoperability in mind, so it features a layered architecture, which decouples transport mechanisms from location services. The protocol becomes attractive to IP-based applications since the XML-based messages can be transported as HTTP/ SOAP payloads. The latter protocols can be used in the lower transportation layer of the MLP three-layered architecture. Above the transport layer, the core location elements layer defines all the common XML elements required by the services to depict location information and manage client-server communication. Location information is coded as a set of coordinates defining a point on a Coordinate System and the description of an uncertainty shape around that point. The top layer defines the MLP services. The MLP defines both immediate (synchronous communication in the request-response pattern) and report (asynchronous) operation modes for three types of services: Standard Location service, Emergency Location Service and Triggered Location Service. Triggered service uses only asynchronous responses.

On the other hand, OSA/Parlay forum has defined an object oriented architectural model consisting of multiple open APIs for homogeneous exposure of different underlying network functionality in an open and abstract manner to the service plane. The OSA/Parlay model requires the establishment of a middleware
layer supporting the execution of specified functional elements. Applications residing and executing in distributed nodes exploit remotely the exposed network resources. Common Object Request Broker Architecture (CORBA), Java (over the Remote Invocation Protocol – RMI) [7] as well as the Web Services Description Language (using the SOAP protocol) are middleware technologies that support the execution of Parlay APIs.

The entire specified interfaces are discriminated in two parts. The Network Side APIs expose the network functionality. Their implementation binds the interfaces with underlying protocols used. The Application Side of the interface is deployed by the applications in order to achieve a standardized interaction model between the two sides. OSA/Parlay interfaces apply a client-server communication pattern exploiting functionality and core services of the established middleware. They follow an asynchronous model, without affecting the application performance, by defining call back objects that receive the results of a method invocation.

Mobility API constitutes a subset of OSA/Parlay APIs with respect to location information exploitation by LBS [8]. The location of the user can be determined by either geographical coordinates (longitude, latitude, altitude) or by telecommunication network-related area identifiers (e.g. Location Area Identity, Cell Global Identity etc). In any case, geographical coordinates are accompanied by a set of parameters describing a so-called uncertainty shape, which the Mobility API of OSA/Parlay, as well as MLP, also supports. In short, the OSA/Parlay Mobility API consists of the following interfaces: User Location (UL) for geographical location, User Location CAMEL (ULC) for network related location information exploiting the CAMEL service environment, User Location Emergency (ULE) for location estimation after an emergency call and User Status providing the status of the user.

III. Reference Middleware Implementation

Two are the main parts of the experimental middleware-based implementation, which is depicted in Figure 1.

The first one is the Application Part, which hosts the service logic of the implemented LBS and implements the application side of OSA/Parlay Mobility API. The location based service which was used during the experiments is the “findFriend” service. According to its service logic, when a user is requesting the location of a friend, the service is executed and interacts with the network, in order to return the desired result (user’s location). In the Application Part of the implementation two main entities of the application side of OSA/Parlay Mobility API have been implemented. These are the IpAppUserLocation and the IpAppTriggeredUserLocation OSA/Parlay version 4.1 objects. The former initiates an immediate or periodic service request to the network concerning user location and implements the corresponding call-back methods. The latter sets the triggering criteria to the network regarding a LBS logic and implements the call-back methods, which are invoked by the network side when the triggering criteria are satisfied. For the “findFriend” LBS, in which direct user location requests are used, only the IpAppUserLocation object is required.

The second part of the experimental implementation is the Network Part, in which a Location Server Simulator implementing the Mobile Location Protocol (MLP version 3.0) over HTTP resides. The Location Server represents the GMLC functionality of a Cellular Mobile Network, since it is responsible to provide the location of users to LBS. Since the methods used by the Location Server as well as the required time in order to localize a user are out of the scope of this paper, the internal implementation of Location Server is not thoroughly analyzed.

The most important entity of the Network Part is the middleware-based Location Gateway, which mediates between the afore-mentioned Application Part and the Location Server. The Location Gateway instance (LocationGW) implements the network side objects of OSA/Parlay Mobility API (both IpUserLocation and IpTriggeredUserLocation of UL interface) exposing in an open way the underlying GMLC location functionality to LBS. It also supports the appropriate wrapping mechanisms to transform the OSA/Parlay methods’ invocations to MLP messages (which are supported by the Location Server) and vice versa. The arrived OSA/Parlay method’s requests are transformed using the features of MLPCreator to MLP messages. These messages are forwarded to the location server via the provided interface (ILocationServer). Specifically, the locationReportReq() OSA/Parlay method is mapped to the <slir> element of the MLP message. The same MLP element is used to translate the extendedLocationReportReq() method of the IpUserLocation interface. In the opposite direction, the location gateway receives the MLP results by the location server and sends them to the callback application side object. MLP messages are parsed and OSA/Parlay methods are constructed and invoked (using MLPParser features). The results contained to either the <sla> or <slirep> elements of the MLP messages are mapped to locationReportRes() or extendedLocationReportRes() OSA/Parlay methods. In the case the gateway identifies an error message within the <sla> elements of the MLP message, it calls the locationReportErr() method to the callback object. The above translations represent the core gateway functionality.

The recently released OSA/Parlay specifications, concerning the distributed technology used (i.e. CORBA, RMI, Web Services), have been taken into account. The described approach of OSA/Parlay Mobility API has been implemented with the usage of these middleware technologies. All implementations are evaluated in the following section. The objectives of the evaluation are multi-folded. The first objective lies on the performance comparison between the different middleware implementations, in order to define and specify the pros and cons of each one. Furthermore it is a big challenge to compare all three implementations with the existing proprietary one, in which the plain MLP is used. This evaluation will probably comprise the preliminary requirements and guidelines for further commercial implementation of a middleware location gateway.

IV. Performance Evaluation

Middleware performance does not depend only on internal architectural design and objects participating. Both the processing power of the node that hosts middleware and the network topology and capacity influence the performance. In addition, quality
characteristics of a middleware have their role in the system performance. If the system supports replication mechanisms, interoperability across heterogeneous platforms and other important characteristics, it is expected that it will have reduced performance compared to a simpler one [9]. One can argue that the final results are very specific and reflect a concrete network topology and setting. This could have been quite a strong argument in the case where the performance evaluation carried out in this paper was directed at providing absolute measurements. On the contrary, we are aiming at results that can provide relative performance indicators about the most appropriate middleware technology for realizing the “Le” interface among the ones examined.

Therefore, the following analysis focuses on the efficiency of the core middleware implementation without including factors related to network topologies such as i.e. how many routers the packets of a request traverse, in order to arrive to the server side. This will help to understand how each open middleware application affects the communication between the core LBS and the location server (GMLC). After reviewing the three middleware solutions, it is useful to compare them with the plain location protocol utilization, in order to identify if the benefits offered by open interfaces have excessive cost to system performance.

A. Experimental Description and Analysis

The test bed is hosted in a Local Area Network (LAN) offering the IP transport facilities. It comprises of two workstations, one – based on Windows XP, with a PIU 733 MHz processor – hosting the network side of OSA/Parlay implementation (i.e. the Location Gateway), as well as the Location Server that was used, and the other – in a remote Windows 2000 machine, with PIU 433 MHz processor – hosting the Application Part (i.e. the “findFriend” service logic. The client–server machines are attached under different IP interfaces of the LAN. In the case of SOAP usage, the Apache AXIS (v1.0) SOAP engine was used as the framework to develop and deploy the Web Services described by the OSA/Parlay specification. The Apache AXIS services were accessible through a servlet deployed into the Apache Tomcat (v4.1) container. Both network and application side services were developed and deployed by feeding the WSDL2Java tool provided by the AXIS framework. For the tests using RMI, the OSA/Parlay interfaces and data type classes were developed independently and the RMI naming service used was the Sun’s reference implementation provided with the Java runtime environment. All tests and development were based on JDK and JRE (v. 1.4.1). Finally, for the tests using CORBA, the OSA/Parlay interfaces and data type classes were developed and deployed with the IDL compiler provided by SUN in the standard edition Software Development Kit of the same version (v 1.4.1), which also includes the ORB and the Naming Service that were used.

For the GMLC implementation a Location Server Simulator that communicates using MLP over HTTP was developed, as explained before. The nature of the Location Server does not influence the middleware performance evaluation, as will be shown below. The Location Server is based on HTTP implementation and provides very basic MLP functionality. A predefined set of user locations is loaded at startup and each request is served with a randomly selected location from this predefined set. In order to produce the MLP response message, the MLP parsing is the minimum required to locate the user’s MSISDN identifier and to associate it with the selected location.

The generated requests from “findFriend” LBS to the location entity of the platform are modeled using the Poisson distribution. This ensures the independency of the entity utilization, inherited by the fortiety of LBS utilization [10]. A step further, the different sources that may cause delays can be identified and assigned time symbols (Figure 2).

![Figure 2. Experiments description and time values definition](image-url)

The times depicted in Figure 2 include the time spent in each processing step in both the request and the response direction. Specifically $t_{CL}$ is the time required for the client side processing, $t_{GW}$ is the time required for the request/response transmission and $t_{M}$ is the time consumed in the core middleware communication mechanisms. Finally, $t_{GW}$ is the time required by the Location Gateway part of the middleware and includes the delays caused by the wrapping of OSA/Parlay methods to MLP messages and vice versa, while $t_{GW}$ is the time required by the Location Server to return the location results to the Location Gateway.

Regarding the factors that affect each one of the times defined above, $t_{CL}$ depends on the internal Network’s Operator topology, the network traffic and other issues that are not related to the middleware application. An adequate consideration is that this time will be almost the same in the case of plain MLP utilization. The time identified as $t_{C}$ reflects the complexity of the used data structures and the additional processing tasks needed for the construction/deconstruction of MLP messages. The time identified as $t_{GW}$ in the case of SOAP, depicts the delays caused by the SOAP message parsing and the delays generated by the used Web server. In the case of the RMI utilization, $t_{M}$ represents the time consumed by the core RMI classes that undertake the role of emerging the request from the sockets to the application level. Regarding CORBA, $t_{M}$ equals the required time that the core CORBA classes need to perform the remote method call by generating the parameters to the application level, from their serialized form in the Internet Inter ORB Protocol (IIOP). It is obvious that $t_{M}$ is difficult to be measured, since it requires programming interference to the middleware implementation. Finally, the $t_{GW}$, in the case of SOAP, is the time required by the Location Gateway to forward the incoming OSA/Parlay request to the location server.

For the experiments, the location request sent from the client is considered to be one single operation and therefore the $t_{M}$ value depends on the network topology used in the experiments and does not include delays arisen by the middleware implementation. Finally, the $t_{GW}$ correlates to the time required by the client of the location server, the application-side implementation of OSA/Parlay APIs, to construct methods and data structures, in order to send the requests to the network side. In the opposite direction, the client receives the responses and returns to the “findFriend” LBS the distances results. This requires the transformation of coordinates received by the location server, and the application of geodetic equations, in order to identify the distance between users. The required processing effort is included in the $t_{GW}$. Although this procedure is common in cases of MLP and
open interfaces use, the deconstruction of complex data structures and the communication mechanisms differ in time, so the assumption that clients behave similarly for both cases is not valid.

From the above times definition and in order to evaluate different middleware’s implementation and performance we define the total time required by the Application Part to serve the “findFriend” LBS, as $t_{\text{TOTAL}} = t_{\text{CL}} + t_{\text{G}} + t_{\text{M}} + t_{\text{CL}} + t_{\text{C}}$. The total time comprises a first indication for middleware’s performance, since it takes into account all several parts of individual pre-mentioned times. Most of these times are middleware dependent ($t_{\text{CL}}, t_{\text{M}}, t_{\text{C}}$), while the rest ($t_{\text{G}}, t_{\text{C}}$) influence the system performance regardless of the middleware application. Thus, the assumption made for the latter is that they contribute similarly for all three tested middleware technologies.

In order to assess the gateway’s behavior over time, measurements were carried for $k$ consecutive time periods. We define the mean serving rate ($\bar{R}_{\text{GW}}$) of the gateway part as the ratio of the total gateway served requests ($r_{\text{ser}}$) to the total arrived requests ($r_{\text{arr}}$) during the whole period of observation:

$$\bar{R}_{\text{GW}} = \frac{\sum_{i=1}^{k} r_{\text{ser}i}}{\sum_{i=1}^{k} r_{\text{arr}i}}$$

This value is directly correlated with the performance of the middleware, since the gateway depends on the middleware technology used. Thus, $\bar{R}_{\text{GW}}$ indicates the response rate of the implemented middleware gateway and as a result its potentiality to avoid congestion or bottleneck phenomenon.

Also, we define the mean throughput ($\bar{S}_{\text{GW}}$) of the gateway as the ratio of the successfully passed location requests ($r_{\text{suc}}$) through gateway to the total arrived requests ($r_{\text{arr}}$) to it, in the same period of observation:

$$\bar{S}_{\text{GW}} = \frac{\sum_{i=1}^{k} r_{\text{suc}i}}{\sum_{i=1}^{k} r_{\text{arr}i}}$$

The $\bar{S}_{\text{GW}}$ value indicates the successful penetration possibility through the gateway.

B. Experimental Results

In the following, the results of the performed experiments are explained and discussed. As pre-mentioned, the mean value of $t_{\text{TOTAL}}$ is estimated for all four cases; plain usage of MLP protocol, as well as for the usage of SOAP, CORBA and RMI respectively for OSA/Parlay implementation. For the latter (SOAP, CORBA and RMI OSA/Parlay implementation), $\bar{R}_{\text{GW}}$ and $\bar{S}_{\text{GW}}$ are also estimated, in order to provide stronger claims to conclude for the most appropriate and convenient middleware implementation.

As afore-mentioned, the used model for the generation and arrival of requests is based on the Poisson distribution. Five cases of 20, 40, 60, 80 and 100 requests per minute respectively, as the $\lambda$-value of Poisson rate, were distinguished. Each experiment lasted about three hours, in order to have as reliable as possible results using the Poisson distribution.

In the cluster columns chart in Figure 3, the mean total serving time of each one of the five experiments is depicted. It is obvious that in the case of plain MLP usage, in $t_{\text{TOTAL}}$ value both $t_M$ and $t_G$ are not included, since no middleware mediates, while the communication is established over HTTP. This is the main reason causing the better results concerning $t_{\text{TOTAL}}$ in MLP measurements. On the other hand we observe OSA/Parlay Mobility API implementation with RMI requires almost the half time for a request-response life cycle, in comparison with the corresponding SOAP one. The variation mainly reflects the differentiations between the following three individual time values, $t_{\text{G}}, t_{\text{C}}$ and $t_{\text{CL}}$, for each middleware technology, even if $t_{\text{C}}$ and $t_{\text{G}}$ fluctuate in equivalent levels. CORBA performs even better than RMI, requiring approximately one quarter of the total time required in the SOAP implementation and half the time required by the RMI implementation. This fact suggests that the CORBA implementation that was used is more efficient than RMI in areas such as thread handling, message parsing and naming service lookup procedures. In the SOAP implementation, $t_{\text{M}}$ is higher in contrast to the other two middleware technologies, because apart from the required time for the establishment of middleware communication mechanisms, it also contains the delays resulting from the Web Server, as well as from the parsing of SOAP messages. Furthermore, the higher $t_{\text{CL}}$ in the SOAP case is also justified, since the construction/deconstruction mechanisms of complex data structures - supported by the OSA/Parlay Mobility API - requires more time for their serialization/de-serialization by AXIS. Additionally, $t_{\text{C}}$ in SOAP implementation is affected by the HTTP transportation (in contrast to RMI and CORBA). Thus, for the transmission of the same information, SOAP uses bigger messages, which reflects in higher values for both $t_{\text{C}}$ and $t_{\text{M}}$.

![Figure 3. $t_{\text{TOTAL}}$ per arrival rate for SOAP, RMI, CORBA and plain MLP implementation](image-url)
\( S_{GW} \) appears. The main reason for this phenomenon is not the gateway’s inefficiency or any disability caused by the middleware. Most of the requests’ rejections are caused by host machines’ resources unavailability, as shown by the exceptions thrown at client’s side.

In all the implementations, multiple threads are generated in the network part, in order to serve multiple requests. The number of generated threads is restricted by the resources of the machine used to host the implementation of the network part. After a critical threshold for the total number of simultaneously live and active threads, the machine cannot produce new ones. Thus, the new entrant requests cannot be served due to the lack of threads. A part of the incoming requests is rejected, while the rest part is served by threads, which have finished the processing and can be reused. By using either more powerful host machines or appropriate queuing algorithms for the network part of the implementation, like the equipment that each network operator owns, throughput augmentation can be certainly achieved.

Coming back to the evaluation of \( S_{GW} \), the performance of CORBA and RMI is depicted a little higher than SOAP. Taking into account the previous analysis, this phenomenon is easily explained, since the SOAP requests require more processing time. Consequently, more threads are staying alive and active for more time, rather than in the other cases, and as a result, more requests are rejected by the gateway in the SOAP case.

V. Conclusions

The paper described the dominant options currently available for exploitation of location information by LBS. It focuses on the standardization of the “Le” interface. Trying to overcome the peculiarities, differences and restrictions of existing protocols and APIs, the establishment of open OSA/Parlay interfaces has been implemented in terms of an OSA/Parlay Location Gateway. The application of open middleware layer offers to applications an abstract view of network resources, without obligating the former to adapt their deployment to a specific protocol. Different distributed object technologies, like CORBA, RMI and SOAP over HTTP, have been applied and evaluated, in order to identify the most appropriate solution for the open middleware support. The different approaches were compared with plain MLP utilization.

The main rationale behind the three middleware implementations for LBS considered in this paper is the advantages that an open service API, such as OSA/Parlay, offers to Service Providers and Public Network Operators. The latter can create added value out of the network technology in a controlled and safe manner by hiding much of the complexity of network from service developers and providers. In a similar fashion, external Service Providers can exploit the resources exposed by the core network in a uniform way without being obliged to consider network-custom implementations. On the other hand, from the user’s perspective, attractive LBS should respond as fast as possible. Users are often annoyed at big delays. For such a case, CORBA and RMI are presented more advantageous against SOAP, since they shorten the network response time, and between the two, CORBA presents slightly increased performance.

Apart from the obvious performance differences between the middleware technologies, there are other points that should be considered. First of all, RMI is not an international standard, but an industrial one, and anyone that decides to use it is bound to the Java language. Even though Java is now available for the most popular platforms, code already developed in other programming languages cannot be reused. SOAP and CORBA on the other hand are presented as standards that guarantee interoperability between all platforms and implementations. However the current state of inter-Orb communication between different vendor’s implementations renders SOAP in practice a more standardized technology compared to CORBA. It may, therefore, be of benefit to the Network Operator to implement the “Le” interface using a middleware technology based on SOAP that is truly as open as possible to potential 3rd party vendors, despite the performance costs that SOAP incurs.

REFERENCES

[8] ETSI Standard ES 202.915-6, Mobility SCF of OSA/Parlay version 4.1, v. 1.2.1, August 2003