An Open and Dynamical Service Oriented Architecture for Supporting Mobile Services

Elena Sánchez-Nielsen
Dpto E.I.O y Computación
Universidad de La Laguna,
38271 La Laguna, Spain
enielsen@ull.es

Sandra Martín-Ruiz
Escuela Técnica Superior de Ingeniería Informática
Universidad de La Laguna,
38271 La Laguna, Spain

Jorge Rodríguez-Pedrianes
Escuela Técnica Superior de Ingeniería Informática
Universidad de La Laguna,
38271 La Laguna, Spain

ABSTRACT
Mobile services (m-services) play an important role in daily life and work of modern societies. Current service oriented architectures (SOA) are widely used to offer Web Services on wired networks. However, no significant research has been done in m-services field to provide availability service and dynamical discovery to mobile users. Nowadays, mobile devices are characterized by limited resources such as processing power, memory, display screen and connection bandwidth. In this context, we propose a service oriented architecture that addresses the following issues: (1) dynamical integration of new services by providers at anytime, (2) dynamical discovery of available services and (3) the use of open source software to develop the solution. With this approach, users require no prior knowledge of accessible services nor require updating the application of their mobile devices when new services are incorporated. The paper includes results with service scenarios, showing that Web Services can safely be used as paradigm to support m-services.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software architectures – *patterns*, Interoperability – *distributed objects.*

General Terms
Design, Performance, Experimentation.

Keywords
Mobile and ubiquitous computing, M-services architecture, dynamical m-services invocation.

1. MOTIVATION
Mobile technologies are revolutionizing the way people interact with daily life, work and business. Many important functions of modern society are being reshaped to exploit opportunities the mobile devices open such is illustrated by I-mode technology [5]. Examples of mobile services (m-services) include access to information resources (e.g., searching, language translation, newspaper reports, weather forecast,…), telemetry (e.g., receiving traffic updates and logistics tracking), mobile shopping (e.g., booking flights, reserving rental cars, restaurants,…) mobile banking (e.g., billing of services, buying stocks and contacting banks through mobile devices), m-government (online services provided by government agencies to mobile users) and technological assistance for dynamic wayfinding [15] .

Having pre-installed services on user's wireless devices is an option that cannot be considered in an open environment with multiple users with different needs and providers looking for more business opportunities (e.g., including publicity in their basic services). Therefore, online delivery of services from providers to mobile users and dynamical service discovery infrastructure (where provider services can announce their presence and mobile users can locate these services) is more appropriate in dynamic contexts than pre-installing services.

At present, service oriented architectures (SOA), implemented by Web Services technology, are widely used as approaches for supporting services on wired networks (e-services); IRS (Internet Reasoning Service) [16] is a clear example with descriptions of semantic Web services at two different levels. On the other hand, OSGI Service Platform [17] defines a standardized, component oriented, computing environment for networked services. However, no significant research has been done in mobile services (m-services) field using Web services technology and dynamical service discovery infrastructures. As the above examples imply, wireless devices have to be viewed as full participants in networked service-oriented architectures. To facilitate users the discovery of these services is a challenging task judging from the diversity of services and the dynamics of users as well as service providers. First, mobile services are dynamic as service providers may create, update and change them at anytime. Second, how to locate a service in a dynamic and efficient way is also an important issue such is illustrated in the Gravity framework [18].

Currently, the use of the available Web Services technology to discover, access and invoke Web Services directly from mobile devices lacks some important features. First, the using of static stub based communication between a mobile client application and Web Services framework implies a stub appended to the client at compile time for each Web Service to be invoked. This situation requires that all services must be specified at design time or end users download specific applications to their devices when new services are incorporated by service providers at the marketplace. Second, available technology using open source software does not allow users to access directly to UDDI registry [9] from mobile devices. This situation involves that mobile users cannot search new services at runtime without updating devices' applications and all services must be described at mobile devices, increasing the need of large memory resources.

ICWE'06, July 11-14, 2006, Palo Alto, California, USA.
ACM 1-59593-352-2/06/0007.

Against this background, this paper investigates the mechanisms to dynamic support for m-services in a service oriented architecture context in order to provide availability service and dynamic discovery to mobile users at anytime and anywhere. In particular, we first discuss the characteristics of service oriented architectures on wired networks and current service discovery infrastructures. Taking into account the available technology for mobile devices and SOA paradigm for wired networks; we propose our m-services architecture for supporting dynamical and scalable mobile services. In more detail, we propose a service manager entity with a service registry as an intermediate layer between service providers and mobile users. This manager is responsible of coordinating the interactions between services providers and mobile users. These interactions involve how to make service information available and delivery services to mobile users at anytime. We investigate the use of dynamic invocation interface as communication mechanism between services providers and service managers in order to compute service description and invocation at runtime. We propose XML-encoded data exchange between service managers and mobile devices to describe: (i) Web Services descriptions, (ii) operation invocations and (iii) searching at UDDI registry.

With the framework proposed, service providers and new services can easily be added at anytime without updating the application of users' mobile devices when new services are incorporated. As a result, mobile users can search for additional facilities, when needed without prior knowledge of available services, bring these facilities to their wireless devices and carry out this process in a transparent way.

In order to take into account open standards and specifications, the entire solution has been developed using open source software. Different service scenarios have been implemented for evaluating the performance of our m-services architecture. We discuss the results of use of traditional UDDI registry for mobile devices and our approach.

The remainder of the paper is structured in the following way. Section 2 gives an overview of the related work about service oriented architecture paradigm on wired networks and service discovery infrastructures. Section 3 describes our architecture and components to support m-services. Section 4 focuses on the implementation of mobile Web Services using open source software. Section 5 discusses results using different service scenarios and Section 6 provides concluding remarks and future work.

2. RELATED WORK

2.1 Service Oriented Architecture

A service-oriented architecture (SOA) is a contractual architecture to offer and consume software as services. There are three entities that make up SOA [4]: (1) *service providers*, (2) s*ervice requestors* (also known as *service consumers*) and (3) *service registry*. Service providers are the owners that offer services. They define descriptions of their services and publish them in the service registry. Service requestors use a find operation to locate services of interest. The registry returns the description of each relevant service. The requestor uses this description to invoke the corresponding service.

Currently, Web Services technology [4], [11] implements SOA by means of standard XML-based initiatives. Three initiatives are used in order to support interactions among Web Services: SOAP (a way to communicate) [13], WSDL (a way to describe services) [14] and UDDI (a name and directory server) [9]. Figure 1 illustrates the SOA paradigm graphically.

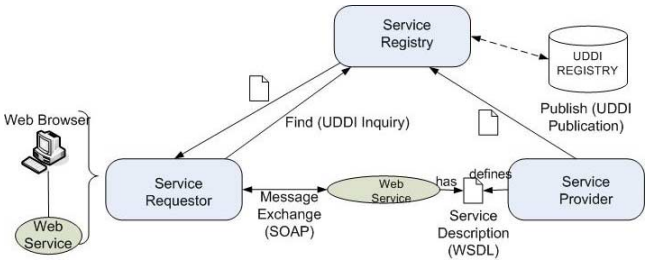

Figure 1: Service oriented architecture

Using SOAP, services can exchange messages by means of standardized conventions to turn a service invocation into an XML message, to exchange the message, and to turn the XML message back into an actual service invocation.

Through WSDL, a designer specifies the programming interface of a Web Service. This interface is specified in terms of methods supported by the Web Service, where each method could take one message as input and return another as output. Four type of messages SOAP are possible: RPC/encoded, RPC/literal, document/literal and document/encoded.

Using Web Services paradigm, a client application (service requestor) makes a procedure call of a Web Service (service provider) in the same way it invokes a local call. There are three types of communication between a client application and a Web Service:

- **Static Stub:** a procedure call of a client application is an invocation of a proxy procedure located in a stub appended to the client at compile time. As a result, a client can invoke methods of a Web Service directly via the stub. The advantage of this model is that it is simple and easy to code. The disadvantage is that the slightest change of Web Service definition leads to the stub being useless and a generation of a new stub. Therefore, the stub based approach is only appropriate in static contexts, where services are not removed and updated by service providers.
- **Dynamic proxy:** in this case, the client application calls a remote procedure through a dynamic proxy that is created at runtime. The dynamic proxy needs to be re-instated whenever the service endpoint interfaces change.
- **Dynamic invocation interface (DII):** this model enables dynamic invocation of Web services without having to know interface details at compile time. With this approach, a client application can query for a service it has never heard of and build on the fly a call to that service. As a result, an application is able to invoke a service that was not known prior to runtime: it can dynamically download the appropriate WSDL file,

parse it, and construct all the elements required to use the service. The use of this approach involves more programming complexity.

2.2 Service Discovery Infrastructures

In a service discovery system, a common service description framework is needed for service providers and service users in order to describe service features so that they can understand each other properly. Generally, each service can be described using a set of attribute-value pairs, where each pair details a property service. There are two ways to organize attributes: (i) a flat structure where all attributes are at the same level and (ii) a hierarchical structure where attributes can be at different levels.

As it is impossible to name and consider all existing service discovery infrastructures in this paper, we have then highlighted those that closely resemble our work and those that are most relevant to the industry:

- **Service Location Protocol (SLP):** is a language-independent protocol for resource discovery on IP networks using an agent-based infrastructure [3]. The base of the SPL lies on predefined service attributes, which can be applied to universally describe software and hardware services. The architecture consists of three types of agents: user agent, service agent and discovery agent. The user agents are responsible for discovering available directory agents, and acquiring service handles on behalf of end-user applications that request services. The service agents are responsible for advertising the service handles to directory agents. Directory agents are responsible for collecting service handles and maintaining the directory of advertised services.
- **Jini:** is a distributed service architecture developed by Sun Microsystems [6], whose services can be realized to represent hardware devices, software programs or their combination using Java language programming. The goal of Jini is to turn the network into an easily tool on which human and computational clients can find services in a flexible way.
- **Universal Plug and Play (UPnP):** this approach extends the original Microsoft Plug and Play peripheral model to support service discovery provided by network devices from different providers [10]. UPnP works and defines standards primarily at the lower-layer network protocol suites. UPnP uses the Simple Service Discovery Protocol (SSDP) for discovery of services over IP networks, which can operate with or without a lookup service in the network.
- **Salutation:** is an open standard and platform-independent service discovery and session management protocol [8]. The architecture provides applications, services and defines a standard method for describing and advertising their capabilities, as well as, locating and determining other services and capabilities.
- **DReggie:** is a project [1] that presents a dynamic service discovery infrastructure based on Jini that uses DAML [2] to describe services and a Prolog reasoning engine to perform matching using the semantic content of service descriptions.
- **UDDI:** is a framework for describing and discovering Web Services [9]. The core of UDDI revolves around the notion of business registry, which is a naming and directory service. UDDI defines data structures and APIs for publishing service descriptions in the registry and for querying the registry to look for published descriptions.
- **METEOR-S:** is a Web Service discovery infrastructure, which presents an ontology (XTRO) based infrastructure to provide access to registries divided, based on business domains and grouped into federations [7].

2.2.1 Drawbacks

In this section, we discuss and analyze discovery infrastructures described in section 2.2. These infrastructures have been designed focusing on two premises:

- **Coordination frameworks:** these frameworks (SLP, Jini, and UPnP) address the issues related to mobile and specialized devices such as ability to announce their presence on the network, automatic discovery of devices in the neighborhood, ability to describe their capabilities as well as query/understand the capabilities of other devices and self configuration without administrative intervention. However, these architectures have some limitations that makes them unsuitable for developing Web Services frameworks to mobile devices: (i) SLP has been designed to locate physical devices or basic services, (ii) UPnP has been designed to accommodate home networks or small office networks and (iii) Jini does not provide any solutions to connect Jini federations which may reside in global networks and this architecture requires service advertisements to be expressed in the form of Java interface descriptions.
- **Web Services discovery based infrastructures:** the use of UDDI registry is only applicable where performance is not a primary concern [4]. In fact, UDDI has been designed neither with the response time capabilities not the facilities necessary to support dynamic binding. Although other Web Services discovery infrastructures add semantic functionalities such as METEOR-S [7], these infrastructures neither solve the problems faced by UDDI infrastructures.

In order to provide a satisfactory solution to support m-services scenarios, a different discovery infrastructure is required. There are two reasons: (i) coordination frameworks work with mobile devices but these frameworks are not targeted towards Web Services solutions and (ii) Web Services discovery based infrastructures are targeted to Web Services based frameworks but they are not focused on dynamic binding and mobile devices solutions.

3. M-SERVICES FRAMEWORK

In the previous sections, service oriented architectures and discovery infrastructures are suggested as potential candidates to the design and development of a framework offering m-services to wireless devices-oriented users. However, current approaches are not appropriate to support m-services, such as discussed in section 2.2.1.

Focusing on preceding premises, our goal is to provide a service oriented framework to mobile users that fits with the available technology using open source tools and the following features: (i) automated access from mobile devices to services of a wired network, without prior knowledge of available services, (ii) no update of mobile users' applications when new services are added, (iii) appropriate response time to mobile users and (iv) the design of a service discovery mechanism that allows service providers to create, update and change services at anytime. Services can be published or not published in UDDI registry.

Traditional service oriented architecture using stub-based model as communication mechanism between client mobile applications and provider services is not appropriate to support m-services. There are two main reasons: (i) every service needs to be coded in the client application assuming a detailed knowledge of each service that will be invoked at runtime and (ii) a stub for each service provided needs to be appended to the client application at compile time in the mobile device, requiring a large use of memory resource. This situation involves that all services must be described (network address, operations to provide, parameters…) at the design-time and no new services can be added after compile time. This context means that service providers cannot create, update and change services at anytime and that mobile users can only access to pre-defined services at their mobile devices.

In order to solve the problems faced by traditional service oriented architecture to mobile users, we propose a framework with an intermediate entity between service providers and service clients. This entity is represented by a service manager that operates as a client of the distributed network of Web Services offered by the different service providers and as server to mobile devices.

3.1 Architecture

Our framework consists of four components, after a new element has been integrated to the service oriented architecture. These components are: (i) service providers, (ii) service managers, (iii) service clients and (iv) UDDI registry. Figure 2 illustrates the architecture of the framework proposed.

3.1.1 Services providers

Service providers are the owners that implement and offer different services. They define descriptions of their services using WSDL specifications [14].

3.1.2 Service clients

Service clients are wireless devices-oriented users interested in standard services and searching of facilities, when it is needed without prior knowledge of available services and bringing these facilities to their wireless devices in a transparent way.

3.1.3 Service managers

Service managers act as a mediator layer between the service providers and mobile devices. They are responsible for information flow between both components. A service manager is a Web Service entity that uses dynamic invocation interface (DII) as communication mechanism between the different service providers. With DII, a service manager can invoke Web Services without knowing their communication interface at compile time. As a result, (i) invocations of Web Services not known prior can be computed by the service manager and (ii) service providers can create, update and change Web Services at runtime.

The framework can support one or multiple service managers. If a single service manager is used, a centralized framework that supports the different service providers is provided. If multiple service managers are used, different operators can be added at anytime, where each operator can support different service providers. In this situation, service clients may choose the service managers, that they are interested.

Client applications that reside in mobile devices only interact with a service manager. This leads to a generation of a single stub class (corresponding to the service manager) and no several stub classes corresponding to each Web Service of the available services.

XML is used as format of exchange data between applications that reside in mobile devices and service managers.

Interactions between mobile devices and service providers using a service manager entity consist of a five-step process:

- **Start up:** When a service manager starts up, it processes a service registry. This registry is a structure that enables service providers to store their list of URL address (URI) of accessible services made available. The service manager maintains a XML based structure as registry. Dynamic invocation interface is used by the service manager in order to obtain the service descriptions at runtime. The use of DII communication mechanism avoids the generation of each stub class required for each available service, because the service manager invokes dynamically the Web Services description and posteriori, invokes them. Therefore, the services list to offer to mobile users can change dynamically, according to new services added at anytime and at runtime.
- **Service discovery:** to discover services in the networked service-oriented architecture, mobile clients send a request for information about services made available by service providers in the service manager entity and/or services in UDDI registry. Mobile clients request "all" services of the service manager or services defined by some "generic" service type. For the discovery of a particular service, keyword is used.
- **Service delivery descriptions:** description of services (operations provided, parameters…) are defined by an XML message, which is sent by the service manager to the client application that resides at mobile devices, after a request of services has been carried out by a mobile user. Client application processes the received information and shows it to the mobile user.
- **Service invocation:** service manager receives a request encoded as an XML message with the necessary information (Web Service name, selected operation, parameter values introduced…) from a mobile device when a user is interested in some service that has been

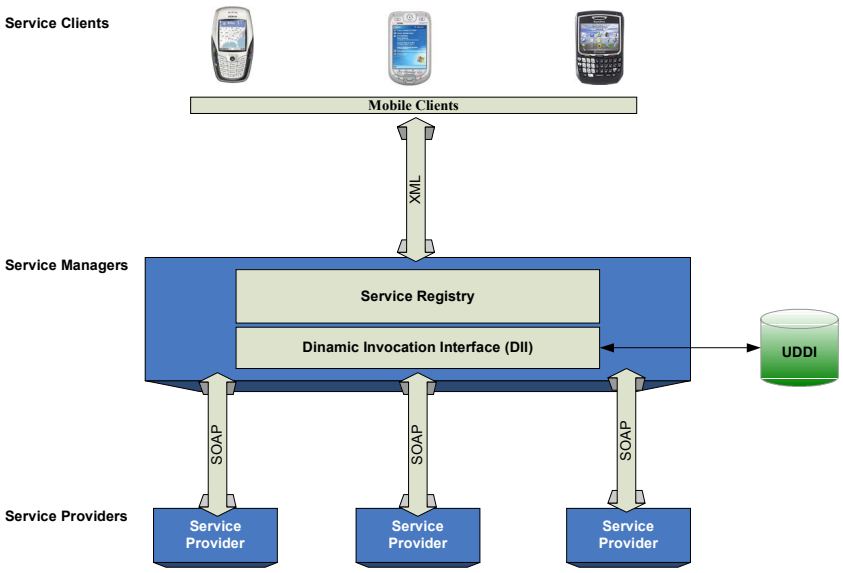

Figure 2. Dynamical m-services architecture

previously delivered to the mobile device in the service delivery description step. Then, the service manager makes the invocation of the Web Service selected using dynamic invocation interface based communication model to the service provider.

- **Results transmission:** the service manager sends the information encoded as an XML message to the mobile user, when it receives the response of the corresponding service provider. This information is shown on the screen display of the mobile device.

3.1.4 UDDI registry

UDDI service directory can also be used by mobile users in order to locate new services. Service discovery is computed at runtime by the service manager, once the user has sent their request of new services in UDDI registry.

3.1.5 XML based infrastructure

A uniform infrastructure using XML-encoded data exchange is used with two purposes: (i) to define the service registry structure and (ii) establish the communication between the mobile device and the service manager.

3.1.5.1 Service Registry

The XML based structure of the service registry enables service providers to store their network address URL of WSDL documents. The format of the XML document is the following:

```
<?xml version="1.0" encoding="windows-1252"?>
<WSDLaddresses>
   <version>1.2</version>
   <wsdl>http://api.google.com/GoogleSearch.wsdl</
   wsdl>
      …
   <wsdl>http://www.webservicex.com/TranslateServi
   ce.asm?WSDL</wsdl>
<WSDLaddresses>
```

In this document, the tag *WSDLaddresses* contains two types of elements: *version* and *wsdl*. *version* is the current version of the Web Services set provided to the service manager. This element is used in order to update new services to mobile devices. *wsdl* is the URL address where is located the WSDL document of a Web Service offered. There are so many *wsdl* elements like Web Services offered.

3.1.5.2 Web Service description

The following format of XML document is sent from the service manager to the mobile device, when the description of Web Services available set and/or location of new services in UDDI are required by the mobile user:

```
<webservices version="1.0">
   <service name="Calculator">
   wsdl="http://81.45.231.68:8080/Server/Calculato
   r.wsdl">
      <porType localPart="Calculator"
      namespaceURI="http://calculator.com">
         <operation name="add">
            <title> Add operation</title>
            <description>To compute add operation
            between two numbers</description>
            <parameters>
               <parameter name="Operator1"
               type="number">
               <parameter name="Operator2"
               type="number">
            </parameters>
            <return>true</return>
         </operation>
         …
         <operation name="subtract">
            …
         </operation>
      </porType>
         …
      <porType>
      …
      </portType>
   </service>
   <service>
```

```
   …
   </service>
</webservices>
```

The root of the document is the *webservices* tag. It represents the start of the set of Web Services. The attribute *version* corresponds to the current version of Web Services descriptions set provided to the mobile device. The use of this attribute allows users check their set of descriptions with the service manager and downloading a new version, when new services have been added. Each Web Service is described with two attributes: *name* and *wsdl*. Different port types can be associated to a specific Web Service through the element *portType*. This element contains three attributes: *localPart*, *namespaceURI* and *operation*. A operation is described by the following elements: *title* (title of the operation), *description* (description of the operation), *parameters* (input parameters of the operation), and *return* (true is returned if the operation returns a value, false in other case)

3.1.5.3 Operation invocation

The structure of the XML document sent from service manager to the mobile device when a Web Service operation is invoked is the following:

```
<results>
   <result>Search time:0.5 seconds</result>
   <result>
      <result>Department Store: XYZ</result>
      <result>
         <result>
            <result>Name: wood table</result>
            <result>Price: 50 euros</result>
         </result>
      …
      </result>
   </result>
</results>
```

The different results are enclosed between *result* tags. If a complex type is returned (such as arrays and structures), the different fields of this type are enclosed into different *result* tags.

3.1.5.4 UDDI search

A service manager sends to the mobile device, the next XML document, when a user requests a search for new Web Services in UDDI registry:

```
<UDDI>
   <webservice
   wsdl="http://81.45.231.68:8080/Server/Calculato
   r.wsdl">
      <description>="To compute add, subtract
      operations…"/>
   …
</UDDI>
```

The different Web Services located are enclosed by a *webservice* tag. *wsdl* and *description* attributes represent the URL address of the WSDL document and the Web Service description.

4. IMPLEMENTATION

The m-services framework described in section 3, allows mobile users to invoke Web Services published at World Wide Web, by requesting a service manager entity at runtime.

In order to test the Web Services framework for mobile devices, we have implemented on mobile phones different scenarios services using a service manager entity.

The following services have been implemented and tested: (1) Searching with Google engine, (2) text translation from one language to another, (3) newspaper reports (this service allows user to select one or several newspapers), (4) converser temperatures, (5) weather forecast, (6) calculator operations and (7) dynamic binding with UDDI registry.

The framework proposed has been implemented using the following open source software: Apache Tomcat 5.0.28 for application server, J2ME Wireless Toolkit (WTK) for programming tool, and Eclipse 3.1 development platform with WTP (Web Tools Platform) plug-in for building software and developing Web applications.

We have developed the Java application and tested it on the Sun emulator. Also, with the purpose of testing correct performance, we have tested the application with mobile emulators of commercial trademarks. Figure 3 shows the application running on the Sun emulator and a commercial emulator.

Figure 3. Example of the J2ME application running on Sun's J2ME emulator and a commercial emulator

In order to invoke Web Services from a J2ME application, the mobile devices must support the Java Specification Request 172 (JSR-172). At present, however, JSR-172 has no support to UDDI specification in a J2ME application. It must be pointed out that our framework solves this problem through the use of the service manager entity.

The application developed to mobile users presents different menus with different options, e.g.,: invocation, update, delete and searching of new Web Services. Figure 4 illustrate the main menu of the application. This application is entirely dynamic, that is, all screen display is generated according to the option selected by the user at runtime. For example, the screen display "Select a service", is made up by the services downloaded by the user at the present time; the screen display "Request of parameters" only request user the parameters required by the user invoked operation.

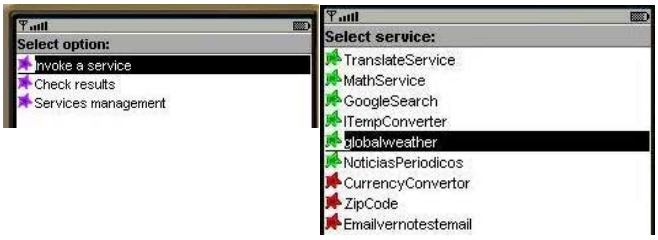

Figure 4. Main menu of the J2ME application and specific menu of "Invoke a service"

The main menu presents three different options to the user such is illustrated in Figure 4:

- **Invoke a service:** the set of available services to the mobile device is shown to the user. Two set of different services are available: (i) standard services registered by the service manager entity (represented by a green star) and (ii) services searched in UDDI (represented by a red star). Right image of figure 4 depicts such functionality. After a service is selected, the set of operations supported is shown to the user. Figure 5 illustrates a weather forecast service. Results of the operation invocated are shown in Figure 6.

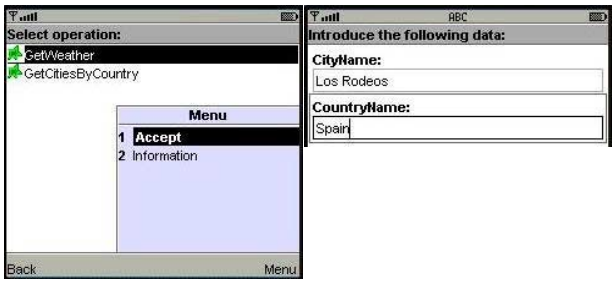

Figure 5. Global Weather forecast service: (a) Selection of the operation to invoke and (b) Introduction of the parameter values

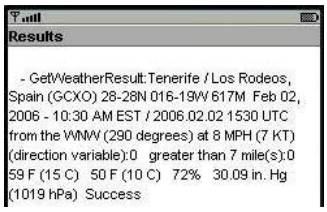

Figure 6. Global Weather forecast service: Results of the invoked operation

- **Check results:** allows users to check the results from the different invocations achieved. A list with the different invoked services by the user is saved with the following attributes: operation, parameters values, invocation date and result computed. This way, user can read results of previous services invoked at anytime. Figure 7 illustrates newspaper reports that have previously been invoked and consulted later.

Figure 7. Consultation of newspaper reports that have previously been invoked

- **Services management:** allows users to choose different management tasks such as: (i) update Web Services from servers, (ii) searching services in UDDI and (iii) remove services from the mobile device. Figure 8 illustrates results computed for a search in UDDI about services related to a currency conversion.

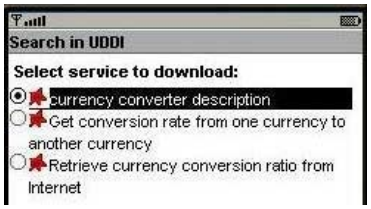

Figure 8: Results computed by services related to "currency converters" in UDDI registry

5. DISCUSSION

The service oriented architecture described in section 3 is based on open standards and specifications that allow accessing Web Services of wired network from mobile phones at runtime without prior knowledge of available services.

At present, we found that open source development tools for building, deploying and testing production quality work well together. However, we have found three drawbacks of using general Web Services technology for developing mobile services.

First, the Java Specification Request 172 (JSR-172) required for invocating Web Services from a mobile J2ME application does not support UDDI specification and SOAP encoded messages. As a result, mobile devices cannot access directly to UDDI registry and the mobile devices can only access to Web Services with SOAP messages with literal representation (RPC/literal and document/literal). We have solved both problems by the use of service manager entities.

Second, we have found that the use of UDDI registry provides high time responses and also many of the services published in UDDI registry are not correctly published. Thus, all the services consulted through UDDI registry must be verified by the service manager, before the results are sent to the mobile user. Also, UDDI registry has been not designed in order to support dynamic binding and fault-tolerance. Moreover, the UDDI registry can do neither any load balancing nor automatic forward to a different

URI in case of failures. These problems need to be solved at the level of individual Web Service providers using techniques like replication and server clustering. In order to solve these problems, a major verification of the services published in UDDI registry is required. We have solved this problem verifying the services required to UDDI registry by the service manager entity. However, these confirmations increase the response time to mobile users.

Third, specific implementations must be developed in order to support complex types when dynamic invocation interface is used.

6. CONCLUSIONS & FUTURE WORK

In this paper, we propose mobile devices as complete participants in networked service-oriented architectures to enlarge the variety of accessible services and new business opportunities in the mobile space. We introduce service managers as intermediate entities between service providers and mobile devices. These managers are Web Services components that act as clients over the network of services and as servers to the mobile devices. We propose the use of dynamic invocation interface in order to compute at runtime: (i) Web Services descriptions from service providers and UDDI registry and (ii) invoke services selected by mobile users. We provide a uniform infrastructure using XML-encoded data exchange between service managers and mobile devices. With this approach, we delegate the business logic to service managers, solving the problems faced by direct access from mobile devices to Web Services and reducing the computational cost of mobile devices and therefore, optimizing the response times to mobile users and memory resources. The design of our framework taking into account service manager's elements allows that service providers to create, update and change services at anytime and mobile users may locate new services without knowing the accessible services. We have implemented a first prototype with open source tools. At the present time, we have found that available technology to implement Web Services technology using open source software is appropriate. However, some drawbacks were detected such as: (i) UDDI registry cannot be directly accessed by mobile devices and (ii) specific implementation is required in order to support complex types when dynamic invocation interface is used.

Once we have tested the viability of Web Services technology and the dynamic invocation interface based paradigm to support basic m-services, our future work will be focused on adding semantic and context-awareness notions with the purpose of intelligent matching and personalized responses. Future trends also involve incorporating authentication policies and testing all it with the available Web Services technology in m-commerce scenarios.

7. REFERENCES

[1] Chakraborty, F., Perich F., Avancha S., and Joshi A. DReggie: Semantic service discovery for m-commerce applications. *Workshop on Reliable Secure Applications Mobile Environments, 20th Symposium Reliable Distributed System*, New Orleans, LA, 2001.

[2] DAML. The Darpa Agent Markup Language. Available from URL: http://www.daml.org/

[3] Guttman, E., Perkins, C.E., Veizades, J., Day, M. RFC 2608 : Service location protocol, version 2. *Internet Engineering Task Force*, june 1999. Available from URL : http://www.rfc-editor.org/rfc/rfc2608.txt

[4] Gustavo, A., Casati, F., Kuno H., Machiraju, V. Web Services: concepts, archictectures and applications. *Springer-Verlag publications*, Berlin 2004.

[5] I-mode. NTTDoCoMo. http://www.nttdocomo.com

[6] Jini Network Technology. http://www.sun.com/software/jini/

[7] Patil A., Oundhakar S., and Verna K. METEOR-S Web Service Annotation Framework, In *Proceedings International WWW Conference*, New York, USA, 2004.

[8] The Salutation Consortium, Salutation Architecture Specification, http://www.salutation.org, October 1997.

[9] UDDI. *Universal Description, Discovery, and Integration*. http://www.uddi.org/

[10] UPnP. Universal plug and play. Available from URL: http://www.upnp.org/

[11] Vinoski, S,. Web Services Interactions Models, Part 1: Current Practice. In *IEEE Internet Computing*, Vol 6, Nº 3, pp. 89-91, 2002.

[12] Waldo., J. The Jini architecture for network-centric computing. *Communications ACM* 42 (7), 79-821 1999.

[13] W3C: World Wide Web Consortium. *Simple Object Access Protocol. (SOAP)*. http://www.w3.org/TR/soap/

[14] W3C: World Wide Web Consortium. 2003. *Web Services Description Language (WSDL)*. http://www.w3.org/TR/wsdl

[15] Katherine S. Willis. Gap: Mobile Applications and Wayfinding. In *Workshop for User Experience Design for Pervasive Computing. Pervasive* 2005, Munich, Germany.

[16] A. Gugliotta, L. Cabral, J. Domingue, V. Roberto, M. Rowlatt, and R. Davies (2005). A Semantic Web Service-based Architecture for the Interoperability of E-government Services. In *Proceeding of Web Information Systems Modeling Workshop (WISM 2005)* in conjunction with The 5th International Conference on Web Engineering (ICWE 2005) Sydney, Australia, 25-29 July, 2005.

[17] OGSI Service Platform. http://www.osgi.org/

[18] Humberto Cervantes, Richard S. Hall. Autonomous adaptation to dynamic availability using a service oriented-component model. In *26th International Conference on Software Engineering (ICSE 2004)*, 23-28 May 2004, Edinburgh, United Kingdom.