A Geo-Located Web Services Architecture for next generation mobile networks

André Claude Bayomock Linwa
Mobile Computing and Networking Research Laboratory (LARIM)
École Polytechnique de Montréal, C.P. 6079, Station Succ. Centre Ville
Montréal, Québec, H3C 3A7 Canada
Fax: (514) 340–5159 E-mail: acbayo@larim.polymtl.ca

Samuel Pierre*
Department of Computer Engineering
École Polytechnique de Montréal, C.P. 6079, Station Succ. Centre Ville
Montréal, Québec, H3C 3A7 Canada
Fax: (514) 340–5159 E-mail: samuel.pierre@polymtl.ca
*Corresponding author

Abstract: As many geo-located web services will be deployed in the future, the mobile clients will be interested in locating a specific application server based on requirements such as proximity, service cost per location area, bandwidth and server utilisation rate. This paper presents a middleware system named GLWSA (Geo-Located Web Services Architecture) that aims at satisfying these requirements as well as a thematic factorisation of common location functions used to get position of mobile clients. A GLWSA supports a set of GLWSMs (Geo-Located Web Services Manager) distributed over the mobile network. It defines protocols to discover and inform a Supplier Application Server (SAS) to migrate the service execution (of a specific client) to the nearest SAS based on the client’s location. This architecture is suitable to assist mobile clients in the discovering geo-located web services process and to maintain the service execution closest to their location context.

Keywords: geo-located services; next generation mobile networks; service discovery; web services.


Biographical notes: André Claude Bayomock Linwa earned his PhD degree in Computer Engineering at École Polytechnique de Montréal, in the Mobile Computing and Networking Research Laboratory (LARIM). He is currently Senior Architect in Business Solutions and Data Intelligence.

Dr. Samuel Pierre is currently Professor of Computer Engineering at École Polytechnique de Montréal where he is Director of the Mobile Computing and Networking Research Laboratory (LARIM) and NSERC/Ericsson Industrial Research Chair in Next Generation Mobile Networking Systems. He is the author of four books, coauthor of two books and ten book chapters, as well as over 300 other technical publications including journal and proceedings papers. His research interests include wireline and wireless networks, mobile
1 Introduction

A geo-located web service is a web service that is offered in a particular geographical region or area. In the execution phase, a geo-located web service uses the mobile position to deliver a service. For example, a geo-located web service could be an emergency application that informs the police or firefighter administration of a particular region about the location of a car accident, using the driver’s mobile phone position. In next generation (superior or equal to the third generation) of mobile networks, to obtain the mobile position, a geo-located web service will interact with the position or LoCation Server (LCS) of a mobile network operator (3GPP, 2002).

Referring to their specifications (3GPP, 2002), the geo-located web services can be deployed on the next generation mobile networks. Then, it will be possible for a Supplier Application Server (SAS) to get the location or geographical position of a mobile client. In this point of view, an SAS is considered as a set of web services that require the client position to deliver the service.

A geo-located web service generates a maintainability problem of service execution when a mobile client moves to a location area where the service is no longer offered by the SAS in execution.

The major challenge for a mobile client, as geo-located web services will be increasingly deployed in organisations (Ericsson, 2001; Schmandt and Marmasse, 2004), will be to discover a specific service corresponding to their requirements such as proximity, cost, network bandwidth or server utilisation rate and to maintain a service in execution.

Traditional protocols for discovering services (SLP, Jini, UDDI, Globe) are not adapted to the geo-located web services as the client position is not an entry parameter to find a service. The deployment of such services thus requires the definition of a service discovery protocol allowing, among others, to select an SAS based on the location context of the mobile client (the location context refers to the current geographical area or position of a mobile client), to migrate services in execution to the nearest or closest SAS (it is an SAS that offers the service in execution at the current location context of a mobile client) and to factorise common location methods based on thematic aspect (for example, locate all taxi drivers of ABC company within two miles of my current position, where ‘taxi drivers of ABC’ is a theme or subject). The selection of common locations based on thematic aspect is called a thematic factorisation.

Related work in this field does not challenge the problem of discovering geo-located web service and maintains the service execution in the location context of a mobile client. Current architectures are either adapted to discover services where fixed clients are involved (Guttman, 1999) and those which consider the mobility put the emphasis on code and agent mobility (Harashima et al., 2001; Michahelles et al., 2002) rather than
data or task mobility to the closest SAS (as the web services are service-oriented and use a client/server model) based on the location context of the mobile client and maintainability of service execution. Moreover, none of the architectures suggested in the literature offer a thematic factorisation of common location methods to a set of application suppliers.

This paper proposes a new middleware system called Geo-Located Web Services Architecture (GLWSA) which extends the UDDI (Universal Description, Discovery, and Integration) registry and the MLP (Mobile Location Protocol) (LIF, 2002) by adding the GLWSA topology and the factorised thematic location methods, respectively. The main objective of the GLWSA is to propose a distributed geo-located web services discovery system that allows to maintain a service execution closest to the location context of a mobile client.

In the remainder of this paper, we present the related work, the overview of UDDI and MLP protocols, the global architecture proposed, the functional architecture, the performance evaluation, the learned lessons and the conclusion.

2 Related work

Various approaches have been proposed in the area of discovering services dependent on the location context of mobile clients. Not all of these approaches consider the maintenance of the nearest service execution during client’s mobility. In Seydim et al. (2001), the authors defined an architecture for location-dependent query processing. The purpose of this work is to use a central middleware where services are published and discovered through a user service agent. A location-dependent service manager controls this system. It analyses the query and binds pseudo-codes used in the request to the correct predicates (e.g., ‘nearest’ can take the value ‘five miles’). It verifies the granularity of the query, dispatches the request to the corresponding databases and returns the results to the client in the desired format.

Devlic and Jezic (2005) proposed a discovering service architecture for mobile clients that uses the RDF (Resource Description Framework) developed by W3C (World Wide Web Consortium) and describes how to exchange metadata on the web. This architecture uses the user profile of mobile clients to inform subscribers about the client terminal status (idle/busy/detached) or the client communication status (available/not available/discreet). In addition to status functionality, the architecture offers the following functionalities: mobile location, service lookup, service publishing/subscribing and a map location representation. The main weaknesses of this architecture are the fact that the system distribution is not addressed and the nearest application server selection based on the mobile location context is not covered.

The MSD (Multiprotocol Service Discovery) architecture (Raverdy and Issarny, 2005) aims to integrate in the same system a set of service discovery protocols (e.g., Jini, Bluetooth SSDP, etc.) to allow mobile clients to use discovery services. The system represents a mobile network as a set of IP domains. Each IP domain is controlled by an MSD manager component, which interacts with SD (Service Discovery) plugins that are each mapped to a service discovery protocol. In the lookup service process, a mobile client interacts first with its home MSD manager that determines the service existence, then transmits the message to the appropriate SD plugin which returns the corresponding MSD description. If the service is not found locally, the MSD manager consults the SD
forwarding rules to address remote MSD managers that will continue the lookup service process. The problem with this architecture is the high cost of the service lookup if the process is forwarding in many domains and the system does not address the nearest server application selection based on a mobile client context.

In Harashima et al. (2001), an architecture named Application Module Request Broker (AMRB) is presented. This architecture enables user application to communicate with Application Modules (AM): location and migration. There are two kinds of migrations: host and code migration. The host migration uses a network location detection by sending periodic polling; code migration detects the migration of an AM. The location change is diffused in multicast to all AMRB of the system. The AM lookup is done first at the local home of a mobile client. If there is no match, the lookup request is forwarded to the next host until the system finds the qualified AM. This architecture can cost high when the AM lookup is forwarded to many hosts and is not adapted for web services technology.

In Panagiotakis and Alonistioti (2002), an architecture for reconfiguration control and service provisioning platforms was proposed. This is a middleware system which mediates between a provider service called Value Added Service Provider (VASP) and the network resources in order to deliver services to end users according to their location context. A published service is called VAS (Value Added Service) and is described in XML language. Each VAS contains information such as service name, category, keywords, implemented language, discovering location, service versions, service location, QoS required, application server address, cost per domain and pricing model. The service location is a geographical area where a VAS is offered. The main component of the proposed architecture is RCSPM (Reconfiguration Control and Service Provisioning Manager) which controls all interactions with the system. Services are published and discovered through a RCSPM. Mobile clients use a web portal for discovery services. In the service lookup request, the RCSPM returns to a mobile client one or many VAS based on the client terminal capability, the mobile location and the client required QoS. Each VAS returned indicates the service tariff per domain or location area. Then a mobile client can select the best VAS. During mobile clients’ roaming, the RCSPM informs the mobile clients of the service cost as they change location area. Although this strategy seems similar to our approach, it fails to maintain a nearest service execution compared to the location context of the mobile client and to use the thematic location.

Other projects, such as Globe, use an architecture called ‘Globe Housing Service’ (Ballintijn et al., 2003). This architecture allows locating the mobile users and services. Globe defines and implements distributed objects in a hierarchical topology tree. When a client looks up an object in a leaf node of a given location area, the object can be present or not. If it is absent, the system returns the contact address of the Globe object requested. This contact address redirects the request to the next node of the path tree. The procedure is repeated until the request is fulfilled. Globe is not adapted to discover services based on the location context of a mobile client. In Dongyan et al. (2001), a discovering service with QoS architecture is proposed. This architecture uses a hierarchical tree topology to discover the services. It registers the QoS obtained by users in a specific domain in the databases associated with each node of the path tree (leaf-to-root nodes). Thus, when a client requests services with QoS, the lookup service verifies the QoS criteria, first in the leaf node, then with the parent node if the service is not registered. This architecture is unreliable when the frequency of the stored QoS is low and is not location-dependent.
In the literature, the two most popular service discovery protocols are SLP and Jini. SLP (Service Location Protocol), a protocol developed by IETF, uses three agents: a UA (User Agent), an SA (Service Agent), and a DA (Directory Agent) (Guttman, 1999). On the other hand, Jini is a technology developed by Sun Microsystems for discovering services. Just as SLP, it involves three actors: the client, the service broker server, and the service supplier server (Govia and Barbeau, 2001). However, SLP and Jini protocols are not designed for mobile clients. In fact, they do not manage the discovery of services based on a client’s location context.

3 Overview of UDDI and MLP protocols

In order to better understand our contribution, we first describe UDDI and MLP protocols, which constitute the basis of the proposed architecture.

3.1 The UDDI protocol

The UDDI is a directory or a register of companies in order to publish web services and describe the manner of reaching each web service in a WSDL (Web Service Description Language) document. The UDDI fulfils the functions of white, yellow and green pages. A detailed description of the UDDI protocol is described in UDDI (2004). The UDDI is made up mainly of four entities: *businessEntity*, *businessServices*, *bindingTemplates* and *tModels*. The *businessEntity* describes a company or an organisation. It contains basic information about an organisation (name, description, category, person contacts, etc.), the *businessServices* that it offers and the URL addresses where its web services can be discovered. In an XML representation, the *businessEntity* envelopes all other entities. The *businessServices* is a collection of *businessService* entities offered by an organisation. Each *businessService* has one or more technical Web Service Descriptions captured in an XML element called a *bindingTemplate*. The *bindingTemplate* contains the information that is relevant for application programmes that need to invoke or to bind to a specific web service. This information includes the web service’s URL address, and other information describing hosted services, routing and load balancing facilities containing the data required to invoke a service. Before invoking a web service, it is useful to determine whether the specific service being invoked complies with a particular behaviour or programming interface. Each *bindingTemplate* element, therefore, contains an element called a *tModel* that has information, which enables a client to determine whether a specific web service is a compliant implementation. The protocol UDDI offers two sets of API (Application Program Interface): inquiry and publish API. The inquiry API looks up or searches for a *businessEntity*, *businessService*, *bindingTemplate* and *tModel* in the registry. The publish API verifies credentials of a publisher, saves or deletes a *businessEntity*, *businessService*, *bindingTemplate* and *tModel* in the registry.

3.2 The MLP protocol

The Mobile Location Protocol (MLP) is an application-level protocol that uses XML messages for querying the position of mobile stations independent of underlying network technology (UMTS, CDMA-2000, or WGS-136). The MLP serves as the interface between a location server and an application server (LIF, 2002). The MLP protocol has
three main layers: service, element and transport. On the lowest level, the transport protocol defines how XML content is transported. Possible MLP transport protocols include HTTP, SOAP and others. The element layer (second layer) defines all common elements used by the services in the service layer. The service layer (top layer) defines the services offered by the MLP. The services are classified into five categories: standard location immediate service, emergency location immediate service, standard location reporting service, emergency location reporting service and triggered location reporting service. The standard location immediate service is a standard query service with support for a large set of parameters. This service is used when a single location response is required immediately (within a set time). The emergency location immediate service is a service used especially for querying the location of a mobile client that has initiated an emergency call. The response to this service is required immediately (within a set time). The standard location reporting service is a service that is used when a mobile client wants an application server to receive its location. The position is sent to the application server from the location server. The concerned application and its address are defined in the location server or specified by the mobile client. The emergency location reporting service is a service that is used when the wireless network automatically initiates the positioning at an emergency call. The position and related data are then sent to the emergency application from the location server.

The triggered location reporting service is a service used when the mobile client’s location should be reported at a specific time interval or on the occurrence of a specific event.

4 The global architecture proposed

The approach used to design the GLWSA is to analyse the web services discovered in mobility context. For these purposes, all entities that interact with the system must be found. We suppose that UDDI is a base registry of web services and extend it for client mobility. We consider MLP as the interoperability location protocol and extend it to add factorised common location methods. In this section, the proposed system GLWSA (Figure 1) is described before the GLWSA topology. Then, the impact of mobility will be illustrated. Finally, the UDDIM (UDDI for Mobiles) registry and the MLPe (Mobile Location Protocol extension) will be described.

4.1 Investigated system GLWSA

The GLWSA, illustrated in Figure 1, is composed of three main entities: the server Geo-Located Web Services Manager (GLWSM), the UDDIM and ClientInfosDB (Client Personal Information DataBase) databases.

The UDDIM database is an extension of UDDI to store the GLWSA topology. This extension is related to the geographical position of GLWSM nodes, the service agreement to a specific GLWSM node and the desired QoS limit values of a specific service. The database ClientInfosDB stores user personal data such as identification (name, address, username, password, etc.), equipment (mobile station identifier, phone number), subscription and quality of service data.
The GLWSM interacts with external entities such as: mobile clients, LCS, and SAS. SOAP (Simple Object Access Protocol) is used between: Client and GLWSM, GLWSM GLWSM, GLWSM and SAS; MLPe protocol is used between GLWSM and LCS. To connect GLWSM to the databases UDDIM and ClientInfosDB, a data connector such as ODBC (Open Data Base Connectivity) is used. The messages exchanged are carried out through the mobile network and the internet.

4.2 GLWSA topology

As shown in Figure 2, the GLWSA topology proposed is a bi-level hierarchical model. A bi-level tree was chosen to reduce the number of exchanged messages among the GLWSM nodes when a publication service is initiated and to limit the lookup service to a single node. The lower level corresponds to the level of the tree leaves and the higher level represents the tree root. The leaf level includes or contains several GLWSM nodes. A leaf node contains a set of published services from the suppliers. The service publication is coordinated by the root node, which stores all GLWSM nodes of the GLWSA topology.

Each mobile access network is associated with one GLWSM node. A GLWSM covers a region that represents the location areas of the mobile access network with which it is associated. Two different GLWSM nodes cannot cover the same location areas. A SAS can be associated with many GLWSMs. A supplier’s particular web service can be distributed over many SAS. The main characteristic of the leaf nodes is that they are dependent on the location areas of the network operator to ensure the mobility tracking, to maintain service execution in a SAS, and to coordinate the service migration when a mobile client moves to location areas controlled by another GLWSM node where the service in execution exists. Otherwise, the service execution will continue to be provided by the SAS in progress. We impose a migration delay constraint of two seconds to migrate a service.
All leaf nodes offer functionalities for authentication and authorisation, subscription, tracking mobile position, coordination of the migration, lookup and publication service. These are the relation properties of GWLSA:

- **Covered areas**
  In the topology suggested, the location areas covered by two different GLWSM leaf nodes are disjoined.

- **Visibility relation**
  Every GLWSM node that offers a service $S_i$ knows all GLWSM nodes, which offer the same service.

- **Home node**
  It is the GLWSM leaf node where the mobile client is registered. Every mobile client is associated with a single home node. The mobile client used the home node to authenticate himself and to lookup for a service.

- **Visited node**
  It is a GLWSM leaf node other than the home node of a mobile client.

- **Nearest node**
  It is a GLWSM leaf node where the mobile client is located.

- **Nearest SAS**
  It is the SAS attached to the nearest GLWSM node and where the service requested by a mobile client is in execution phase.
4.3 Impact of the mobility

Two mobility scenarios are possible when a mobile client interacts with a GLWSM node: micromobility and macromobility (Figure 3). We suppose that in the mobile network, an IP micromobility protocol (such as HMIPv6, Cellular IPv6 or Hawaii IPv6) is used to know the location address of a mobile client in the network layer.

Figure 3 Impact of the mobility in GLWSA in case of HMIPv6

In case of micromobility with HMIPv6 for example, let us suppose that a mobile client who is attached to access router ARi1 sends a lookup message to GLWSMh and changes point of attachment to ARi2 before receiving the result. During this local handoff, the mobile client will receive two care-of-addresses: new local care-of-address LCoA; and a regional care-of-address RCoA;. He will send a binding update containing RCoA; and LCoA; to a gateway router GRi acting as MAP (Mobile Anchor Point). The GRi extracts the LCoA; and RCoA;, then update its routing table to forward all received packets sent by GLWSMh with RCoA; to LCoA;'. Thus, the mobile client will receive the server SASi address and will interact with it.

If the handoff occurs in macromobility for GRi to GRj when the lookup message is sent to GLWSMh, the mobile client will receive a new local care-of-address LCoA;i; and a new regional care-of-address RCoA;i. He will send three binding updates: the first one to GRj acting as MAP that contains the LCoA;i; and RCoA;i; the second to the home agent that contains the RCoA;i and the third to the correspondent GLWSMh that contains the RCoA;i too. All packets addressed to the mobile client will be routed to its new RCoA;i. Thus, the GRj will forward them to the new LCoA;i.
4.4 UDDIM registry: UDDI extension for mobiles

The current UDDI version does not define data structures and API to find and publish a geo-located web service. In our work, we extend the UDDI protocol to add the GLWSA topology and the agreement service in order to find and publish a geo-located web service. We call this UDDI extension a UDDIM (UDDI for Mobiles). We select the UDDI as the basis protocol to extend because it is considered as the standard in the web service discovering process.

Practically, we add four XML data structures to the UDDI: Node, Agreement, AgreementNode and PolygonLineSegment. The UML representation of these elements is given in Figure 4.

The data structure of a Node element describes information about a GLWSM node. The parameters that identify a GLWSM node are node key, the node URL, the type of the node (‘root’ or ‘leaf’), the geographical position of the node (xPos, yPos, zPos), the maximum distance distanceMaxNodeService allowed between an SAS and the node nodeId, the MAP address of the MAP associated with the node and the number of line segments numberOfPolygonLineSegment that delimit the node covered area (we consider that a node covered area has a polygon shape).

Figure 4 UML class diagram of UDDIM data structures

```
Agreement
- agreementId: String
- serviceId: String
- adminId: String
- systCoord: String

AgreementNode
- agreementId: String
- nodeId: String
- urlApp: String
- costService: double
- BW_min: double
- BW_max: double
- PPD_min: double
- PPD_max: double
- SDU_smax: int
  - xPos: double
  - yPos: double
  - zPos: double
  - distanceMaxNodeService: double
  - Map_Addr: String
  - numberOfPolygonLineSegment: int

PolygonLineSegment
- lineSegmentId: String
  - X1: double
  - Y1: double
  - X2: double
  - Y2: double
  - sign: String

Node
- nodeId: String
  - type: String
  - xPos: double
  - yPos: double
  - Map_Addr: String

```

The Agreement element represents service agreement of a particular service. It is composed of the agreement key, service key, service administrator identifier and coordinate system reference. The AgreementNode element represents the detailed description of an agreement service in a particular GLWSM node. It is composed of the static QoS parameters (costService, BW_min, BW_max, PPD_min, PPD_max, SDU_smax). The BW_min and BW_max represent the minimum and maximum bandwidth a service serviceId can offer at the node nodeId. The PPD_min and PPD_max represent the minimum and maximum propagation path delay a service serviceId can offer at the node nodeId. The SDU_smax is the maximum size of the service data unit.
We chose the provided QoS parameters in conformity with the 3GPP recommendation of interactive applications (Ericsson, 2001). The parameters (xPos, yPos, and zPos) of an AgreementNode element materialise the geographical position of a SAS that offers the service `serviceId` at the node `nodeId`. The PolygonLineSegment allows to determine one line segment of a node covered area which has a polygon shape. The \((X1,Y1)\) and \((X2,Y2)\) parameters represent the \(X\) and \(Y\) cartesian coordinate values of two points ‘1’ and ‘2’ belonging to one line segment, respectively. The attribute `sign` (can take one of the values ‘<=’ or ‘>=’) of a PolygonLineSegment is one of the line segment inequations of a node.

The UDDIM API are methods that manage the data structures that we described above. The following API have been added to the UDDIM:

- `addNode(parameters are all parameters of element ‘Node’)`: adds a node in the topology; `removeNode(nodeId: String)`: removes a node; `findNode(nodeId: String)`: finds a specific node; `getRootNode(nodeId: String)`: finds and returns the root node of GLWSA topology. The described API of this paragraph and those presented below are implemented in the topology manager object `TopologyManager` and the agreement manager `AgreementManager` described in Section 5, respectively.

- `createAgreement(parameters are all parameters of element ‘Agreement’)`: creates a new Agreement for a specific service; `removeAgreement(agreementId: String)`: removes a specific agreement; `findAgreement(agreementId: String)`: finds a specific agreement

- `createAgreementNode(parameters are all parameters of element ‘AgreementNode’)`: creates a new agreement for a specific service in a particular node; `removeAgreementNode(agreementId: String, nodeKey: String)`: removes a specific agreement for particular node; `findAgreementNode(agreementId: String, nodeId: String)`: finds a specific agreement for a particular node

- `createPolygonLineSegment(parameters are all parameters of element ‘PolygonLineSegment’)`: creates a new line segment for a specific node; `removePolygonLineSegment(lineSegmentId: String)`: removes a specific line segment; `findPolygonLineSegment(lineSegmentId: String)`: finds a specific line segment.

4.5 MLPe: MLP extension

The usability of mobile client location to offer services is a common characteristic of geo-located applications. With the current MLP version 3.0, when an application requests the positions of a group of mobiles, it sends a list of mobile identifiers to the LCS (Figure 5). We analysed these applications and found that in the location request of a group of mobiles, the mobiles are generally linked by a thematic subject. Thus, we factorised the common location methods related to a group of mobiles on the basis of a thematic subject to interact with the LCS. Thematic factorisation of the common location methods appeared important to us, as it allows code re-utilisation and decreases the message size of the location request of the group of mobiles loaded on the network. Indeed, thematic factorisation would allow requesting the positions of a group of mobiles bound by a subject and thus avoid loading on the network a list of the mobiles which would increase the transmission time of the requests to the GLWSM. For these purposes, the current MLP was extended.
Figure 5  Group of MSIDS construction in current MLP version

<msids>
  <msid>461018765710</msid>
  <msid>441018905710</msid>
  <msid>401318905543</msid>
  <msid>341617905548</msid>
  <msid>286018436597</msid>
  <msid>494455789532</msid>
</msids>

The data structures of elements added to the MLP protocol are represented in the UML diagram (Figure 6).

Figure 6  UML class diagram of MLPe data structures

The MLPe was also provided with the API to manage elements Theme and ThemeMsid: addTheme(parameters are all parameters of element Theme) adds a theme to an LCS, removeTheme(themeId: String) removes a specific theme; findTheme(themeId: String) finds a specific theme; addThemeMsid (themeId: String, msId: String) relates or adds a specific mobile identifier msId to the particular theme themeId; findAllMsidOfTheme(themeId: String) finds all mobile client identifiers related to the theme themeId; removeThemeMsid (themeId: String, msid: String ) removes a specific mobile identifier msId from the particular theme themeId.

In order to factorise the common geo-located services, we established a typology of the geo-located applications documented in the literature. We distinguish among four categories of applications: informational services, mobile client tracking, resource management services and navigation services (Dru and Saada, 2001).

These are the factorised location methods:

- locateSubject(subject): allows to locate the position of all the mobiles bound on the queried subject
- locateSubjectNearOf(subject, position, precision): allows to locate the position of all the mobiles of the queried subject whose distance compared to the value ‘position’ is less than or equal to ‘precision’
- locateSubjectInZone(subject, idzone): allows locating the position of all the mobiles of the subject queried which are in the area ‘idzone’
A Geo-Located Web Services Architecture

- \textit{ifEntry} (subject, idzone): allows to find the mobiles of the subject queried entering the location area ‘idzone’ and to notify the entry to the SAS server; \textit{notifyEntry} (msid): notifies the SAS that the mobile identifier is entering into the covered location areas.

- \textit{ifExit} (subject, idzone): allows to find the mobiles of the subject queried leaving the location area ‘idzone’ and to notify the exit to the SAS server; \textit{notifyExit} (msid): notifies the SAS that the mobile identifier ‘msid’ is leaving.

- \textit{collocate} (msid, subject): allows finding the mobiles of the queried subject which are in the same location area as the mobile ‘msid’.

To implement the MLPe protocol, we modified the MPS 6.0 tool of Ericsson that simulates mobile location. This tool is originally implemented using Java language. We created a new package ‘com.ericsson.mps.jml.mlp.Theme’ where we added the data structure of Theme, ThemeMsid and PolygonLineSegment. For simulation purposes, we mapped this structure to a database ClientInfosDB by creating associated tables. An extract of the detailed implementation of thematic methods added is described in Figure 7. Due to the basic function \textit{getLocation} (of \textit{LocationProtocol} class) called to locate a mobile in the LCS server, we were constraint to add to every thematic location method (except \textit{notifyEntry} and \textit{notifyExit}), a new parameter that represents a request context \textit{RequestContext} object in order to initialise the location parameters in MPS tool. We modified the MPS ‘com.ericsson.mps.jml.DefaultConnection’ class to implement all thematic location methods and APIs that manage data structure Theme and ThemeMsid.

We also added in this class a private method \textit{getSpatialData} that converts longitude/latitude/altitude coordinates to Cartesian coordinates and a private method \textit{isInsideCoveredArea} which verifies if a mobile is inside a node covered area.

\textbf{Figure 7} Extract of thematic location methods

```java
public synchronized void ifEntry(String subject, ArrayList zone, String urlApp, LocationRequestContext requestContext)
{
    LocationResult result=null;
    ArrayList mobileFound=null;
    boolean flag=false;
    try{
        result=locateSubject(subject, requestContext);
        ArrayList locations=(ArrayList) result.getLocations();
        for (int i=0; i<locations.size(); i++)
        if((((LocationData)locations.get(i)).getLocationStatus() == LocationData.SUCCESSFUL_LOCATION)){
            LocatedMobileStation lms =((LocatedMobileStation)locations.get(i));
            Spatial spatial = getSpatialData(lms);
            flag=isInsideCoveredArea(zone, spatial);
            if(flag=true){
                notifyEntry(((LocationData)locations.get(i)).getMobileStationId(),urlApp);
            }
```
public synchronized void notifyEntry(String msid, String urlApp) {
    try {
        Service service = new Service();
        Call call = (Call) service.createCall();
        call.setTargetEndpointAddress( new java.net.URL(urlApp) );
        call.setOperationName(new QName("http://soapinterop.org/",
                "pushAnnounceEntry"));
        String rep= (String) call.invoke( new Object[] {msid} );
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}

public synchronized LocationResult locateSubject(String subject, LocationRequestContext requestContext) {
    Theme theme=null;
    ThemeMsid themeMsid =null;
    com.ericsson.snf.mps.Connection conn=null;
    LocationResult result=null;
    CommunicationException communicationexception = null;
    LocationProtocol locationprotocol = null;
    java.net.URL urlList[]= factory.getURLs();
    java.net.URL urlLCS=null;
    try{
        themeMsid = new ThemeMsid();
        theme = new Theme();
        ArrayList msidList=(ArrayList)themeMsid.findAllMsid(theme.findThemeBySubject(subject).getThemeId());
        if (msidList.size()!=0){
            LocationTarget[] target=new LocationTarget[msidList.size()];
            for (int i=0; i<msidList.size(); i++){
                target[i]=new MobileStation(msidList.get(i).toString());
            }
            locationprotocol = new LocationProtocol();
            locationprotocol.setLocationTarget(target);
        }
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}

A Geo-Located Web Services Architecture

Figure 7  Extract of thematic location methods (continued)

```java
if (urlList[0] !=null) urlLCS=urlList[0];
if (urlList[1] !=null) urlLCS=urlList[1];
HttpURLConnection httpurlconnection =
    locationprotocol.openConnection(urlLCS);
result = locationprotocol.getLocation(httpurlconnection, myUser,
    myPwd, target, requestContext);
httpurlconnection.disconnect();
}
}catch(Exception e) {
e.printStackTrace();
} finally{
    try{
        conn.close();
    }catch(LocationException lex){
        System.out.println(lex.getMessage());
    }
}
return result;

public synchronized ArrayList locateSubjectInZone(String subject,
    ArrayList zone, LocationRequestContext requestContext){
    LocationResult  result=null;
    ArrayList mobileFound=null;
    boolean flag=false;
    try{
        result=locateSubject(subject,requestContext);
        ArrayList locations=(ArrayList) result.getLocations();
        for (int i=0; i<locations.size(); i++){
            if(((LocationData)locations.get(i)).getLocationStatus() ==
                LocationData.SUCCESSFUL_LOCATION)
                LocatedMobileStation lms =
                    (LocatedMobileStation)locations.get(i);
                Spatial spatial = getSpatialData(lms);
                flag=isInsideCoveredArea(zone, spatial);
            if(flag==true){
                mobileFound.add(((LocationData)locations.get(i)).getMobileStationId());
            }
        }
    }catch(Exception e) {
        e.printStackTrace();
    } finally{
        try{
            conn.close();
        }catch(LocationException lex){
            System.out.println(lex.getMessage());
        }
    }
    return mobileFound;
}
```
private boolean isInsideCoveredArea(ArrayList lineSegments, Spatial spatial) {
    boolean flag = false;
    int j = 0;
    try {
        if (lineSegments.size() >= 3) flag = true;
        while (j < lineSegments.size() && flag == true) {
            PolygonLineSegment lineSegment = (PolygonLineSegment) lineSegments.get(i);
            double xCoeff = (lineSegment.getY2() - lineSegment.getY1()) / (lineSegment.getX2() - lineSegment.getX1());
            double leftValue = spatial.getY() - xCoeff * spatial.getX();
            double rightValue = lineSegment.getY1() - xCoeff * lineSegment.getX1();
            if (lineSegment.getSign().equals("<=") && (leftValue <= rightValue)) j = j + 1;
            else {
                flag = false;
                break;
            }
            if (lineSegment.getSign().equals(">=") && (leftValue >= rightValue)) j = j + 1;
            else {
                flag = false;
                break;
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return flag;
}
In Figure 7, parameters myUser and myPassword are attributes of the DefaultConnection class. The LocationData object contains the position result status (success or failure) of a specific mobile. The zone is the node covered area (specifically, it is a collection of PolygonLineSegment elements). The detailed information of a mobile position is contained in the LocatedMobileStation object. The parameter urlList contains two URL addresses listened by the Ericsson MPS Emulator. The method pushAnnounceEntry called by notifyEntry just adds a found mobile identifier in the list of found mobiles in the SAS.

The thematic location generates a security problem to know if a location requester that is authorised to use a thematic location has the right to obtain the position of a particular client. Then, for security reasons, the authorisation process to get a mobile location verifies first if a particular user has the right to use the requested thematic location and if he has the right to see the position of a client before getting a mobile client location. We propose a scheme <authorised subject to track the members of a thematic location> <authorised operations> <target object> <object conditions> to perform this kind of access control. For example, <authorised user to track the members of the subject ABC Taxi> <collocate> <the mobile client identifier msid> <if the mobile client identifier msid is a member of the subject ABC Taxi>.

5 The functional architecture

To determine the functional architecture, we used a top-down approach based on the use cases analysis of the different types of applications that can be deployed in a GLWSA to discover objects and API functions. The functional architecture of a GLWSA is illustrated in Figure 8. In this section, we describe the manager objects where API functions are implemented.

The functional architecture has three main layers: published services and asynchronous message, service managers and basic classes.

5.1 Published services and asynchronous message listeners layer

The published services and asynchronous message listeners expose some synchronous methods of the service managers layer that will be accessible to an external client and listen for asynchronous messages sent by another entity. The terms synchronous and asynchronous mean that in a service execution, a client will wait or will not wait for the server response, respectively. A published service is a web service. To publish services (Table 1), we used Axis Apache container that parses developed classes to SOAP (and SOAP to classes), and deploys the methods selected in a service manager to a web service.
Table 1  Published services or web services sub-layer description

<table>
<thead>
<tr>
<th>Web service or published service</th>
<th>The associated service manager class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DvtService</td>
<td>DvtServiceManager</td>
<td>Offers APIs that allow to lookup a nearest geo-located web service with or without QoS and to publish a geo-located web service.</td>
</tr>
<tr>
<td>AgreementService</td>
<td>AgreementManager</td>
<td>Offers APIs that allow to lookup, create and delete an agreement of service and an agreement in a particular node.</td>
</tr>
<tr>
<td>SubscriptionService</td>
<td>SubscriptionManager</td>
<td>Offers APIs that allow lookup, create and delete a subscription, a user and a mobile equipment profile.</td>
</tr>
<tr>
<td>LocationService</td>
<td>LocationManager</td>
<td>Offers API that allows to lookup a mobile client position.</td>
</tr>
<tr>
<td>TopologyService</td>
<td>TopologyManager</td>
<td>Offers APIs that allow to lookup, create and delete a node.</td>
</tr>
</tbody>
</table>

The asynchronous message listeners consist of listening to the message sent to the broker destination addresses of a specific GLWSM node. To implement asynchronous messages, we used the Sun Message Queue 3.5 toolkit. Each message has a producer who writes and sends the message and one or many subscribers that consume the message.
Table 2  Asynchronous messages listeners sub-layer

<table>
<thead>
<tr>
<th>Message sent</th>
<th>Producer</th>
<th>Subscriber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile tracking (is implemented in the migration manager).</td>
<td>The current GLWSM leaf node where the mobile is located or the home node of a mobile client.</td>
<td>The next leaf node where a mobile has just entered.</td>
</tr>
<tr>
<td>Service publication (is implemented in the discovery service manager).</td>
<td>The GLWSM root node.</td>
<td>All GLWSM leaf nodes where the service will be published.</td>
</tr>
<tr>
<td>Collected QoS propagation (is implemented in the network QoS manager).</td>
<td>Each GLWSM leaf node that offers the service concerned by the QoS collection.</td>
<td>All other GLWSM leaf nodes where the concerned service has been published (except the producer).</td>
</tr>
</tbody>
</table>

5.2 The service managers layer


1 Authentication and authorisation manager

Authentication and authorisation operations are conducted in GLWSM home. Mobile clients first connect with GLWSM home giving their credentials (username, password). After successful authentication, access control takes place. The access control operation is based on the user’s role and capabilities. There are three types of roles: administrator, supplier and client.

2 Subscription manager

Suppliers use a subscription manager to register new clients who want to use one of their applications. This operation will assign a GLWSM home to a client according to his home address. Personal data such as name, username, password, address, subscribed services, mobile equipment identifier will be recorded in the ClientInfosDB database attached to the GLWSM home node.

3 Discovery services manager

Nearest service lookup

When an authorised mobile client looks up for a geo-located web service (Figure 7), the following steps are executed:

a The mobile client sends first a nearest service lookup request containing the service name and the mobile identifier to his home GLWSM node.

b The home GLWSM node sends a client position request to the LCS. The LCS returns the client’s geographical position to the GLWSM home.

c The home GLWSM node finds and selects in his UDDIM database the GLWSM leaf node that covers the current location area of the mobile client (the covered areas property described in Section 4.2 is used in this step). If the mobile position does not
belong in the covered areas of any of the GLWSM leaf nodes, then the requested service is not offered in the location area where the mobile client currently resides and no SAS will be selected.

d If a GLWSM leaf node is found, it is considered to be the nearest GLWSM node.

e Then, the home GLWSM node finds and selects the requested service in his UDDIM (the visibility relation described in Section 4.2 is used in this step).

f The URL of the requested service associated with the nearest GLWSM will be returned to the mobile client.

g The SAS that offers the returned service will be considered to be the nearest SAS offering this service.

Figure 9  Nearest service lookup with or without QoS
Nearest service lookup with QoS:

The detailed description of this feature consists of finding a service with the associated network QoS. This aspect is beyond the scope of this paper.

Service publication:

After a successful authentication of an SAS administrator to the GLWSM root, the administrator sends the service entity to be published and its agreement identifier to the root node (Figure 10). The system verifies the agreement conformity and saves the service in the UDDIM root. This service is forwarded to all GLWSM leaf nodes that are included in the agreement service. The forwarding message is done by sending an asynchronous message destined to all nodes that subscribed to this service publication topic (all GLWSM leaf nodes included in the agreement service contract except the producer of the message). Each concerned leaf node receives the message, stores the data and sends back a receipt to the GLWSM root. The GLWSM root keeps a list of received messages and compares it to the list of selected leaf nodes to confirm whether publication was well done in all nodes.

Figure 10  Service publication
4 Migration manager

The migration manager of a current nearest GLWSM node tracks the position of any mobile client in communication with a current SAS attached to it. It also coordinates the service migration by informing the current nearest SAS in execution of the next SAS address. At the beginning of the process, a mobile client MC is attached to a current nearest SAS in execution. This SAS is considered as the current SAS where the requested service is in execution. The detailed description of the coordination of the service migration with QoS is beyond the scope of this paper.

The steps below describe the coordination of a migration process without QoS (Figure 11).

a. The current nearest GLWSM where the mobile MC is attached sends to an LCS a location request that consists of tracking if the mobile MC is presently in its covered location areas.

b. The geographical position obtained is processed by the current nearest GLWSM to find the next nearest GLWSM node that covers the current location area of the mobile client.

c. Then, the current nearest GLWSM verifies if the next nearest GLWSM found is a member of the service agreement nodes. In other terms, the current nearest GLWSM verifies if the requested service is offered in the next nearest GLWSM.

d. If not, the service does not exist in the location area where the mobile is moving. Otherwise, the next nearest GLWSM is either attached to the same current nearest SAS or a next nearest SAS.

e. If the next nearest GLWSM is attached to the current nearest SAS, the service execution will continue to this current nearest SAS and the next nearest GLWSM will be notified by the current nearest GLWSM to track the position of the mobile client who has just entered into his covered location areas.

f. If the next nearest GLWSM is attached to a next nearest SAS that offers this service, the current nearest GLWSM informs the current nearest SAS in execution progress of the URL address of the next nearest SAS where the service must be migrated so that the service remains closer to the client’s current position. Once the service has migrated, the current nearest GLWSM associated with the current nearest SAS closes the client session and instructs the next nearest GLWSM to continue tracking the target client position.

The maintainability of services in execution provides the continuity of a service and eventually reduces the communication costs. Indeed, when a client moves over a mobile network topology, the number of used hops between him and the SAS taking part in the communication will generally increase if the client moves farther away. Consequently, the total delay of exchanged messages between these two entities may considerably increase due principally to the queueing delay in the hops. The service migration will reduce this latency by selecting the nearest SAS that offers the service in the mobile client location context and decrease communication costs in cases where time billing model is used.²
The process migration between two SAS is not in the scope of the GLWSA. Meanwhile, the SAS process migration can be done using thread migration without transferring the application code (in fact, each SAS has a persistent application code) or using session migration. An interesting work in thread migration realised by Bouchenak et al. (2002), is conducted to build a thread migration tool that does not generate the message overhead. This tool is certified and protected by Sun Microsystems. The performance of thread or session migration depends on the size of frames and the size of serialised objects to migrate. The session or thread median migration latency is in the order of three milliseconds to few hundreds of milliseconds.

5 Topology manager

The topology manager object maintains the topology of the GLWSA. It adds, removes and modifies GLWSM node information. Those operations can only be performed by a GLWSM administrator.

6 Location manager

The location manager interacts with the LCS to obtain the position of mobile clients. It contains location API and thematic location API to get the position of a mobile or a group of mobile clients.
7 Agreement manager

The agreement manager defines the agreement contract of a geo-located web service. It binds a geo-located web service of a SAS to a list of GLWSM nodes, defines the service cost at each GLWSM node and the minimum and maximum QoS requirements that the supplier wishes to be controlled.

8 Network QoS manager

The network QoS will be used to collect the QoS parameters (network path bandwidth, SAS utilisation rate) in a particular domain controlled by a GLWSM node. Thus, when the mobile client sends a lookup message with QoS, the GLWSM home will verify if the desired QoS can be obtained. Then, the found service URL will be returned to the mobile client. The proposed network manager does not provide the negotiation or the renegotiation QoS mechanisms. The detailed operations of the network QoS manager fall beyond the scope of this paper.

6 Implementation details and evaluation

In order to evaluate the proposed GLWSA architecture, we built a prototype using the Java programming language (Jbuilder 7 and Sun Message Queue 3.5). This prototype implements the functionalities of the GLWSM and communicates with the LCS and the SAS servers. Communication with the LCS was carried out through Ericsson MPS 6.0 emulator. The emulator uses an MLP V3.0 protocol to determine the position of a client (or a group of clients) moving over the network topology. We generate a network topology with the network density set to urban, the distance between two base stations is set to 5000 metres and created a mobile trajectory with a constant speed value (Figure 12). In Figure 12, the GLWSMh is the home GLWSM domain of the target mobile client and its geographical covered areas, the GLWSM1 and GLWSM2 are the visited GLWSM domains of the target mobile client with their associated geographical covered areas. In our tests, we used three constant speeds: 50 km/h, 100 km/h and 200 km/h. By using the configuration settings described above, a static route file that contains the current cell identifier where the mobile resides, the relative distance between the mobile and the current base station and the mobile position data calculated every ten seconds and given in geographical system coordinate (latitude/longitude/altitude) is created. The tool calculates the client position between two consecutive offset times of the route file (for example between zero and ten seconds) by using the interpolation operations. But the formula used to do the interpolation operations is not given by the MPS tool specifications. At the beginning of the simulation, the emulator starts a clock and reads the mobile position in function clock time in the static route file created. We used the database management system Oracle 9i to store data in UDDIM registry and ClientInfosDB. We used JUDDI3 and appended UDDIM API to interact with the registry UDDIM. The machines used to materialise the GLWSM, the SAS and the LCS are similar (1.2 GHz Pentium III with 512 Mo RAM). The machines are connected in a wireless LAN. The WLAN has a transmission rate of 11 Mbits/s and is compliant IEEE 802.11b. The relevant metrics selected to evaluate the performances of the GLWSM are: the nearest service lookup time, the lookup mobile time, the migration service time and the publication service time.
The Round Trip Time (RTT) is the time difference between the reception time of the request response at a client machine and the time when the client sent a request to a server. In Figure 13, we measured the RTT of a nearest service lookup when a mobile client sends a request every two seconds to his home GLWSM. We found a minimal RTT value of 20 milliseconds, an average RTT of 28 milliseconds and a maximum RTT of 52 milliseconds. Then, we compared in Figure 14, the average RTT obtained with the lookup service time (RTT) in Jini and SLP (Govia and Barbeau, 2001) to which we added the average calculation time of the position to the LCS which is 23 milliseconds; a total lookup service time of 46 and 51 milliseconds are obtained for SLP and Jini, respectively. Note that the substantial gain of the GLWSM over Jini and SLP in the nearest service lookup of geo-located web service is related to the fact that the two traditional protocols Jini and SLP do not integrate the UDDI structure in their architecture and thus require additional data transformation operations to be compatible with a UDDI data structure.
In Figure 15, we measured the RTT of the nearest service lookup when the number of clients varies from 1 to 50. We found an average RTT value of 52 milliseconds and a maximal RTT of 620 milliseconds. We also found that the RTT increases considerably when the number of clients is greater than or equal to 28. We think that the increasing of RTT and the observed peak values are due to the GLWSM buffer overflow.

The coordination time of a service migration shown in Figure 16 represents the RTT to send the URL of the next SAS (that implement the service in execution of a mobile client) and the mobile identifier parameter of a mobile client (who just changed the SAS domain) to the current SAS in execution. At the receiving of the message, the current SAS just sends back an acknowledgement to the GLWSM sender. Then, the GLWSM sender notifies the next GLWSM to track the target mobile. The coordination time of a service migration has an average RTT of 11 milliseconds, a minimal value of seven milliseconds and a peak value of 40 milliseconds. We varied the speed of the target mobile and we remarked that the speed does not have a direct impact in the coordination of the service migration. Meanwhile, as we imposed that the delay migration constraint must be less than or equal to two seconds, if a mobile client has a speed of 300 Km/h when the migration is relevated, the target mobile will be at 166.67 metres of the precedent GLWSM domain when the service migration will be terminated. As the service migration for SAS to SAS has a maximum average rate of 300 milliseconds (Bouchenak et al., 2002), we will have a maximum total service migration time (SAS service migration time and coordination migration time) of 355 milliseconds, which is less than the delay migration service constraint. Thus, the system has a margin time of 1645 milliseconds for huge applications.

Figure 17 shows a geo-located web service publication time in two GLWSM leaf nodes. To measure the service publication, we suppose that an authorised supplier sent a size of 1116 bytes parameter data (service, agreement service, agreement node entities) to publish to the root GLWSM machine. Then, the root GLWSM stores the data in his UDDIM and forwards them to the publication queue listened by the two GLWSM leaf
nodes. After storing the forwarded data in their UDDIM, each GLWSM leaf node sends back a receipt to the root GLWSM. We found an average RTT of 105 milliseconds with a minimal and maximal RTT of 88 and 300 milliseconds during a ten-minute observation. We also analysed the publication time in function of the number GLWSM leaf nodes (which increases the size of parameters sent in a publication process) where they are published. We found that the publication time increases when the number of GLWSM leaf nodes increases, too. We found an average RTT of 110, 147 and 247 milliseconds for five (Figure 18), ten (Figure 19) and 20 GLWSM leaf nodes (Figure 20), respectively. This variation is due to the fact that the size of messages sent to a root GLWSM increases when the number of selected GLWSM leaf nodes increases, too.

The sending location time of a group of mobiles in a LCS was measured to evaluate the relevance of thematic factorisation of the common location functionalities. Tests were designed to compare a sending location request of a group or list of mobiles versus a thematic location. For this purpose, we built two methods: `locateSubject` that has one parameter representing a subject or a theme and `locateMsids` that takes as parameter the list of mobile identifiers that we want to locate. We calculated the sending time by building a SOAP client request that loads these two methods. The sending time of `locateSubject` is calculated by adding the time difference between the received time and sent time of the request and the lookup time to retrieve all mobile identifiers in the database ClientInfosDB on the server side. The sending time of `locateMsids` is calculated by adding the reading time of all mobile identifiers in the database ClientInfosDB of client machine and the time difference between the received time and sent time of the request. Figure 21 shows the comparison between the two curves with the maximum subject size implemented (250 characters) and the size of mobile identifiers implemented (20 characters). Note that the thematic location is more efficient. We also remarked that the thematic location gain increases when the number of mobiles to locate increases, too.

**Figure 15** Round trip time of a nearest service lookup in the case where the number of clients varies between 1 and 50
Figure 16 Coordination time of a service migration without QoS

Figure 17 Geo-located web service publication time in two GLWSM leaf nodes
Figure 18  Geo-located web service publication time in two GLWSM leaf nodes

![Graph showing RTT (milliseconds) over time for two GLWSM leaf nodes.]

Figure 19  Geo-located web service publication time in ten GLWSM leaf nodes

![Graph showing RTT (milliseconds) over time for ten GLWSM leaf nodes.]

A Geo-Located Web Services Architecture
Figure 20  Geo-located web service publication time in 20 GLWSM leaf nodes

Figure 21  Comparison of the sending location time of a group of mobiles (locateMsids) versus a thematic location (locateSubject)
7 Learned lesson

In this paper, we learned first the functionalities of the basis protocols MLP and UDDI used in the proposed system. The overview of these protocols is described in Section 3 and gave the lecturer the basis notions to understand the GLWSA system. We also provided a description of the macromobility and the micromobility of a mobile client roaming in the mobile access networks to better understand the impact of the mobility in the proposed system.

8 Conclusion

We presented in this paper a discovering architecture for geo-located web services for the next generation mobile networks. This architecture allows mobile clients to discover geo-located web services and to maintain the execution service closest to their location context. We proposed an extension of UDDI registry and MLP protocol to reach this goal. We also emphasised the importance of a thematic factorisation of common location methods in the delay or cost decreasing of the location request. Future work will consider the collection of the QoS (the network bandwidth in a particular GLWSM domain and the SAS factor utilisation) and a cache information to store the collected QoS quotation in order to offer a discovering web service with QoS and measure the architecture performance.

References


Notes