



# Using Web Services for supporting the users of wireless devices

T. Pilioura\*, S. Hadjieftymiades, A. Tsalgatidou, M. Spanoudakis

*Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Panepistimiopolis, TYPA Buildings, Ilisia, 157 84, Athens, Greece*

Available online 27 June 2005

## Abstract

The Web Service paradigm is currently considered as the most promising and rapidly evolving technology for developing applications in open, distributed and heterogeneous environments. The proliferation of this new technology has coincided with significant advances in the hardware and software capabilities of wireless devices. The combination of the two worlds (i.e. making wireless devices capable of providing and consuming Web Services) is considered of major importance to the computing industry for the forthcoming years. This paper describes two scenarios of using Web Services in wireless devices, identifies their advantages and supporting technologies. A reference architecture for the use of Web Services in the wireless world is proposed. Finally, the results of the performance evaluation of an experimental setup are presented.

© 2005 Elsevier B.V. All rights reserved.

*Keywords:* Web Services; Wireless devices; Reference architecture

## 1. Introduction

Nowadays, we are witnessing the rapid explosion of wireless communications. The continuously increasing number of users of wireless terminals makes the latter an ideal channel for offering personalized services to the users of wireless devices. Currently, wireless carriers offer services that allow information such as weather, stock quotes, news, traffic, or sports updates to be “pushed” to wireless terminals. This static provision of services, although

useful, can not efficiently address the needs of users of wireless devices who want to dynamically decide on the services they would like to “consume” depending on their individual preferences and context. For example, a driver would like to receive at a low price, information about traffic in the specific area s/he is currently moving. This dynamic discovery, composition and use of services on the spot, regardless of where the service requestor or provider is located (i.e. either at his/her office or home using a desktop computer or on the move, using a wireless device) and of how the service has been implemented is the main topic investigated in this paper. More specifically, we investigate how this can be achieved through the use of the Web Service (WS) technology [2,4]. WS technology transforms the WWW from a publication me-

\* Corresponding author.

*E-mail addresses:* [thomi@di.uoa.gr](mailto:thomi@di.uoa.gr) (T. Pilioura), [shadj@di.uoa.gr](mailto:shadj@di.uoa.gr) (S. Hadjieftymiades), [atsalga@di.uoa.gr](mailto:atsalga@di.uoa.gr) (A. Tsalgatidou), [e.spanoudakis@di.uoa.gr](mailto:e.spanoudakis@di.uoa.gr) (M. Spanoudakis).

dium to a programming platform, where software components are described, advertised, discovered and invoked through their declared Application Programming Interfaces (API). The successful integration of the Wireless Communications and Web Service technology realms can lead to the creation of new and compelling business scenarios that expand on a broad array of endpoints (wireless devices, servers and PCs operating over the Internet, home devices), thus facilitating “computing at the edge”.

However, most of the protocols used in the wired WS world (e.g. SOAP [19], UDDI [26]) are inappropriate for the wireless world due to the specific characteristics of the, still expensive, wireless medium, of the limitations of the wireless devices and of the different context where they are used. This means that, in order to efficiently use the WS technology for supporting the needs of the users of wireless devices, the WS technical considerations need to take into account issues like cost, temporary unavailability, i.e. link outages (which is quite common in the wireless world), as well as performance related issues in order to cater for the efficient operation under restricted bandwidth conditions.

The goal of this paper is to investigate the use of WS in the wireless world and to propose a reference architecture for supporting the users of wireless devices. The rest of the paper is organised as follows. Section 2 presents a motivating scenario and highlights the requirements both for developers and end-users. Section 3 describes the WS model and the way it addresses the developers’ requirements. This is followed by a state-of-the-art in the area of WS and wireless devices described in Section 4. Then, in Section 5, we present two scenarios of using WS in wireless devices. In the first scenario, the wireless device assumes the role of the WS requestor under two alternative architectural configurations: a fat client and a thin client configuration. In the second scenario, the wireless device assumes the role of the WS provider. For each scenario we discuss its advantages and the technologies that help in addressing the end-users’ requirements. In Section 6, we provide a reference view of the building blocks required by the various entities participating in the scenarios and in Section 7 we present the results of our experiments on the second scenario. Our concluding remarks are included in Section 8.

## 2. Motivation

Consider an auction-based marketplace where buyers and sellers come together in order to meet their respective buying and selling needs. This marketplace may be supported by either a centralised or a decentralised architecture. In a centralised architecture, there is a central node where sellers register their goods for sale and buyers make bids on them (e.g. eBay, eBid). In a decentralised architecture, every seller disposes the appropriate infrastructure for advertising his goods and for accepting bids submitted by the buyers. The latter architecture has several advantages over the centralised one (hence, we adopt it for our scenario):

- (i) No single point of failure,
- (ii) Better distribution of work,
- (iii) Instant notification of the seller regarding the auction status, and
- (iv) Sellers’ autonomy in the provided functionality.

Buyers and sellers can be connected to this marketplace using various means such as desktop computers and wireless devices. The latter enable the sellers and buyers to monitor auctions, make or accept bids, etc., not just when they are home or at the office, close to their “wired” desktop PCs, but also when they are on the go, using wireless Internet devices such as mobile phones and personal digital assistants (PDAs).

The feasibility of the above scenario depends on the satisfaction of the following functional requirements:

- R1: Buyers want to search the auction listings (provided by various sellers) for goods related to their interests and needs using either their desktop PCs or their wireless devices.
- R2: Buyers should be able to register and, automatically, receive notifications by the various sellers when goods of their interest are being put into auction.
- R3: Whenever a seller (who may also be mobile and thus participate in an auction via a wireless terminal) offers a certain good for sale in the marketplace, auction bids should reach him with the minimum possible interruption.

R4: Sellers should be able to inform the interested buyers about the result of the auction in which they are participating and arrange the financial part of the transaction.

R5: Sellers need to obtain information about the buyers, e.g. payment choices, home address or current location in order to deliver the goods.

The satisfaction of the above functional requirements leads to a number of requirements for the developers and the end-users (sellers and buyers) of the distributed marketplace scenario:

### 2.1. Developers' requirements

- *Easy and fast deployment*: Developers should be able to provide new services to the users of wireless devices without the investment and delays of traditional application development. They should be able to develop new applications by reusing and/or combining existing functionality.
- *Interoperability*: Applications running on the wireless devices should communicate seamlessly with the applications of the wired world (e.g. with a banking application for the financial settlement of an auction-related transaction). This means that developers need not be aware of the operating system, development environment and technology used for building the various applications.

### 2.2. End-users' requirements

- *Dynamic selection of services*: The services provided to the end-users should not be predefined but dynamically specified at run-time based on parameters like the current context (e.g. location), the required Quality of Service (QoS) and the cost. For example, a buyer might look for an auction service

provided by a seller, which is situated at a convenient location for the buyer.

- *Quick response*: The limitations of wireless devices such as restricted bandwidth, low processor power and limited memory should not hinder the operation of an application. Moreover, the time criticality of the auction process intensifies the need for quick response.
- *Disconnected operation*: Problems caused by non-deterministic loss of network connectivity or finite battery power that wireless devices experience should not affect the functionality of the application. The results of the application (e.g. the submission of a bid to the seller) should be returned at a later time when the connection is re-established.

## 3. Web Service (WS) technology and the wireless world

In this section we examine the Web Service (WS) technology and the way it can be used in supporting scenarios (like the one described above) which can be materialized over wireless networks. Web services constitute a new model of using the WWW. This model allows the publishing of business functions to the Web and enables universal access to these functions.

The WS model involves the following basic activities: describe, publish/unpublish/update, discover and invoke/bind. Three different roles are foreseen: WS provider, WS requestor and WS broker (see Fig. 1). A WS Provider is the party that provides specific software applications as Web Services. A WS Requestor has a need, which can be met by an available WS. A Requestor could be an application program, another WS or a user accessing the service through a desktop or a wireless browser. A WS Broker is the party that

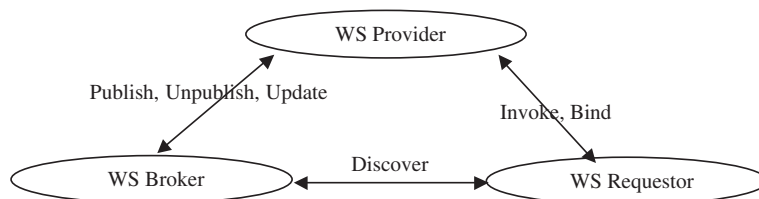


Fig. 1. The WS model.

provides a searchable repository of service descriptions where service providers publish their services and service requestors find services and obtain relevant binding information for them.

The use of standard technologies for service description, communication and data formats is essential for supporting the interaction between the above parties. Currently, the most widely used standards for describing, advertising, discovering and binding WS are the Web Service Description Language (WSDL) [27], the Universal Description, Discovery, Integration (UDDI) [26] and the Simple Object Access Protocol (SOAP) [19]. However, despite the support offered by these standards, there are a lot of issues related to WS description, composition, discovery, etc., which still remain open [25].

WS technology is characterised by several major advantages, such as easy and fast development, interoperability and just-in-time integration. Thus, it supports many of the developer's requirements set forth in the previous section. For example, the functional requirement R1 (query) of the discussed application scenario can be satisfied by using an existing WS (that operates successfully over wired networks) and adapting its output to the needs of the wireless medium, thus achieving easy and fast development.

Furthermore, in order to satisfy the aforementioned end-user requirements, we need to extend the WS technology in the wireless world, as some of the protocols used in the wired WS world (e.g. SOAP, UDDI) are inappropriate for use in the wireless medium. A relevant discussion is included in the following section.

#### 4. State-of-the-art

In this section we provide a brief overview of the key technologies associated with the extension of the WS framework in the wireless world. We identify three areas of interest:

- (a) the use of the WS protocols over the wireless interface,
- (b) the execution of WS-ready software (servers, clients) in the wireless device, and
- (c) the use of other web protocols in the wireless device.

With respect to area (a), we would like to emphasize that some of the protocols used in the wired WS world (e.g. SOAP, UDDI) are inappropriate for use in the wireless medium, hence, the increased interest for their optimisation for use in the wireless Internet world. Key commercial players like Microsoft and Sun recognise the importance of extending the WS paradigm in the wireless world and introduce specific features in their products [15,11]. In the recent past, optimised versions of SOAP and XML have been proposed for the wireless landscape. These proposals are trying to overcome the problems associated with the verbose (ASCII) nature of XML/SOAP. Examples of such proposals include the kXML [14], kSOAP [13] and Pocket SOAP [18] which are open source implementations of WS for execution on small wireless devices. Academic efforts in the same direction are presented in [3] and [24]. In [3] the issue of limited bandwidth is partially tackled by a two-phase protocol extension to the SOAP protocol. The first phase executes the server side Web operation and determines the information volume to be returned. The second phase uses such meta-information to perform time and cost estimates for determining whether to proceed or not with the results transmission to the wireless device. Tian et al. [24] propose a simple scheme that allows clients to specify whether they want to receive data compressed when requesting a web service. They show that compression is useful for poorly connected clients with resource-constrained devices despite the CPU time required for decompressing the responses.

With respect to area (b), we provide a brief overview of the software execution capabilities offered in contemporary mobile devices. Technologies like Java and J2ME, WAP, Mobile Execution Environment (MExE), EPOC/Symbian [1] have been incorporated by manufacturers into their products and provide a vast range of capabilities to mobile users.

In order to standardize components for mobile devices, the 3GPP (3G Partnership Project) developed the MExE specification [23] that defines a relevant framework. The aim is to support application development for a certain class of devices with a common classmark, rather than for specific hardware. The technologies that are considered by MExE are Java and WAP. MExE has specified two classmarks: Classmark 1 defines devices that support WAP 1.1 and

newer versions while Classmark 2 defines Java enabled devices (Personal Java). MExE also defines security features: security certificates indicating that an application is secure and also where it can be executed.

EPOC (from Symbian) has been designed as a communications centric operating system for mobile devices [6,1]. EPOC is an evolution of older software for PSION hand-held computers. EPOC targets smartphones and communicators, where there is a close interaction between the communications equipment and the software. Symbian has developed a number of different reference designs namely the Pearl, Crystal and Quartz.

There are also a number of interesting projects for porting Linux at hand-held devices. An interesting one is Pocket Linux [1], which can be executed on the Compaq Ipaq. The kernel has been re-engineered and optimized for small devices. Applications can be written in Java. Everything in Pocket Linux is based on XML, even the system databases.

With the advent of Java 2, Java split into 3 editions targeted at different devices and applications. Java 2 Enterprise Edition (J2EE) targets large enterprise applications and servers. Java 2 Standard Edition (J2SE) is the natural evolution of previous Java versions. The Java 2 Micro Edition (J2ME) is the scaled-down version that is optimized for mobile devices. J2ME comes in two configurations: the Connected Device Configuration (CDC) for devices where resources are not so scarce (e.g. set-top boxes and in-car systems) and the Connected Limited Device Configuration (CLDC) for small, mobile Internet devices.

The introduction of fully featured and compatible Java virtual machines in powerful mobile devices rendered feasible the execution of resource consuming server software in such environments. Typical examples of the discussed configurations are Enago mobile from IKV++ [9] and Apache/Tomcat.

In the (c) area of interest we are mostly concerned with the use of conventional (wireline) Web protocols in the wireless domain. This area is also termed “Mobile Internet”. The most important representative is the OMA WAP family of protocols [16]. The Open Mobile Alliance (OMA) is a very significant initiative towards the highest possible interoperability of mobile service platforms. Users

benefit from industry-wide efforts to enable services such as MMS (multimedia messaging service), web surfing and e-mail access to be shared by everyone, regardless of their operator, network or mobile device used. The most important technologies considered by OMA are MMS, Java and XHTML. Older versions of the WAP specification were incompatible with their wireline counterparts. The inter-working process was allocated to the WAP gateway component. WAP 2.0 relies on widely established technologies like HTTP and XHTML [28]. The latter is a family of current and future document types and modules that reproduce, subset, and extend HTML 4. XHTML family document types are XML based, and are designed to work in conjunction with XML-based user agents. In order to allow applications and devices which are not capable of supporting the full set of elements to work with a well-defined support, the W3C has introduced the concept of XHTML modules. OMA has defined a mobile profile for XHTML (XHTMLMP) as the base for mobile devices supporting WAP [6]. XHTMLMP consists of XHTML basic and some additional XHTML modules.

During the past years a number of efforts were made to architecturally fuse conventional WWW standards with wireless communications [5]. Notable examples of such efforts include the MobiScape Project and the IBM WebExpress platform [7]. Both systems capitalise on the proxy interface that modern browsers support and use a similar architecture (“intercepting technology”). The Web Express platform addresses the requirements of transaction processing through the adoption of “differencing techniques”. As discussed in [22] such mechanism achieves a 50% reduction in the volume of information exchanged over the wireless medium.

## 5. Scenarios of using Web Services in wireless devices

In this section we examine two scenarios for using WS in wireless devices and we discuss the advantages of using WS and supporting technologies. In the first scenario the wireless device assumes the role of the WS requestor under two alternative architectural configurations, a fat client scenario and a thin client

scenario. In the second scenario, the wireless device plays the role of the WS provider.

### 5.1. Wireless device acting as WS requestor

This scenario, with the wireless device acting as a requestor of WS, is best suited for nomadic users invoking typical services, such as services returning currency exchange rates, while on the move. It is also well suited for context-aware applications, e.g., locate auction services provided by sellers doing business in the proximity of the user. We examine two architectural variations of the scenario: a fat client scenario and a thin client scenario with the introduction of a proxy server.

#### 5.1.1. WS-aware wireless device (fat client scenario)

In this architectural variation, the entity that plays the role of the WS requestor is the wireless device itself (Fig. 2). The wireless device needs to dispose a WS client in order to enable the provision of services to its users. It interacts with the WS provider and the WS broker using WS protocols over the wireless network (e.g., WLAN, GSM/GPRS). The wireless device is a *fat client* with XML processing capabilities. Fat clients provide more sophisticated support in terms of application specific functionality [3]. The use of WS in such fat clients has numerous advantages:

- It enables handset manufacturers to rapidly deploy Internet solutions built on open standards.
- It makes applications more dynamic as they can invoke different services or service implementations based on the user's context.

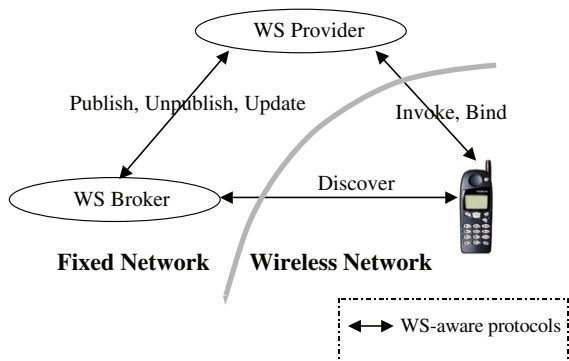


Fig. 2. Wireless device as a service requestor.

- It facilitates the interoperability and integration with enterprise applications and applications running on other wireless devices.

This scenario is particularly applicable in cases where a user of a wireless device must interact with multiple companies/entities or information sources in order to satisfy his/her needs. The human involvement in this interaction must be minimised for efficiency and security reasons, since the user is usually involved in another activity. For example in the auction-based marketplace scenario if the buyer, while driving, wishes to participate in some auctions for laptops, it is more convenient and safe to have all the interaction with various companies/entities or information sources handled automatically through the use of the WS technology and, at the end, to inform the buyer about the result of his participation in the various auctions.

As mentioned before, the use of WS in the wireless world need to take into account the device limitations and characteristics in order to address the end-user requirements discussed in Section 2. In the following paragraphs we discuss how these requirements can be addressed.

The requirement for *dynamic selection of services* is still an open issue in the wired WS world. This is further complicated in the wireless world due to the mobility issue. To be more specific, discovering services based on context such as location depends on the ability to enhance the WS directory's service entries with contextual information. WSDL documents do not include this kind of information. UDDI partially covers this need through taxonomies. One of the three standard taxonomies cited by UDDI is a geographic categorisation system complying with the ISO geographic taxonomy (ISO-3166). However, the information in this taxonomy is very coarse-grained and describes the geographical area but not specific city or district where the products/services are offered. In general, the more information (syntactic, semantic, functional, non-functional and behavioural) used for describing Web Services, the richer users' requests for services can be and the better the chances for finding the best fit. Therefore, the research work needs to be performed along these lines in order to address the requirement of dynamic service selection in the wireless world. Preliminary research results in

this area include a proposal (described in [17]) for extending the UDDI registry in order to overcome the limitations of the UDDI standard in the support of the location-based service discovery.

The requirement for *quick response* can be tackled by using wireless optimised versions of SOAP and XML as the ones discussed in Section 4. Another way for achieving *quick response* is by using a call-back pattern for information needed by the requestor at periodic time intervals. Thus, instead of having a requestor sending request messages to the provider, the provider periodically pushes the service responses to the requestor. This eliminates the burden of continuously sending the same request. Provisions for such notification mechanism are foreseen in the latest version of the SOAP standard [20]. Thus, the functional requirement R2 of the distributed marketplace scenario may be provided by the subscription of the buyer to the relevant web service and subsequently receiving notifications by it.

Furthermore, a quick response can be obtained by calling a WS asynchronously, which is also supported by SOAP [20]. Asynchronous communication can mask out the slow wireless network and let the wireless device perform other tasks instead of being blocked, while waiting for the response to return. This requires a Message Identifier and Correlation Handler at both the sending and the receiving SOAP application (see Fig. 3). The Message Identifier Handler in Host I generates a unique message identifier and inserts it into the SOAP Header of the request SOAP message. This message is processed by App2 and a response message is assembled. This includes a SOAP Header (built by

the Message Correlation Handler in Host II) that links the response message to its associated request. App1 is notified when the response is received and uses the correlation information within the received message to match the response to the concerned previous message.

The requirement for *quick response* can also be addressed at a lower protocol level. SOAP is based on HTTP, which, in turn, is based on TCP. Since bandwidth in the air interface is a very scarce resource, several efforts have been made to optimise the WWW for wireless devices. From those efforts, notable is the IBM WebExpress, briefly presented in Section 4.

The requirement for *disconnected operation* can be satisfied by using techniques like message queuing. Outbound messages are added to a queue and are sent when a network connection can be established between a wireless device and a server. Lightweight versions of JMS [10], such as the one provided by the Ibus/Mobile product [21] allow the invocation of a WS in a fault-tolerant and reliable manner.

5.1.2. *WS-agnostic wireless device—a proxy-based architecture (thin client scenario)*

In order to cater for wireless devices without WS functionality, we introduce another variation of the first scenario. In this architectural variation, we introduce a proxy entity that plays the role of the wireless device representative (Fig. 4) in the fixed network infrastructure. This entity may be provided for example by the wireless carrier. The proxy takes the user’s request (i.e. required functionality, constraints, etc.), interacts through WS protocols with the service broker and the service provider and returns the results to

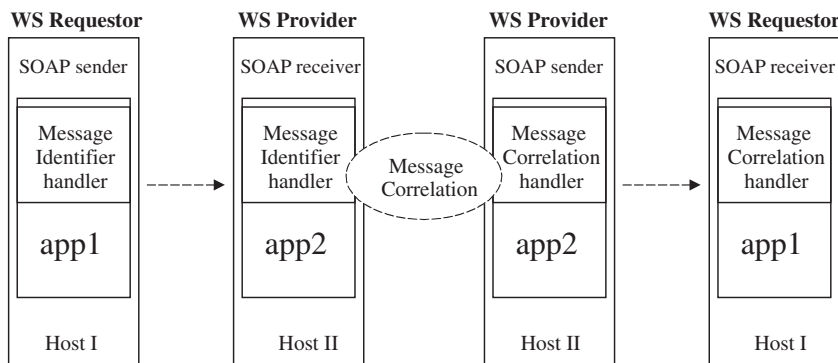


Fig. 3. Asynchronous messaging.

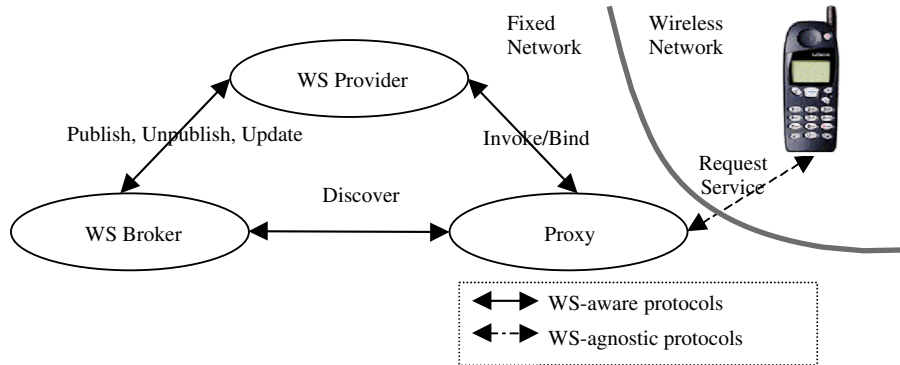


Fig. 4. Proxy as a wireless device representative.

the wireless device using WS-agnostic wireless protocols (such as WAP/XHTML, i-Mode/cHTML). Its role is the provision of services in a way adapted to the capabilities and the constraints of a wireless device. The use of the proxy relieves the wireless device of the time- and processor-consuming WS tasks as the relevant workload is passed to the proxy. The functionality of the proposed proxy is quite similar to that of the WAP gateway performing adaptation between conventional WWW protocols (i.e., HTTP, TCP) to wireless-optimised variants (i.e., WSP, WTP, UDP).

The proxy-based architecture is particularly applicable in the case where the user of the wireless device moves into an unfamiliar computational environment. In such a case the wireless device may detect a local proxy and obtain through it services for which it has no previous knowledge. For example, we could consider an individual entering an area and obtaining access to auction services provided by sellers in the area.

For addressing the end-user’s requirements set forth in Section 2 several technologies could be used and these are discussed below. Regarding the *dynamic selection of services* there are no further issues to be discussed here than the ones that we have already analysed in the fat client scenario.

The *quick response* requirement may be addressed by using caching. The proxy could cache the response to a WS request in situations where a subsequent request can be safely answered with the same result. For example, the response of a WS providing the currency exchange rates for a specific day remains valid during the day. Thus, caching can be used in order to provide faster access to the information,

reduce network bandwidth requirements and lower the workload on the service provider. Provisions for such caching mechanism are foreseen in the SOAP standard [20]. Fig. 5 illustrates a potential architecture. An example of the messages that are exchanged to achieve such interaction pattern is shown in Fig. 6. The first of the messages shown is a SOAP request (“ExchangeRatesRequest”) sent by the proxy (“Host I”) to the provider of the service (“Host II”). The service provider sends the service response (“ExchangeRatesResponse”) back to the proxy with an additional SOAP Header to control any caches along the return path. The CacheControl Header contains a CacheKey (“ExchangeRates”) that allows matching of future requests to the cached response together with an Expires element that sets the time that the local copy must be considered invalid. At the proxy, the CacheControl header information is used to make a local copy of the response message, keyed by

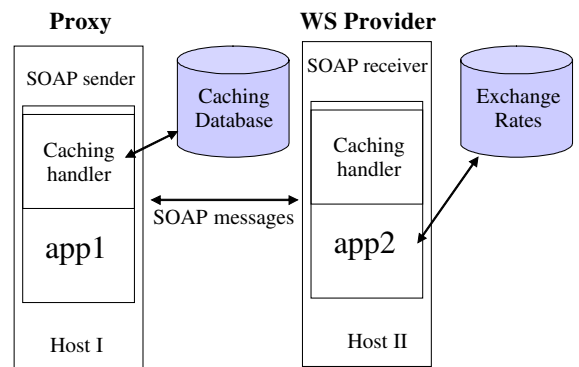


Fig. 5. WS response caching.

```

// SOAP request for exchange rates
<?xml version="1.0" ?>
<env:Envelope xmlns:env="http://www.w3.org/2001/09/soap-envelope">
  <env:Body>
    <c:ExchangeRatesRequest xmlns:c=" http://www.di.uoa.gr/2001/06/exchangeRates">
      </c:ExchangeRatesRequest>
    </env:Body>
  </env:Envelope>

// SOAP response with caching header received by the service provider
<?xml version="1.0" ?>
<env:Envelope xmlns:env="http://www.w3.org/2001/09/soap-envelope">
  <env:Header>
    <ca:CacheControl xmlns:ca="http://www.di.uoa.gr/2001/06/cache">
      <ca:CacheKey>ExchangeRates</ca:CacheKey>
      <ca:Expires>2004-05-29T08:00:00Z</ca:Expires>
    </ca:CacheControl>
  </env:Header>
  <env:Body>
    <c:ExchangeRatesResponse xmlns:c="http://www.di.uoa.gr/2001/06/ exchangeRates">
      .....
    </c:ExchangeRatesResponse>
  </env:Body>
</env:Envelope>

```

Fig. 6. SOAP messages for caching interaction scenario.

the CacheKey. The copy will be invalidated at the time specified by the Expires element. The proxy can use the cached list of currency exchange rates in order to respond to subsequent requests.

Given the caching capabilities provided by SOAP, it is worth considering another optimisation, namely the differencing mechanism, which is provided by the WebExpress, described in Section 4.

As regards the disconnected operation requirement, this could also be addressed by using the caching mechanism. The proxy can cache service invocation results and forward them to the wireless device when connection is re-established.

We need to mention here that there is a special challenge related to this variation of the first scenario: The proxy is a central entity that controls all user data

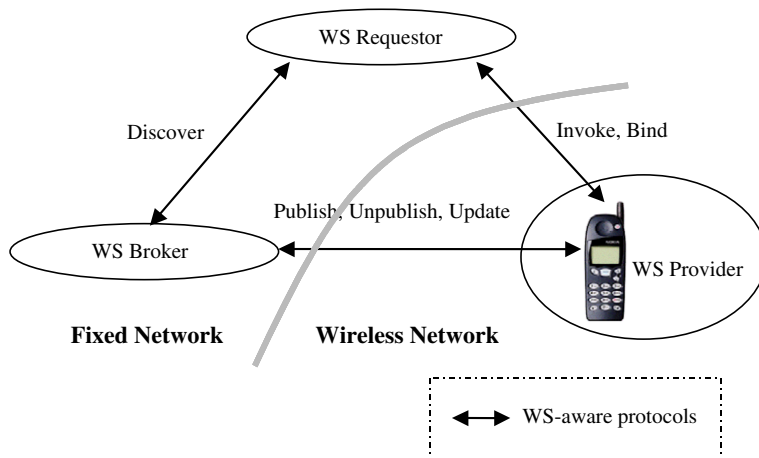


Fig. 7. Wireless device as WS provider.

and decides what services are accessible by the wireless device. This model could cause serious business concerns in case that the proxy is controlled by an untrustworthy, insecure or abusive monopoly. Furthermore, the proxy constitutes a single point of failure, exposing the whole network to attacks.

### 5.2. Wireless device acting as WS provider

A challenging scenario regarding the provision of WS is the one involving the WS service provision at a wireless terminal (see Fig. 7). As we examined in Section 4, nowadays, the capabilities offered by portable equipment are getting more and more advanced. Hence, the case of a Wireless device Hosted WS provider (hereinafter to be referred to through its acronym, WH-WSP) scenario is highly foreseeable and technically feasible.

We could consider various applications where this scenario can be used. For example, in our distributed marketplace scenario, a seller can be a mobile entity using a wireless device to offer goods for sale in an auction. This can be supported by a WS (provided by the wireless device) that performs auctions for the goods offered for sale. This WS enables prospective buyers to obtain different types of information. (e.g., auction status, list of offered goods).

In this scenario, in order to meet the requirement for *dynamic selection of services* provided by wireless devices, we must confront the challenge which is related to the continuously changing location of the WS provider. Typically, when Internet-powered wireless nodes move to another location, a new address is assigned to them. By changing its address, a WH-WSP becomes partly inaccessible by the rest of the world since the offering point of the service changes. Provision should be taken to update the WS registry on the new address where services will continue to be offered to their requestors (Fig. 8). In our auction-based marketplace, this is particularly important in order to continue to receive bids by the buyers with the minimum possible interruption.

To meet the technical challenge of updating the WS registry so that it always reflects the current address of the WH-WSP, we consider the following two solutions:

- The WH-WSP can rely on currently available APIs for automatically updating the UDDI service when relocations are experienced. Such functionality is quite critical and should, potentially, be incorporated in the emerging wireless WS frameworks.
- A second solution to the problems caused by the WH-WSP mobility is to hold the WS provider responsible for immediate notifications to known

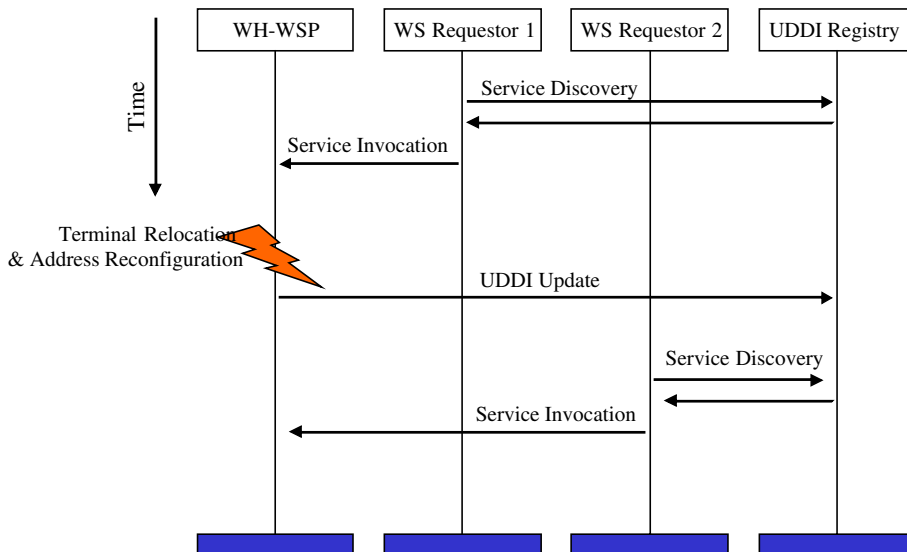


Fig. 8. UDDI update after wireless device address reconfiguration.

clients–requestors or clients that have asked to receive notifications for any relocation of the WS provider and for related side effects. This applies in cases where the number of requestors is limited and there is no entity acting as a WS broker.

In terms of implementation, the first solution can be realised through the use of the established UDDI APIs (Java Web Services Development Pack, jUDDI [12], IBM Web Services Toolkit) and a network monitoring agent within the wireless device. The monitoring agent constantly observes the network interfaces and detects address changes. Such changes are then passed, by the WS provider, to the UDDI registry through SOAP calls belonging to the publishers' API. One could easily extend this scheme to handle other contextual changes beyond a simple network address change (e.g., change of the terminal's geographical position, change of environmental conditions). This extended scheme requires that some kind of contextual information exists within (or can be deduced from) the service registry (Fig. 9). In this case, the monitoring agent is some kind of sensor (e.g., GPS receiver, digital thermometer) that detects changes in the general environment and stimulates the appropriate notifications.

The second solution which holds the WS provider responsible for immediate notifications to known clients–requestors, can be realized using SOAP, since provisions for such notification mechanism are foreseen in the latest version of the protocol.

Regarding the requirements for quick response and disconnected operation there are no further issues to

be discussed here than the ones that we have already analysed in the previous scenarios.

## 6. Building blocks for using Web services in wireless devices

The examination of the various scenarios leads to some conclusions regarding the necessary components that realise the use of web services in the wireless world. Fig. 10 depicts the building blocks of the 4 entities (WS Provider, WS Broker, WS Requestor and Proxy) participating in the two scenarios presented in the previous sections.

We notice that the blocks of Fig. 10 that correspond to the WS Provider and the WS Requestor are split in two parts. The blocks that appear on the left part contain all the components that are necessary for both wireless and wired WS Providers/Requestors. The blocks that appear on the right part comprise the additional components that are needed in case that the WS Provider/Requestor is hosted on a wireless device. In the following we provide a brief description of the functionality of the various blocks of each entity.

A WS Provider, in order to support service provision in a wireless context, needs to have the following building blocks:

- *Adaptation handler*: This block allows the generation of a personalised and adaptive user interface or markup (e.g., HTML, WML, VoiceXML) by taking into account the user preferences and the capabilities of the wireless device.

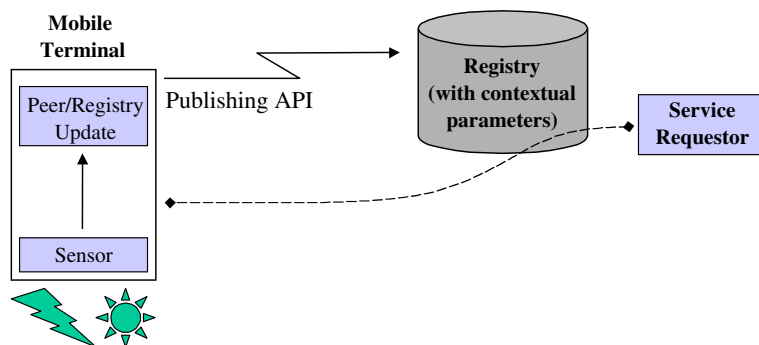


Fig. 9. General architecture for context-sensitive WS provision.

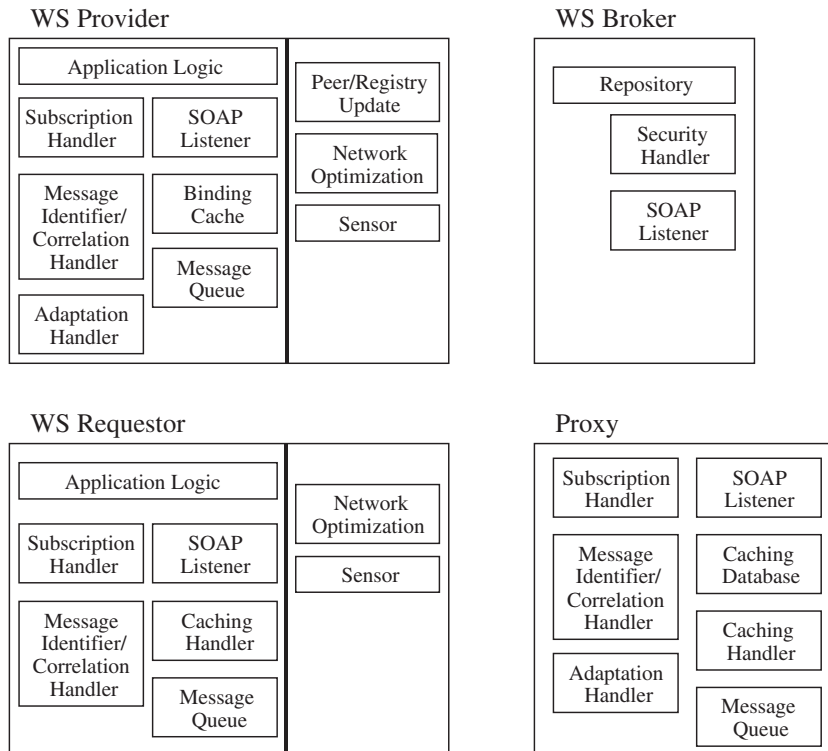


Fig. 10. Building blocks for using web services in the wireless world.

- *Subscription handler:* It is responsible for realising the subscribe/notify interaction pattern. It enables a SOAP application to subscribe to a WS for notifications as well as to handle notification subscriptions by other applications.
- *SOAP listener:* This block is responsible for performing the following operations: serialise method calls into SOAP packets, deserialise SOAP packets into Java calls, wrap XML documents into SOAP packets, unwrap XML documents from SOAP packets, submit SOAP requests and handle the responses as well as accept SOAP requests and return the responses.
- *Message identifier/Correlation handler:* The role of this block is to enable asynchronous messaging. The Message Identifier Handler/Correlation generates unique message identifiers and adapts the header of the SOAP request and response in order to link the response message to its associated request.
- *Message queue:* It represents the place where the SOAP messages are maintained in order to achieve reliable and fault-tolerant WS invocation.
- *Application logic:* This block refers to the coding of the business functionality that the WS Requestor or WS Provider must perform.
- *Binding cache:* It constitutes a table maintained by the WS Providers with the addresses or contact information of known or recent WS Requestors.
- *Sensor:* This block detects changes in the general environment such as alterations of the physical location, physiological state (body temperature, heart rate, etc.) or emotional state (angry, calm, etc.) and stimulates the appropriate notifications.
- *Network optimiser:* The role of this block is to reduce the data volume and latency for communication over the wireless interface by using techniques such as differencing and HTTP Header reduction.
- *Peer/Registry update:* It is present only on a WH-WSP and serves the need for notifying the WS registry and/or some known requestors (that have asked, beforehand, to receive notifications) when the WH-WSP changes address.

A WS Requestor acting in a wireless context should have the following building blocks:

- *Subscription handler, SOAP listener, Message identifier/Correlation handler, Message queue, Application logic, Sensor and Network optimiser*: as described above.
- *Caching handler*: It is responsible for managing the caching of WS responses (build appropriate SOAP headers, invalidate cached responses when the caching period expires, etc.).

The WS Broker should consist of the following components:

- *SOAP listener*: as described above.
- *Security handler*: It is responsible for all security facets, namely identification, authentication, authorisation, integrity, confidentiality and non-repudiation. It allows the broker to make informed decisions about who is accessing the information and to enforce access control on the data it keeps.
- *Repository*: This block constitutes the place where the service descriptions are stored.

Finally, the proxy should have the following blocks:

- *Subscription handler, Adaptation handler, SOAP listener, Message identifier/Correlation handler, Message queue, Application logic and Caching handler*: as described above.
- *Caching database*: This block represents the place where the cached WS responses are being held.

The blocks described above are the main elements that are needed in order to provide and consume WS on wireless devices. Fig. 10 could act as a point of reference in order to provide a high level understanding of how WS-based wireless applications function, being used as a basis for architectural comparisons, or to act as possible starting point for the development of respective types of systems.

## 7. Experiments on the wireless WS provider scenario

To assess and illustrate the feasibility of the scenario introduced in Section 5.2, we have developed a

pilot implementation of the Wireless device Hosted WS Provider (WH-WSP). The basic objectives of our implementation were:

- To investigate performance issues (benchmarking) with respect to the quick response requirement stated in Section 2,
- To assess the feasibility of the proposed technical solution with existing development tools,
- To demonstrate how the WH-WSP supports the dynamic selection of services by handling the change in the location of the WS provider and the change in the contextual information,
- To quantify the overheads associated with the mechanisms introduced in Section 5.2.

Our WH-WSP implementation is based on a PDA (Ipaq 3870, running Pocket PC 2002) equipped with a 10 Mbps WLAN (IEEE 802.11b) PCMCIA card. The Java Virtual Machine is based on the JclFoundation profile of the J9 JVM, provided by IBM's WebSphere Studio Device Developer. Extra classes were added on this JVM to support UDDI, SSL and SOAP functionality. The web server running on the PDA is a lightweight multi-threaded implementation serving SOAP requests. As a registry, we have used the IBM test UDDI registry [8].

The developed web service is hosted on the PDA and reports current information on the status of the mobile terminal, along with other terminal information, e.g. its new IP address. Thus, it emulates the distributed marketplace scenario discussed in Section 2 (the PDA represents the goods selling entity). Further operations performed by the PDA-hosted WS provider include:

- Detects changes in the assigned IP address caused by relocations of the PDA.
- On the detection of a change in the IP address, it updates the UDDI, preserving in this way its accessibility by other clients who wish to use the provided PDA-hosted web service.

The emulated scenario involved a constantly roaming user (actor) that tries to sell several goods through the auctioning process. The Actor carries a PDA that can respond to numerous, concurrent WS queries of

interested parties (list of goods, auction status, etc.). Our focus is on quantifying:

- The UDDI Registry Update time consumed by the PDA-hosted WS provider in order to notify UDDI about its current address.
- The time taken by clients to query the UDDI Registry (UDDI Search time) about the current address where the Web Service is being offered.
- The time taken by the wireless terminal to serve incoming WS requests caused by numerous clients (WS Execution Time).

To illustrate the differences between the wireless and the fixed WS providers, e.g., implication of limited bandwidth and computing power, we have also implemented the following functionality in a fixed terminal:

- Simulate and detect the address change and subsequently update the UDDI registry.
- Receive and dispatch client requests.

A custom stress tool was implemented for addressing multiple concurrent requests, originating from different nodes, to the UDDI and to the PDA-hosted WS Provider. Our specific findings from the realization of a series of experiments are discussed below. Initially, we examine the UDDI Registry Update operation, then the UDDI Search operation and the WS Execution, both at a PDA-hosted WS Provider and at a fixed terminal.

In Fig. 11 we present the time taken by two different types of WS Providers (a wireless provider hosting services on a PDA and a fixed one) to update the IBM UDDI Registry. We observe that the time

requirement is comparable in the two considered scenarios. Differences are in the order of 50–60 msec and can be attributed to the bandwidth and computing power deficiency that is inherent to the wireless terminal. The presented time refers to the detection of the IP address change and subsequent invocation of the UDDI update routines.

In Fig. 12 we plot the time required for the UDDI Search in order to discover a WS and the time required for the execution of the WS hosted at the PDA, against the number of concurrent clients accessing the UDDI and calling the PDA-hosted WS. To statistically secure our observations we have repeated each experiment five times irrespectively of the number of clients, e.g. in the 10-clients case we plot  $5 \times 10$  measurements. In Fig. 12 we have also included 3rd order polynomial interpolation lines. Our initial observations are that the UDDI Search times are consistently higher than the WS Execution Times. The same observation applies to the variance of the measured times as illustrated in Fig. 13, where we plot the standard deviation for UDDI Search and WS Execution Times.

From the previous figures, it can be inferred that the UDDI Search times are highly unpredictable. The UDDI infrastructure may become saturated when many clients try to concurrently access it. When a bottleneck is formed, UDDI responses are returned to the querying clients in a rather sequential way. Hence, the UDDI infrastructure behaves as a traffic regulator for clients' queries addressed to the PDA-hosted web service. The workload pattern seen by the PDA-hosted WS is changed from a high number of concurrent requests (in case that the UDDI is lightly loaded) to bursts (of relatively long duration) of se-

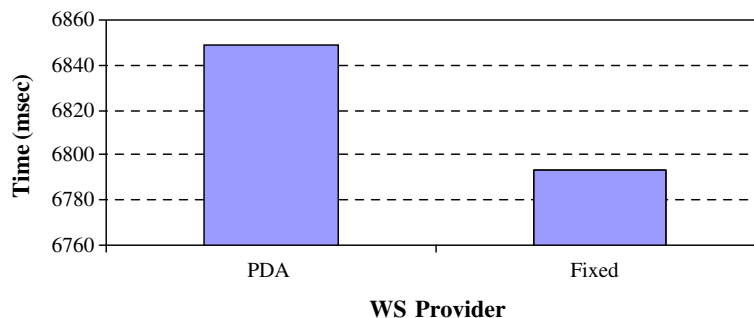


Fig. 11. UDDI registry update times.

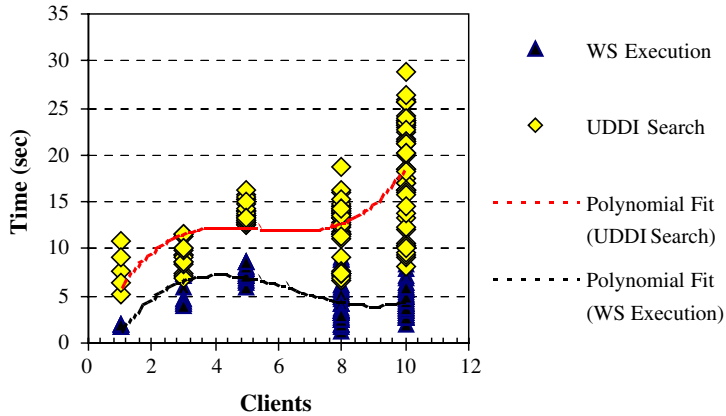


Fig. 12. UDDI search and WS execution times (PDA-hosted WS provider).

quential requests (in case of a heavily loaded UDDI registry which thus returns sequentially responses to its querying clients). Consequently, the WS Execution Time may be shorter in a scenario of highly loaded UDDI (which results in sequential WS requests), in relation to scenarios of a lightly loaded UDDI with very short search times which may result in many clients concurrently requesting the PDA-hosted WS.

In Fig. 14 we plot the times required for UDDI Search and WS Execution against the number of concurrent Clients accessing a WS Provider hosted by a standard desktop PC. We observe that the statistics collected through this setup demonstrate a quite similar behaviour (regarding the UDDI search times and WS execution Times) to that of the WH-WSP (Fig. 12).

In Fig. 15 we observe the difference in the average WS Execution Times between the PDA-hosted and

the desktop PC-hosted WS Provider configurations. Execution times experienced by WS clients are consistently lower in the desktop PC case, where computing power is significantly higher (AMD Athlon XP 2000+, 512 MB RAM) than that of the PDA (Intel StrongARM 32-bit RISC 206 MHz, 64 MB RAM). Execution times are also affected by the network bandwidth which is higher in the case of the fixed WS provider.

To summarize our findings, the WH-WSP scenario is totally feasible with existing software technologies. Continuous relocations (and address changes) of the mobile terminal necessitate continuous UDDI queries by the clients, hence, their performance is highly unpredictable as the UDDI registry becomes saturated. The time required to complete the WS invocation is consistently higher in the WH-WSP case than in the

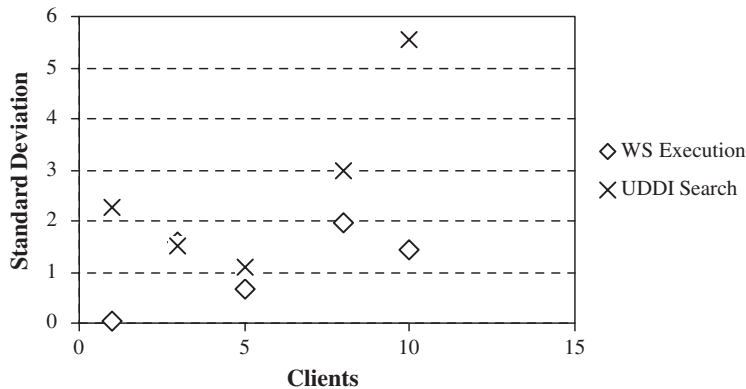


Fig. 13. Standard deviation for UDDI search and WS execution times (PDA-hosted WS provider).

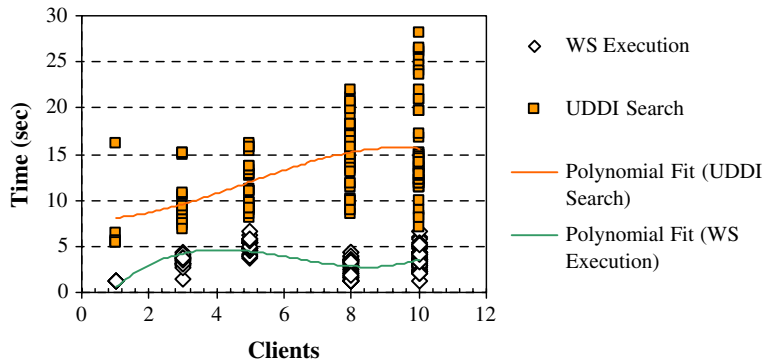


Fig. 14. UDDI search and WS execution times (PC-hosted WS provider).

fixed WS provider alternative due to limited processing power and bandwidth limitations.

### 8. Summary and conclusions

In this paper we have discussed the challenges related to the introduction of WS in the wireless/mobile domain. We argue that this introduction is extremely important to the mobile e-commerce (Internet e-commerce over mobile terminals) world. The expansion in the use of the WS technology in the wireless domain (mobile Internet) drastically broadens the scope of e-commerce (where WS is an established technology) to a much wider audience (as the number of users of fixed networks is practically equal to the number of users of their wireless counterparts). The WS technology is particularly important to the transaction services that are a crucial ingredient in the

mobile e-commerce landscape. Hence, businesses, organisations and individuals, sellers and buyers, are greatly facilitated in their tasks by a ubiquitous, WS-aware, e-commerce infrastructure where services can be published, queried and executed in a standard way. WS facilitate access to any data and service, on any device at any time. They also facilitate dynamic service selection by users of wireless devices rather than static provision which is the case with the services currently offered by various wireless operators.

In this paper we have presented some scenarios that uncover the benefits of the introduction of WS technology in the wireless world. In the first scenario, the wireless device assumes the role of the WS requester under two possible architectural configurations (thin/fat client) whereas in the second it plays the role of the WS provider. Selection of one of the three alternatives depends on the application requirements and the capabilities of the wireless device.

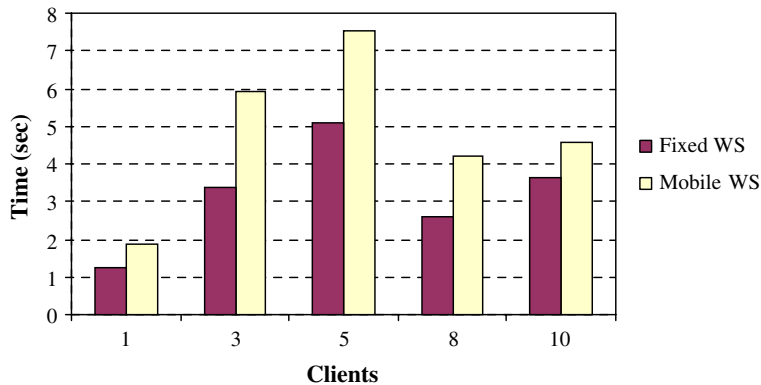


Fig. 15. WS execution times.

In the thin client scenario, the wireless device takes advantage of the WS technology indirectly through the use of a proxy server. This has the advantage of enjoying some of the benefits of the WS technology while at the same time keeping the wireless device free from the time- and resource-consuming WS tasks. On the other hand, thin clients do not support sophisticated application functionality and they are dependent on a central entity that controls all data and services accessible by them.

Fat clients provide more sophisticated support in terms of application specific functionality and exploit the strong points of WS (e.g. interoperability, dynamic service discovery and invocation). In terms of implementation, there are several preliminary attempts by key commercial players and research groups that reinforce the feasibility of this scenario. However, since most of the technologies supporting these scenarios are still or just emerging, many challenges prevail (e.g., performance, security and user interface issues) and need to be further investigated. In this respect, we propose a series of solutions such as message queuing and WWW-wireless optimisation mechanisms.

The scenario where the wireless device is acting as a WS provider is the most challenging as it gives pace to the introduction of the pervasive computing framework. A prototype implementation of this scenario has been developed to assess its feasibility and performance characteristics. Our findings show that the registry update/notification is the most important and time-consuming step in this scheme.

The solutions proposed in this paper for the various identified requirements have been consolidated in an architectural framework for supporting the users of Wireless Devices. We believe that this framework could be used as a reference or as a starting point for designing and implementing a wide range of wireless computing systems (e.g., m-commerce, context-aware applications, pervasive computing).

## References

- [1] Christoffer Andersson, GPRS and 3G Wireless Applications, Wiley, 2001.
- [2] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, S. Weerawanara, Unravelling the web services web, an introduction to SOAP, WSDL, UDDI, IEEE Internet Computing Journal (2002 (March–April)) 86–93.
- [3] D. Dahlem, J.H. Jahnke, A. Onabajo, O. Wang, The challenges of implementing web services on small devices, Proceedings of the Workshop on Pervasive Computing, Going Beyond Internet for Small Screens, OOPSLA 2002, Washington, USA, November 4th, 2002.
- [4] Thomas Erl, Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services, Prentice Hall, 2004 (April).
- [5] S. Hadjiefthymiades, L. Merakos, A survey of web architectures for wireless communication environments, Journal of Universal Computer Science, vol. 5, No. 7, Springer Verlag, 1999.
- [6] Uwe Hansmann, Lothar Merk, Martin Nicklous, Thomas Stober, Pervasive Computing, 2nd Edition, Springer, 2003.
- [7] B.C. Housel, G. Samaras, D.B. Lindquist, WebExpress: a client/intercept based system for optimizing web browsing in a wireless environment, MONET 3 (4) (1998) 419–431.
- [8] <http://www.ibm.com/services/uddi/testregistry/>.
- [9] [http://www.ikv.de/content/Produkte/enago\\_mobile\\_e.htm](http://www.ikv.de/content/Produkte/enago_mobile_e.htm).
- [10] Java Message Service, <http://java.sun.com/products/jms/>.
- [11] JSR 172, J2METM Web Services Specification, <http://www.jcp.org/en/jsr/detail?id=172>.
- [12] JUDDI, <http://sourceforge.net/projects/juddi/>.
- [13] kSOAP, <http://ksoap.org>.
- [14] kXML, <http://kxml.org/>.
- [15] NET Compact Framework, <http://msdn.microsoft.com/vstudio/device/compactfx.asp>.
- [16] Open Mobile Alliance, <http://www.openmobilealliance.com/>.
- [17] Helder Pinto, Noé Vilas Boas, Riu José, Using a private UDDI for publishing location-based information to mobile users, ICC/IFIP 7th International Conference on Electronic Publishing (EIPub2003), June 25–28, Guimarães, Portugal, 2003.
- [18] Pocket SOAP, <http://www.pocketsoap.com/>.
- [19] SOAP, <http://www.w3.org/TR/SOAP/>.
- [20] SOAP Version 1.2 Usage Scenarios, <http://www.w3.org/TR/2002/WD-xmlp-scenarios-20020626/>.
- [21] Softwired SOAP enable version 3.0 of its iBus/Mobile messaging middleware, <http://www.webservices.org/index.php/article/view/250/1/13/>.
- [22] M. Spanoudakis, S. Hadjiefthymiades, L. Merakos, A WWW system for wireless networks: design, implementation and performance assessment, Proceedings of IEEE SoftCOM 2001, Split Croatia, Ancona/Bari Italy, 2001 (October).
- [23] Gavin Stone, Mobile Execution Environment White Paper, MExE Forum, RONIN Wireless. (2000).
- [24] M. Tian, T. Voigt, T. Naumowicz, H. Ritter, J. Schiller, Performance considerations for mobile web services, Workshop on Applications and Services in Wireless Networks, Bern, CH, 2003 (July).
- [25] A. Tsalgatidou, T. Pilioura, An overview of standards and related technology in web services, International Journal of Distributed and Parallel Databases, Special Issue on E-Services, September, vol. 12, No. 2, 2002, pp. 135–162.
- [26] UDDI, <http://www.uddi.org>.
- [27] WSDL, <http://www.w3.org/TR/wsdl>.
- [28] XHTML 1.0 The Extensible HyperText Markup Language (2nd Ed.), W3C Recommendation, Aug. 2002.



**Thomi Pilioura** is a Ph.D. student in the Department of Informatics and Telecommunications at the University of Athens. Her research interests focus on service-oriented architectures, web service discovery and mobile commerce. She received her diploma in 1995 from the Department of Informatics and Telecommunications of the University of Athens and in 1998 her Masters degree from the Department of Computer Science of the University of Geneva

in the area of Information Systems. She works as a software engineer in the National Bank of Greece since 2000.



**S. Hadjiefthymiades** received his B.Sc. (honors) in Informatics from the Department of Informatics at the University of Athens, Greece, in 1993 and his M.Sc. (honors) in Informatics (Advanced information systems) from the same department in 1996. In 1999 he received his Ph.D. from the University of Athens (Department of Informatics and Telecommunications). In 2002 he received a joint engineering-economics M.Sc. degree from the National Technical

University of Athens. In 1992 he joined the Greek consulting firm Advanced Services Group, Ltd., where he was involved in telematics applications. In 1995 he became a member of the Communication Networks Laboratory (UoA-CNL) of the University of Athens. During the period September 2001 to July 2002, he served as a visiting assistant professor at the University of Aegean, Department of Information and Communication Systems Engineering. On the summer of 2002 he joined the faculty of the Hellenic Open University (Department of Informatics), Patras, Greece, as an assistant professor. Since December 2003, he is in the faculty of the Department of Informatics and Telecommunications, University of Athens, where he is presently an assistant professor and coordinator of the Pervasive Computing Research Group. He has participated in numerous projects realized in the context of EU programs (e.g., IST), EURESCOM projects, as well as national initiatives. His research interests are in the areas of web engineering, wireless/mobile computing, and networked multimedia applications. He is the author of over 80 publications in the above areas.



**Aphrodite Tsalgatidou** is a professor at the Department of Informatics of the National and Kapodistrian University of Athens (NKUA), Greece. She holds an M.Sc. and Ph.D. in Computer Science from the University of Manchester (UMIST), UK and a B.Sc. in Chemistry from NKUA. Her current scientific interests and research projects lie in the areas of business process modeling and reengineering, (mobile) electronic commerce, software architectures, peer-to-peer systems,

e-services and interoperability on the business, architectural and platform level. She has published several articles on these topics. Currently she is the Technical Manager of the project SODIUM which concerns unified publication and discovery of heterogeneous e-services. She has chaired two workshops on e-commerce and business process reengineering, she is the managing editor of an electronic journal and has served/serves on more than 30 scientific committees and on the advisory board for two international journals. More information can be found at her home page <http://www.di.uoa.gr/~afrodite>.



**Manos Spanoudakis** received his B.Sc. in Informatics and Telecommunications from the Department of Informatics and Telecommunication, University of Athens, Athens, Greece in 2000 and his M.Sc. (with honors) in Computer System Technologies from the same Department in 2003. He has performed research on WWW architectures for wireless/mobile systems and caching. Since 2000, he has been a member of the Communication Networks Laboratory of the University of Athens (UoA-CNL). His research interests are in the area of wireless/mobile computing, Web Caching and Content Distribution Networks. He is currently pursuing a Ph.D. in the above mentioned areas.