About an Architecture That Allows to Become a Mobile Web Service Provider

Marc Jansen
Computer Science Institute
University of Applied Sciences Ruhr West
Bottrop, Germany
marc.jansen@hs-ruhrwest.de

***Abstract*—The role of mobile devices as Web Service consumers is widely accepted and a large number of mobile applications already consume Web Services in order to fulfill their task. Nevertheless, no reasonable approach exists, as yet, to allow deploying Web Services on mobile devices and thus uses these kinds of devices as Web Service providers. This paper presents an approach that allows deploying Web Services on mobile devices by the usage of the well-known protocols and standards and, at the same time, can overcome problems that usually occur when mobile devices are used as service providers. Here, we provide both the description of an implementation with results of a first performance test. The test shows that the described approach provides a reasonable way to introduce Web Service provisioning for mobile devices.**

Keywords - mobile devices; Web Services; mobile Web Service provider.

I. Introduction

In recent years, the number of reasonably powerful mobile devices has much increased. According to [1], the number of smartphones worldwide counts about 300 million units.

On the other hand, this huge number of smartphones represents a large number of heterogeneous devices with respect to the operating systems smartphones are currently using. According to [2], there were at least five different operating systems for smartphones available on the market in 2010, and their distribution is shown in Fig. 1.

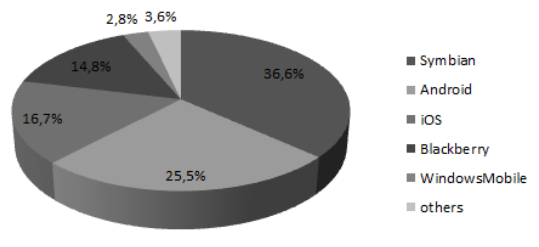

Figure 1. Distribution of different operating systems for smartphones in 2010

It thus seems to be necessary to have a platform-independent mechanism for the communication with services provided by smartphones in order to not re-implement each service for each of the mentioned operating systems.

Usually, Web Services are used in order to provide a standardized and widely used methodology that is capable of achieving a platform-independent way to provide services. Unfortunately, in contrast to consuming Web Services on mobile devices, providing Web Services on mobile devices is not yet standardized due to several problems that occur when a service runs on a mobile device.

This paper presents the description of a framework that allows providing Web Services on mobile devices. The outline of the paper is as follows: the next section provides an overview of related work, after which the scenario - together with the problems that usually occur should Web Services be provided by a mobile device - is explained. The following section explains the implementation of the framework in detail and the results of a first performance test are presented. The paper is closed by a conclusion.

II. State of the Art

The idea of providing Web Services on mobile devices was probably presented first by IBM [3]. This work presents a solution for a specific scenario where Web Services are hosted on mobile devices. More general approaches for providing Web Services on mobile devices are presented in [4] and [5]. In [6], another approach, focusing on the optimization of the HTTP protocol for mobile Web Services provisioning, is presented.

Importantly, none of the mentioned approaches manages to overcome certain limitations of mobile devices, as demonstrated in the next section.

The major difference between previous research and the approach presented in this paper is that, to the best of our knowledge, previous research focused very much on bringing Web Services to mobile devices by implementing server side functionality to the mobile device in question. The approach presented here follows a different line: from a technical and communication point of view, the mobile Web Service provider communicates as a Web Service client with a dynamically generated Web Service proxy.

This approach provides an advantage for overcoming certain problems with mobile Web Services as described in the next section. Furthermore, this approach does not rely on an efficient server side implementation of Web Services on the mobile device, and thus allows to implement a very lightweight substitution to a common application server where a common Web Service is running.

Since nothing comes for free, this approach has some drawbacks as well, e.g., it implements a polling mechanism that permanently polls for new service requests. Therefore, this approach produces an overhead with respect to the network communication and the computational power of the mobile device. The computational overhead, though, can be dramatically reduced by adjusting the priority of the polling mechanism according to the priority of the provided Web Service.

Another drawback of the presented approach is that it relies on a publicly available proxy infrastructure for the part of the framework that dynamically generates the Web Service proxies. This drawback can be overcome if, for example, mobile telecommunication companies provide this kind of infrastructure centrally.

In contrast to the before mentioned approaches, the approach presented in this paper differs with respect to one major aspect: from a network technical point of view there is no server instance installed on the mobile device. Therefore, a certain Web Service client does not call the Web Service on the mobile device directly but calls a centrally deployed proxy. The Web Service running on the mobile device polls in regular intervals for any new message requests of interest. The sequence of the Web Service request from the client point of view and from the Web Service point of view is shown in the sequence diagram in Fig. 2.

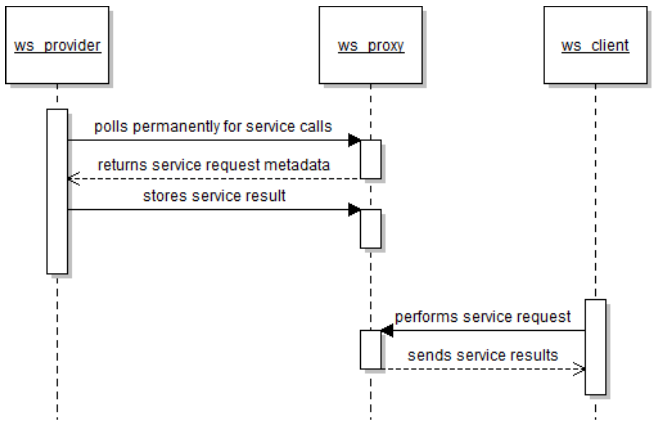

Figure 1: Sequence diagram of the Web Service calls in the presented approach

The exact sequence of the different messages and events will be described in more detail later. Since especially polling mechanisms cause a certain drawback, one of the major questions concerning the presented approach is the question of benefits and drawbacks of the polling mechanism and, in particular, whether the benefits justify the drawbacks.

As already mentioned, one of the major problems of dealing with Web Services on mobile devices is the fact that mobile devices often switch between networks. Therefore, the Web Service running on a mobile device is usually not available under a fixed address, a fact that leads to a number of problems for the consumer of a mobile Web Service: Besides the usual network switch, the fact that mobile devices are usually not meant to provide 24/7 availability, but are designed towards providing the user with the possibility to exploit certain services, e.g., phone calls, short messages, writing and receiving emails, etc., yields the problem that mobile devices might get switched off by the user. Hence, not only that the provided Web Service might be unavailable under different network addresses, but it might not be available at all.

All these drawbacks can be solved by using the approach presented here. By using the central proxy, the service requests of a certain Web Service client can be stored and if the mobile Web Service is running, it can pull for service requests that are of interest to it. Since from a technical point of view the Web Service provider only acts as a client to the Web Service proxy, the potentially changing network addresses of the mobile device do not pose a problem at all.

In addition, one of the major drawbacks of the described polling mechanism can be limited by adjusting the priority of the Web Service running on the mobile device, resulting in a lower frequency of the polling for the service request.

To conclude, in our opinion, the advantages of the described mechanism justify the drawbacks that are inherent to the approach.

III. Scenario Description

The major idea behind the implementation of the middleware is to provide a Web Service proxy, according to the proxy design pattern [7], in order to overcome certain problems in mobile scenarios as described by [8]. One major problem here is that mobile devices often switch networks, e.g., at home the mobile phone might be connected to a WiFi network, at work the connection might be established through another WiFi network and on the way home from work the mobile phone might be connected to a GPRS/UMTS-network. Each of these different networks provides different IP addresses and possibly different network constellations. For example, it can be private IP addresses with network address translation (NAT), where the Web Services running on the device are not directly accessible from the internet, or public IP addresses.

Frequently switching between IP addresses might raise certain problems for the provision of Web Services, since the client of a certain service always needs to know the actual IP address at which the service can be reached. More than that, within a private network the provided Web Services are usually not reachable at all from the internet.

Therefore, the problem, from the client point of view, is that the service is not always accessible under the same (and constant) IP address. The presented approach provides a solution to overcome this problem, with the exception of the case when a device is completely switched off. The switch off problem can be overcome as well, in which case slight modifications to the presented approach, together with an asynchronous call of the Web Service, are necessary.

The approach presented here suggests solving these problems by implementing a Web Service proxy that dynamically creates a proxy for each Web Service that gets deployed on a mobile device. The created proxy allows receiving service requests as a representative to the actual service and storing a service request along with the necessary data. In the next step, the mobile Web Service provider continuously polls for requests to its services, performs the services and sends the result back to the dynamically generated Web Service proxy. Receiving the result, the Web Service proxy can send the result back to the client that originally performed the service request.

IV. IMPLEMENTATION

The major goal of the work presented here is to provide a solution to the described scenario. Therefore, we implemented a middleware that allows the provision of Web Services on mobile devices. Here, the standard protocols, e.g., WSDL for the description of the Web Service interface, SOAP/REST as the standard network protocol and http as the usual transport protocol, are used such that there is no additional effort on the client side for requesting a mobile Web Service.

The following three sections provide a short introduction to the services offered by the middleware, followed by a description of the communication between the mobile Web Service provider and the Web Service client/consumer. Last but not least some details are presented about the Java based implementation for the test scenario.

A. Use-Case Analysis

In order to achieve the goal of implementing a Web Service proxy, an analysis of use-cases that this proxy will have to support has been performed. The result of this analysis is shown in Fig. 3.

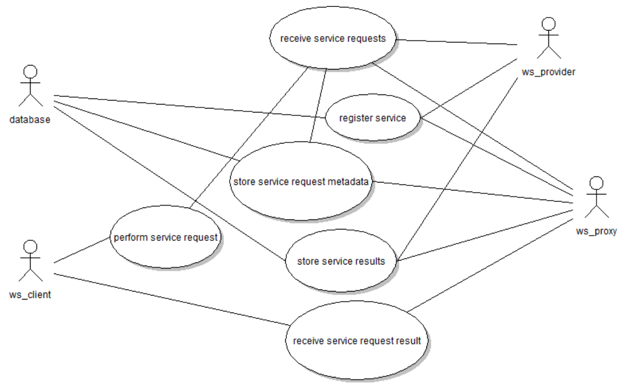

Figure 2: Use case description of the developed middleware.

From a technology point of view four different actors participate in the scenario. Obviously, a provider for the mobile Web Service is necessary. This is a piece of software running on the mobile device that provides the Web Service itself. This piece of software can best be compared with an application server hosting a Web Service in a scenario where the Web Service is provided by a common server system.

The second quite obvious actor is the consumer of the Web Service: the Web Service client. This is a piece of software running on the client side, performing requests to the Web Service.

As already described, one of the major ideas of the presented approach is to provide a proxy for the Web Services provided by the mobile devices. Therefore, the Web Service proxy is another actor that participates in the scenario. The proxy represents a surrogate of the Web Service provided by the mobile device. The basic function of this proxy is to implement the same interface (same methods with identical parameter lists and return values) as the Web Service itself. Moreover, the methods provided by the proxy (in order to register a service, de-register a service, etc.), should be accessible via the standard network protocols of Web Services and the description of the proxy interface should also be available in WSDL (in the implementation here the SOAP protocol was chosen). The proxy's' major task is to receive client requests, store them in a database and wait for the mobile Web Service to provide the result of the service request. While in the traditional proxy pattern the proxy would directly forward (push) the incoming service requests to the Web Service, we have decided to just store the requests in a database in order to allow the mobile Web Services to pull the requests from the proxy. This change to the traditional proxy pattern basically allows handling constantly changing network connections (as explained before), since within this approach neither the Web Service proxy nor the Web Service client need to know the actual IP address of the mobile device that provides the actual Web Service.

Fourth and last, the database is taken to be an actor of the middleware. Usually, the database would more likely be modeled as a system (and not as an actor), but for the sake of clarity and consistency, we decided to model the database also as an actor in the system. The major task of the database is to store the necessary information about the service request in order to allow the Web Service running on the mobile device to perform the requested task, and to later-on store the return values of the service request as well. By storing also the return value, the Web Service proxy is able to send the result back to the client that made the request. This is necessary since the usage of the proxy is transparent to the client, in the sense that the client is not aware that the actual service request is not answered by the proxy, but by the Web Service running on the mobile device. Therefore, the Web Service proxy needs to send the result of the service to the Web Service client, and not the mobile Web Service itself.

Besides the four actors, a number of use-cases need to be implemented in order to fully run the described scenario:

First of all, a mobile Web Service provider needs to be able to register a service to be provided. Besides the Web Service provider, the Web Service proxy and the database are interacting within this use-case, too. The Web Service proxy needs to dynamically implement the interface of the mobile Web Service and the storage of the metadata (basically the name of the method that should be called and its parameter values) of the service requests. The database needs to provide certain storage for the parameter values of each method (in case of a relational database: a table) and the according return values of the mobile Web Service.

The second, quite obvious, use-case is that the mobile Web Service provider needs to be able to receive service requests. Besides the mobile Web Service provider, the Web Service proxy participates in this use-case also, since this is the instance that directly receives the requests from the Web Service client and stores the necessary information in the database. Two additional use-cases, namely, perform service requests and receive service request results, participate in the store service request metadata use-case.

Additionally, we have identified two other use-cases that are necessary for the handling of the service request metadata (store service request metadata) and the handling of the return values (store service result). The first of these two use-cases interacts with two actors: the Web Service proxy and the database; the second one additionally interacts with the Web Service Provider.

Beside the fact that the provision of these use-cases allows the implementation of the described scenario, one of the major advantages of this approach is that the Web Service client only interacts with the preformed service request and receives corresponding answers from the service request result use-case. Therefore, from a client point of view, the request to a mobile Web Service is no more than a usual service request. No additional effort is necessary on the client side in order to receive results from a Web Service running on a mobile device.

B. Communication between the mobile Web Service and its clients

In order to explain the necessary communication for a service request from the Web Service client to the mobile Web Service provider, we modeled the communication flow within the sequence diagram shown in Fig. 4.

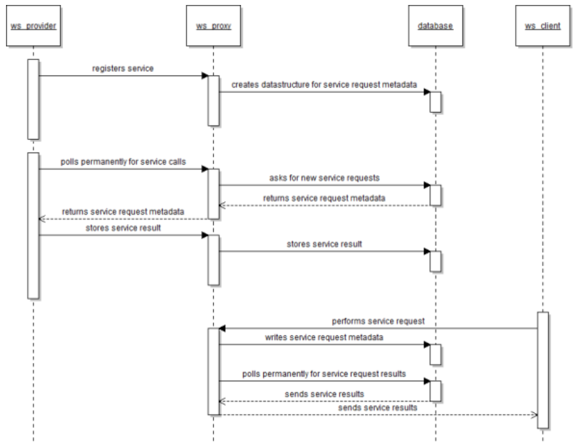
ws_provider
ws_proxy
database
ws_client
registers service
creates datastructure for service request metadata
polls permanently for service calls
asks for new service requests
returns service request metadata
returns service request metadata
stores service result
stores service result
performs service request
writes service request metadata
polls permanently for service request results
sends service results
sends service results

Figure 3: The UML sequence diagram for the communication between a Web Service provider and its client.

Within the sequence diagram we have modeled an object life line for each of the actors, to be discussed later. First of all, the mobile Web Service provider needs to register its service with the Web Service proxy. As part of the service registration process the Web Service proxy creates the necessary data structure for storing the service requests in the database.

After the mobile Web Service provider has registered its service, it permanently polls the Web Service proxy for new service requests. The Web Service proxy asks the database if a new service request for the respective mobile Web Service provider is available and if so, returns the request's metadata to the mobile Web Service provider. After receiving the metadata of a new service request, the mobile Web Service provider performs the service and sends the result of the service to the Web Service proxy that directly stores the result in the database.

From a client point of view, the Web Service client simply calls the service provided by the Web Service proxy. While receiving a new service request, the Web Service proxy stores the necessary request metadata in the database. Afterwards the Web Service proxy directly starts to permanently poll the database for the result of the respective service request. Once the mobile Web Service provider has finished performing the request and has stored the result (via the Web Service proxy) in the database, the Web Service provider is able to send the result of the service request back to the client.

C. A sample implementation

In order to test the described approach with respect to its performance, we implemented the Web Service proxy in Java. Additionally, the mobile Web Service provider was implemented for Android. Here, we focused on an intuitive and easy way for the implementation of the Web Service, and have therefore, oriented ourselves by the JAX-WS (Java API for XML-Based Web Services), as described in the Java Specification Request 224 (JSR 224). The major idea, adapted from JAX-WS, was that a Web Service can easily be implemented by the use of two different annotations: the @MobileWebService annotation marks a class as a Web Service, and methods within this class can be marked as methods available through the mobile Web Service with the @MobileWebMethod annotation.

With the help of these two annotations a simple mobile Web Service, which only calculates given integer values, can be implemented as follows:

```
@MobileWebService
public class TestService {

	@MobileWebMethod
	public int add(int a, int b) {
		return a + b;
	}

}
```

The basic relationships between the major classes of the sample implementation are shown in Fig. 5. For the sake of simplicity and transparency, less important classes (and methods of each class) have not been modeled.

Basically, the implementation consists of two packages. Package one is the proxy package which is usually deployed on a server that is reachable from the internet via a public IP address. Here, we find one class that implements the necessary methods for the registration of a new mobile Web Service, the permanent polling from the mobile Web Service for the service request metadata and the method that allows storing the result of the service request in the database. All these methods are reachable as Web Services themselves, so that the communication between the instance running the mobile Web Service and the Web Service proxy is completely Web Service-based.

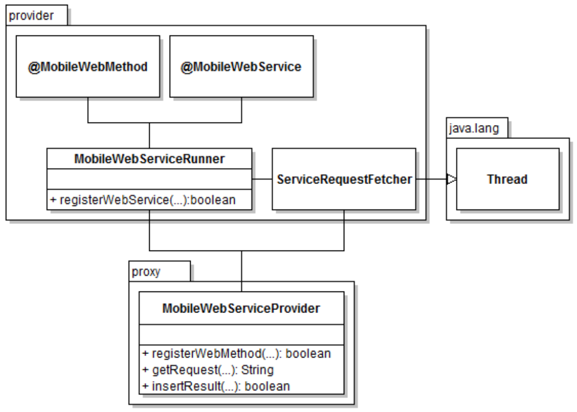

Figure 4: UML class diagram of major parts of the sample implementation

Basically, the implementation consists of two packages. One package that is usually deployed on a server that is reachable from the internet via a public IP address, this is the proxy package. Here, we find one class that implements the necessary methods for the registration of a new mobile Web Service, the permanent polling from the mobile Web Service for the service request metadata and the method that allows to store the result of the service request in the database. All of these methods are themselves reachable as Web Services, so that the communication between the instance running the mobile Web Service and the Web Service proxy is also completely Web Service based.

In the provider package we find, as one of the major classes, the MobileWebServiceRunner class to which the mobile Web Service gets deployed. This class is basically comparable to an application server in a common Web Service environment, but with a dramatically lower footprint. This lower footprint is extremely important to mobile devices due to their usually limited resources. Additionally, this package also provides the two formerly mentioned annotations that allow an easy marking of a class as a mobile Web Service and, accordingly, a certain method of such a class as a mobile Web Method. Last but not least, this package also implements the ServiceRequestFetcher class. This class inherits the java.lang.Thread class since its responsibility is to permanently poll the Web Service provider for new service requests.

V. PERFORMANCE TESTS

Since the communication is a little bit more complicated, in comparison to a common Web Service call, one concern of this approach is the question of its performance. In order to get a first idea of how good or bad this implementation behaves with respect to performance issues, we implemented a simple performance test.

A. *Description of the test scenario*

For the performance test we implemented a very simple mobile Web Service. This service only calculates the sum of two given integers and returns the respective value as the result. The major advantage of such a simple mobile Web Service is that almost the entire duration of the mobile Web Service call is dedicated to the communication, and almost no amount of the round-trip time is used for the calculation itself. Since the communication is the complex part of the presented approach, we assume that this method of performance testing would provide the best overview about the communication performance of the presented approach. In the test scenario a common client (running on a common PC) had to put a number of service requests to the mobile Web Service.

In order to compare the results against the performance of common Web Service calls, we implemented the test scenario also the other way around: we implemented a common Web Service (running on a common server) and called this Web Service from a mobile device. Here, the basic idea was to use the same hard- and software-environment with minimal changes and also to maintain the same network environments in all of the tests.

In addition, we were interested in the communication performance in different network settings. Therefore, we performed the same tests in four different network settings. For each of the tests the (mobile) Web Service and its consumer where running:

- … in the same (WiFi) network,
- … different networks, and the mobile device was connected via WiFi,
- … different networks, and the mobile device was connected via UMTS
- … different networks, and the mobile device was connected via GPRS

We conducted eight different test cases: four for the different network constellations with a mobile Web Service running on a mobile device and a Web Service client running on a common PC, and four test cases where the Web Service was running on a common Server and the client was running on a mobile device.

In the test cases where the (mobile) Web Service provider and the client were not connected to the same network, the central components have been deployed to a server running via Amazon Web Services (AWS), as a Cloud Computing provider.

B. *Test results*

Within each of these eight test cases, one hundred service calls were performed and the duration of each call was measured.

The results for the mobile Web Service in the different network scenarios are shown in Fig. 6.

As expected, the performance for the mobile Web Service calls was pretty good and pretty constant in the case the mobile device was connected with a WiFi network. If both the mobile Web Service provider and the client were connected to the same WiFi network, the average duration was M = 147.69ms (SD = 76.00ms). Having the mobile Web Service provider connected to a different WiFi network, the average duration for one service call was M = 339.04ms (SD = 61.71ms).

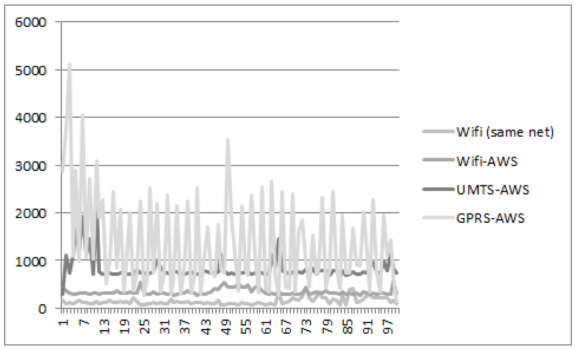

Figure 5: Results for the mobile Web Service in the different network constellations

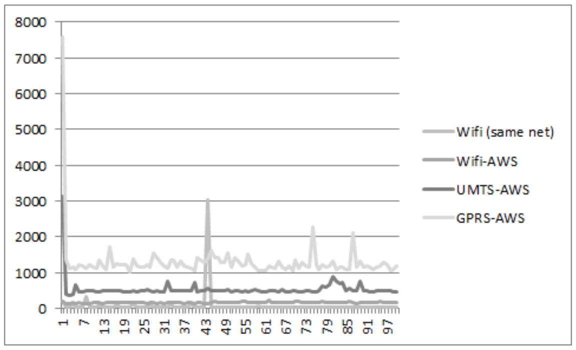

Figure 6: Results for the usual Web Service calls in the different network constellations

As expected the performance for the mobile Web Service calls are pretty good and pretty constant if the mobile device is connected with a WiFi network. The average time if both the mobile Web Service provider and the client are connected to the same WiFi network was M = 147.69ms (SD = 76.00ms). Having the mobile Web Service provider connected to a different, still WiFi, network the average time for one service call calculates to M = 339.04ms (SD = 61.71ms).

Of course, we measured less performance of the service calls when the mobile Web Service provider was connected to a mobile network, the performance of the service calls was lower. The results for the UMTS based network connection of the mobile Web Service show an average of M = 827.55ms (SD = 250.35ms) for each service call, while the results for the GPRS based network are even worse. Here, the average for a single service call is M = 1355.96ms (SD = 986.38ms). As can be seen from the values for the standard deviation, the performance of single service calls differs dramatically as well, e.g., the minimum duration measured within the UMTS scenario was MIN = 283ms and the maximum was MAX = 2169ms. The results for the GPRS based scenario are even worse, with a MIN = 142ms and MAX = 5123ms.

The task of the second step of the test was to compare the performance results with the performance of a common Web Service call. For that purpose we conducted the same test, but this time the Web Service was not running on a mobile device but on a common server, while the Web Service client was running on a mobile device - again in the four different network settings. The results of these tests are shown in Fig. 7.

As demonstrated, the results are better from both perspectives - the overall performance and the standard deviation in the different network settings. A common Web Service call, if the Web Service provider and the mobile Web Service consumer are connected to the same WiFi network, has an average round-trip duration of M = 61.16ms (SD = 301.36ms). When the Web Service client was connected to a different (still WiFi) network the average performance was M = 156.71ms (SD = 15.24ms).

Here, again, the values for the Web Service client connected to a mobile network are somewhat lower. In the case of the UMTS network, the average service call showed a performance of M = 528.55ms (SD = 273.34ms), and the results for the GPRS based network even worse with an average for each of the service calls of M = 1299.10ms (SD = 658.75ms).

The next step was to compare the different results. The major goal of this comparison was to get an idea of how good the performance of the presented approach for mobile Web Service calls is, in comparison to common Web Service calls. Therefore, we calculated the difference in the average performance of a single Web Service call in the different scenarios first, and as a second step calculated the percentage of the performance difference in the different scenarios. The results are shown in Table 1.

TABLE 1: COMPARISON OF THE COMMON WEB SERVICE CALLS AND THE MOBILE WEB SERVICE CALLS IN THE DIFFERENT NETWORK SCENARIOS

	WiFi (same net)	WiFi-AWS	UMTS-AWS	GPRS-AWS
difference	85.53ms	182.33ms	299.00ms	56.86ms
percentage	137.60%	116.35%	56.57%	4.38%

The table shows that, in comparison to common Web Service calls, the performance of the presented approach was not too good when the mobile Web Service was connected to a WiFi network. The results for the mobile Web Service provider and the client connected to the same network showed a performance overhead of 137.60 per cent, and when the mobile Web Service was provided within a different WiFi network the performance overhead was about 116.35 per cent. But, if the mobile Web Service was connected to a mobile network, the performance overhead was not that dramatic anymore. In the case of the UMTS network the overhead was limited to 56.57 per cent, and for the GPRS based network the overhead was even lower at 4.38 per cent. Therefore, on the basis of our test results, it can be said that the performance of the presented approach for mobile Web Services (in comparison to common Web Services) seems to improve the lower the network bandwidth is. This could best be seen by the results for the GPRS based network, where the actual overhead in our test was below 5 per cent.

VI. Conclusions

As demonstrated in this paper, today's modern and powerful mobile devices can be used as Web Service providers by using well-known and accepted standards and protocols. The presented approach is capable of solving some of the problems that usually occur while providing Web Services on mobile devices, e.g., the problem of constantly changing IP addresses. Furthermore, the overhead that is inherent in the presented approach does not seem to be a show stopper. As shown, the performance in commonly available mobile networks, like UMTS or GPRS, is comparable to common Web Service calls.

It can, therefore, be concluded that the presented approach provides an interesting alternative to the common Web Service provisioning by using mobile devices that act as a server also from a technical point of view. It eliminates certain problems that usually occur if mobile devices provide Web Service provider infrastructures, and the resulting drawbacks from the performance point of view are acceptable.

Having in mind the power that the presented approach would provide for new approaches and scenarios, it could be asserted that bringing Web Services to mobile devices will probably become more important in the future and that we will most likely see an increasing number of applications making use of that kind of technology.

Acknowledgment

This work was partly supported by an Amazon AWS research grant.

References

[1] IDC Worldwide Quarterly Mobile Phone Tracker, January 27, 2011.

[2] Tudor, B. and Pettey, C., 2010. Gartner Says Worldwide Mobile Phone Sales Grew 35 Percent in Third Quarter 2010, Smartphone Sales Increased 96 Percent, Gartner, http://www.gartner.com/it/page.jsp?id=1466313, last visited 19.11.2011

[3] McFaddin, S., Narayanaswami, C., and Raghunath, M., 2003. Web Services on Mobile Devices – Implementation and Experience, In: Proceedings of the 5th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA'03), pp. 100-109, Monterey, CA

[4] Srirama, S., Jarke, M., and Prinz, W., 2006. Mobile Web Service Provisioning, In: Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006), p. 120, Guadeloupe, French Caribbean

[5] AlShahwan, F. and Moessner, K., 2010. Providing SOAP Web Services and REST Web Services from Mobile Hosts, In: Fifth International Conference on Internet and Web Applications and Services (ICIW), pp. 174-179.

[6] Li, L. and Chou, W., 2011. COFOCUS – Compact and Expanded Restful Services for Mobile Environments, In: Proceedings of the 7th International Conference on Web Information Systems and Technologies, pp. 51-60, Noordwijkerhout, The Netherlands

[7] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., 1995. Design Pattern – Elements of Reusable Object-Oriented Software, pp. 185-195, Addison-Wesley.

[8] Svensson, D., 2009. Assemblies of Pervasive Services. Dept. of Computer Science, Institutional Repository – Lund University.