Evaluation of Distributed SOAP and RESTful Mobile Web Services

Feda AlShahwan
Centre for Communications Systems Research
University of Surrey
Surrey, UK
F. AlShahwan@surrey.ac.uk

Klaus Moessner
Centre for Communications Systems Research
University of Surrey
Surrey, UK
K. Moessner@surrey.ac.uk

Francois Carrez
Centre for Communications Systems Research
University of Surrey
Surrey, UK
F.Carrez @surrey.ac.uk

***Abstract*— Even mobile Web Services are still provided using servers that usually reside in the core networks. Main reason for not providing large and complex Web Services from resource limited mobile devices is not only the volatility of wireless connections and mobility of mobile hosts, but also, the often limited processing power. Offloading of some of the processing tasks is one step towards achieving optimal mobile Web Service provision. This paper presents two frameworks for providing distributed mobile Web Services: One mobile service provision framework is built on *Simple Object Access Protocol* (SOAP), while the other implements *Representational State Transfer* (REST) architecture. Both frameworks have been extended with offloading functionality and different types of resource intensive operations, i.e., process intensive and bandwidth intensive services, have been tested. The results show that using a REST-based framework leads of a better performing offloading behaviour, compared to SOAP-based mobile services. Distributed mobile services based on REST consume fewer resources and achieve better performance compared to SOAP based mobile services. The paper describes the approach, evaluation method and findings.**

Keywords-Mobile Web Services; REST; SOAP; Service Distribution.

I. INTRODUCTION AND MOTIVATION

Mobile Web services are self-contained modular applications that are defined, published and accessed across the Internet using standard protocols in a mobile communications environment. This technology has evolved from advances in the mobile device technology, rapid growth of Web Services development and progression of wireless communication in parallel with widespread use of Internet applications. However, it is still in its early stages and there are many challenges to overcome. Those challenges result from constraints in mobile resources, mobility issues and intermittent wireless network.

In literature, three different types of Mobile Web-services have been explored; they are characterized by the role acted by the mobile device when providing or consuming Web Services (see Fig. 1). These types include: **(Mobile) consumer, provider, and P2P Web Services**. In the mobile

Figure 1: Classification of Mobile Web Services

Web-service consumer case, mobile devices act as clients and request a service. In the provider case, mobile devices act as servers and provide them to any type of client. In the P2P case, mobile devices are connected in Ad hoc manner and each node may act as client or server, or both.

Most research into mobile Web Services has focused on consuming standard Web Services from mobile devices. However, the ubiquitous availability of mobile devices and their capability to provide information (e.g., Sensing information), or to provide complete/integrated services is a viable proposition. Hence, there is a need of exploring the provisioning of Web Services from mobile hosts. Our previous work [1] has investigated providing Web Services from mobile devices.

Hosting Web Services from mobile devices has an enormous number of useful real life applications. Location-based applications are an example of these useful applications. Location-based Web Services can be provided from mobile devices and have shown performance enhancement to companies who have employees deployed in the field. For example, a *Mobile Host* (MH) with a built-in *Global Positioning System* (GPS) receiver allows tracking of products and goods [2]. Health care applications are further evidence of the kind of applications provided by hosting Web Services from mobile devices. They might be useful for both doctors as well as patients. For example, deploying an appropriate service on a doctor's mobile allows tracking professionals' location and context to handle

emergency cases. Health care services can also be extended and provided from patients' mobile devices. This takes place by exposing a remote tele-monitoring service on the patient's MH [3] that allows monitoring their conditions using log files with the aid of some measurement devices such as a *Body Area Network* (BAN) sensor suite [4]. Not all location-based applications can be provided from the conventional fixed servers. This is because providing any location-based service is highly dependent on the actual current location of the service provider. For instance, providing the latest updated news and scene snapshots for a specific location in a predefined format requires portable devices with built-in GPS and cameras that are capable to move to the actual location of the event. Furthermore, it requires MHs that are aware of their location to publish the event as a live feed and takes latest information gathered at the current location. MHs allow processing of the gathered information and can then make it available, instantly to clients. Consequently, for the server, it may be more efficient in terms of cost and performance since it eliminates the need to upload the gathered location dependent information to static web server. Mobile devices are ubiquitous; they have small form factors, portable and almost anywhere accessible. As such, managing and maintaining handheld mobile hosts is easier, faster and more portable than static terminals. Moreover, mobile Web Services can be useful in polling-based applications that require using and triggering the most recent data, which is changing dynamically. Since checking an updated Really Simple Syndication (RSS) feeds through polling scheme requires exchanging a significant amount of information between each client and the standard fixed server. However, if the web server is a mobile device then the polling scheme is eliminated and substituted by sending a message from mobile host to all mobile clients when an update occurs. Context-based applications constitute another application discipline that benefits from hosting Web Services from mobile devices. Accessing the user profile of the mobile host and sharing the contents with others could be a useful application that allows clients to access the mobile host data contents, pictures and share the profile or modify it. The owner can also use web user interface of his mobile using standard desktop or laptop to get messages, information about incoming phone calls and phone book log when mobile host is currently unavailable or a better interface is required for accessing mobile contents. However, there are some issues related to the internal and external resource limitations of mobile hosts see Table1 that act as a barrier against the easy development of this area.

The motivation that leads towards this research is the large number of useful applications that can be provided from hosting services on resource constrained mobile devices. However, there are clear limitations in terms of complexity and size of the services that may be executed on mobile host.

TABLE 1. Internal and External mobile Constraints

Internal Constraints	External Constraints
• **Memory capacity, processing power and short battery life** • **Some data types that are defined with web services are not supported by the mobile devices** • **Most mobile devices support only short range wireless communication.**	• **Heterogeneity of the wireless environment** • **Limited bandwidth and large communication delay.** • **Frequent context and location change of mobile host** • **Mobile devices continuously need static IP address**

Our goal is to allow providing large and complex mobile Web Services continuously and without interfering with the main functionality of the mobile host that is making phone calls. Thus, lightweight processing and provisioning of mobile Web Services is needed to compensate for the limited resources of mobile hosts. This can be achieved through supporting automatic and autonomous self configuring distributed systems.

The technology used for developing Web Services can be classified into two main categories: *Representational State Transfer* (RESTful) and *Simple Object Access Protocol* (SOAP) Web Services. This classification is based on the architectural style used in the implementation technology. SOAP is an object-oriented technology that defines a standard protocol used for exchanging XML-based messages. It is defined as protocol specification for exchanging structured information in the implementation of Web Services in computer networks [5]. The specification defines an XML-based envelope for exchanging messages and the protocol defines a set of rules for converting platform specific data types into XML representations. REST is a resource oriented technology and it is defined by Fielding in [6] as an architectural style that consists of a set of design criteria that define the proper way for using web standards such as HTTP and URIs. Although REST is originally defined in the context of the Web, it is becoming a common implementation technology for developing Web Services. RESTful Web Services are implemented with Web standards (HTTP, XML and URI) and REST principles. REST principles include addressability, uniformity, connectivity and stateless. RESTful Web Services are based on uniform interface used to define specific operations that are operated on URL resources. Both SOAP and REST are used for implementing Web Services. However, each has its own distinct features and

shortcomings that make it more or less suitable for certain types of applications as shown in Table 2.

This paper is an extended version of [1] . It focuses on investigating mechanisms that facilitate distribution of provisioning and executing mobile Web Services. This can be accomplished through extending our previous SOAP- and REST-based *Mobile Host Web Service Frameworks* (MHWFs) that were implemented to deploy, execute and provide mobile Web Services. Our original implementation is extended in this paper to allow offloading of services and service fragments. In addition, this paper evaluates the performance and offloading overhead for both SOAP- and RESTful-based frameworks. This evaluation assists in selecting the framework that best suits mobile environment capabilities and fulfils our goal to provide mobile Web Services continuously with a light-weight processing requirement.

TABLE 2. Comparison of SOAP/ RESTful-based Web Services

Criteria	SOAP-based WS	RESTful-based WS
Server/ Client	Tightly coupled	Loosely coupled
URI	One URI representing the service endpoint	URI for each resource
Transport Layer Support	All	Only HTTP
Caching	Not Supported	Supported
Interface	Non Uniform Interface (WSDL)	Uniform Interface
Context aware	Client context aware of WS behaviour	Implicit Web Service behaviour
Data Types	Binary requires attachment parsing	Supports all data types directly
Method Information	Body Entity of HTTP	HTTP Method
Data Information	Body Entity of HTTP	HTTP URI
Describing Web Services	WSDL	WADL
Expandability	Not Expandable (No hyperlinks)	Expandable without creating new WS (using xlink)
Standards used	SOAP specific standards (WSDL ,UDDI, WS-Security)	Web standards (URL, HTTP methods, XML, MIME Types)
Security/Confidentiality	WS-security standard specification	HTTP Security

The rest of the paper is organized as follows: Section II presents a short introduction to the current state of art for providing Web Services from mobile devices and highlights the main issues encountered when distributing mobile Web Services. Section III describes the main modules that are used for building standard SOAP and RESTful mobile services. Section IV presents an evaluation between SOAP and RESTful MHWFs in non-offloading environment. Section V explores some distribution mechanisms that allow reliable and light weight provisioning of complex mobile Web Services and outlines different types of offloading mechanism. Section VI describes our architecture and implementation that supports provisioning of distributed mobile services. Section VII introduces a critical analysis between the two extended frameworks (i.e., the SOAP and REST MHWFs) in handling offloading strategies for different types of resource intensive applications. Some of their features and issues are also addressed in this section. Finally, conclusions from this work are presented in the last section along with recommendations for some future work.

II. STATE OF THE ART

There has been extensive research into the development of MHWFs. Most of the implemented frameworks allow deploying and providing SOAP-based mobile Web Services either in a client / server environment [7-9] or in a P2P network [10-11]. Some researchers have focused on applying mechanisms that allow adaptation and compensation for the lack of resources. For example [12] proposed a partitioning technique to the layered MHWF approach [13] that allows the execution of complex large Web Services on mobile hosts. However, in this approach clients send requests first to a stationary intermediate node, which contradicts an essential mobility requirement of mobile Web Service hosts.

Furthermore, this approach relies only on SOAP-based Web Services that require heavy weight parsers and large message payloads. Consequently the overall MH performance is degraded. The Modular Hosting Web Services architecture [14] contains built-in modules to support continuous provisioning of mobile Web Services in P2P network environment. This is accomplished through migrating services to another surrogate mobile node when the mobile host becomes inaccessible due to location changes or drained battery power. However, this framework provides only SOAP-based simple Web Services and does not allow light weight processing of complex services. Recent research studies focus on building resource aware mobile Web Service provisioning architecture that supports RESTful-based mobile Web Services. An evaluation of RESTful Web Services that are consumed from mobile devices is presented in [15], however, this evaluation is constrained to mobile Web Service consumers and does not include mobile Web Service providers. The concept of REST-based *Mobile Web Services* (MobWS) is introduced in [16] and a comparison with SOAP architecture in terms

of HTTP payload is carried out in [17] but the implementation of a mobile host that provides RESTful Web Service is not addressed. Providing adaptive mobile Web Services and testing REST for distributed environment are also not tackled. RESTful-based mobile Web Service framework is proposed for the first time in [1] and a detailed comparison is carried out between SOAP- and RESTful-based MHWFs and analyzed. The evaluation involves performance, resource consumptions and scalability. The analyzed preliminary results showed that RESTful-based MHWF is a promising technology that is more suitable for limited resource mobile network environments. However, the proposed frameworks have not address the provisioning of complex mobile Web Services. Mobile Web Service distribution is acquired for executing complex and large applications to lessen the burden on mobile host and preserve its resources and energy consumption [18].

In contrast to the approaches described above for providing mobile Web Services from mobile hosts, we aim to allow light weight provisioning of mobile Web Services, reduce mobile host energy usage and increase scalability and throughput. This aim can be achieved through distributing the execution of mobile Web Services for both SOAP and RESTful-based MHWFs and comparing them to each other. This comparison is needed to allow us to define the most suitable framework for distributing the execution of complex mobile Web Services. The selection criteria used for comparison are based on minimizing the offloading overheads and increasing overall performance.

III. System Architecture

Web Services are not explicitly defined for the mobile wireless environment. The current standard Web Service frameworks are developed for static servers. In addition, these standard frameworks are too large to be deployed on resource constraint mobile devices and they require a running time environment that is not available on mobile devices. Also providing Web Services from mobile hosts consumes a large amount of resources and drains the batteries within a short period of time. Thus, providing Web Services from mobile devices requires building a dedicated framework for deploying, providing and executing Web Services. In our previous work [1] we developed two different frameworks. One supports RESTful-based mobile Web Services that is built for the first time up to our extent knowledge and the other supports SOAP-based mobile Web Services. In implementing our framework, Java for Mobile Edition JME is used as the best language for launching applications on limited resource mobile devices. JME defines two configurations: the *Connected Device Configuration* (CDC) and the *Connected Limited Device Configuration* (CLDC). In this research CLDC has been selected because it is a low-level specification, suitable for wide range of mobile devices with limited memory capacity. Thus, CLDC achieves scalability and generality. APIs and libraries are added to support more features *through Mobile Information Device Profile* (MIDP). In this research MIDP 2.0 is chosen because it supports devices with limited network communication resources and device internal resources. Also it provides more networking functionality and it supports HTTP protocol. In addition, it supports the Server Socket connection that is required for implementing mobile server. In general the execution model and the architecture of the two frameworks are identical MHWF. The architecture is presented in Fig. 2.

The model consists of five main building blocks:

1. Web ServiceServlet
2. HTTP Listener
3. Request Handler
4. Parser Module
5. Response Composer

Although the overall architecture of SOAP and RESTful-based MHWF is similar, they differ in the details for handling and parsing the request. For example, in SOAP-based MHWF the Request Handler will un-wrap the incoming HTTP POST request to extract the hidden SOAP envelope then it will dispatch the envelope to the message parser. On the other hand the request handler for RESTful-based MHWF will extract the HTTP request directly and send it to the Message Parser Module. The main function for the Parser Module is to get the needed information for invoking a Web Service such as the name of the service, service URL and some parameters. Then the extracted information is sent to the Service Servlet. However, the way this is performed is different between the two frameworks. In SOAP-based MHWF, the SOAP parser de-serializes the SOAP object and maps the data types into Java objects using kSOAP2 and kXML2 that are open source APIs for SOAP parsing. However, in RESTful MHWF we have created our own String Manipulator -based parser. This parser will extract the server name and the parameters that are required for executing this service.

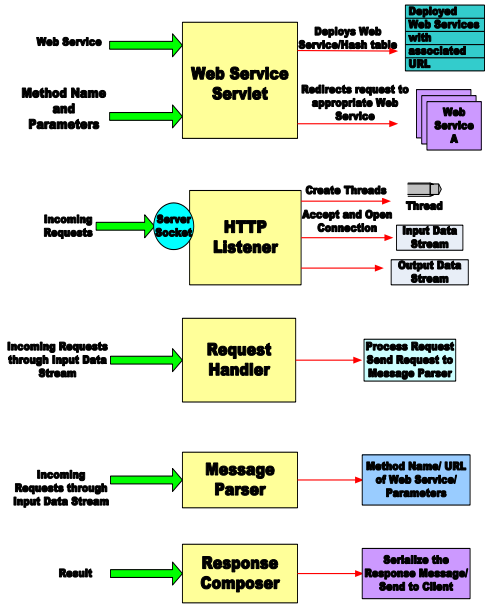

Figure 2: Architecture of Mobile Web Service Framework

The next section introduces an analytical and experimental analysis between the two SOAP and RESTful architectures in non-offloading environment.

IV. NON-OFFLOADING EXPERIMENTS AND RESULTS

On a first claim the difference between the two previously implemented architectures are fairly similar and there is no apparent difference in complexity but the major different comes when we have tested the architectures' performance, scalability and amount of resource consumption.

The evaluation is conducted using a small test-bed that consists of a mobile host developed on N80 Nokia mobile device running Symbian OS, MIDP 2.0 profile. It is connected in a wireless network through built-in IEEE 802.11b interface and it provides services to a client that is simulated using Sun Wireless Toolkits 2.5.2 emulator. The evaluation involves three different scenarios. The first set of experiments is done to test the performance of the mobile host. Performance is analyzed through measuring the effect of varying the request message size on the average processing time. Results in Fig. 3 and Fig. 4 show that the average processing time increases when the request message size increases.

Moreover, the average processing time for SOAP-based MHWF is larger than the average processing time for RESTful-based MHWF for the same message request. This is because processing SOAP requests requires heavy weight parsers to un-wrap the SOAP envelope from the incoming

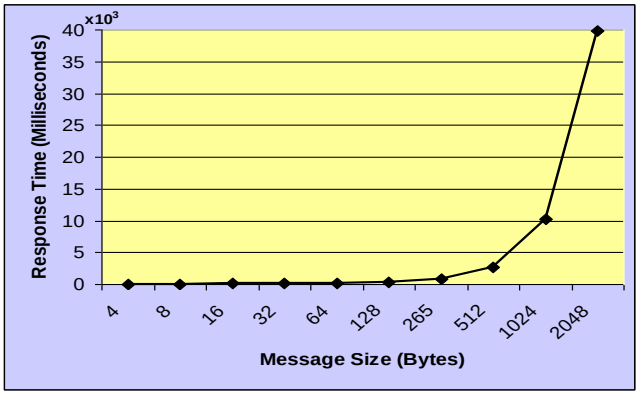

Figure 3: Effect of message size on process time of SOAP-based MHWF

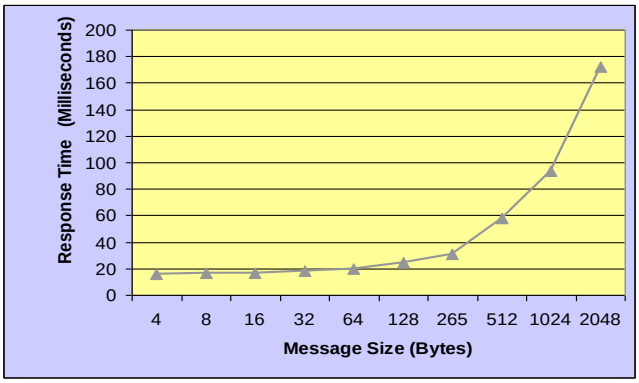

Figure 4: Effect of message size on process time of RESTful-based MHWF

HTTP POST request, then de-serialize the SOAP object and map the data types of the XML-based message into Java objects. This is done to extract the hidden information needed for invoking the required Web Service. However, processing RESTful requests uses light weight parser that is created by us to extract the information required for invoking the designated Web Service. Moreover, the required information resides explicitly on the HTTP request. Thus, RESTful-based MHWF has better performance than SOAP-based framework.

The second scenario evaluates reliability and scalability of the frameworks. This evaluation is carried out by testing concurrency where a number of clients send requests to the same host simultaneously. Concurrency is accomplished through initiating threads and loops on the client emulator. Then the average process time for each concurrent request is calculated. Results Fig. 5 and Fig. 6 show that as the number of concurrent requests increases, the average process time also increases. This increase is more obvious in SOAP-based framework where more time is consumed to parse the SOAP envelope and to manage the threads. However, we observe that the increase in RESTful-based MHWF is almost steady. This is because RESTful Web Services support caching and demand light processes power. Hence, RESTful-based MHWF is more rigid and robust to changes in the number of concurrent requests.

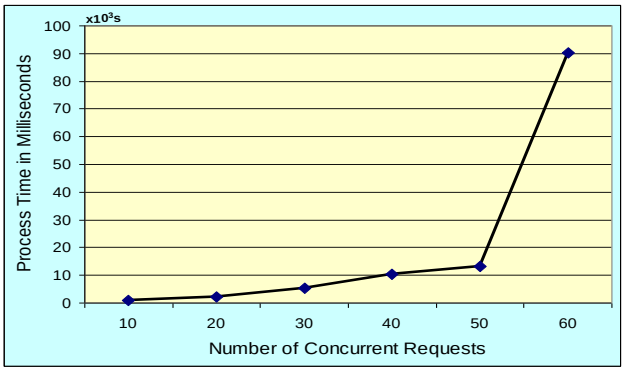

Figure 5: Effect of Concurrent requests on process time for SOAP-based MHWF

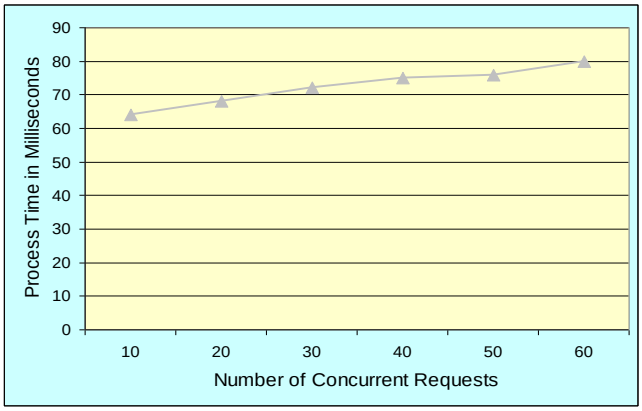

Figure 6: Effect of Concurrent requests on process time for RESTful-based MHWF

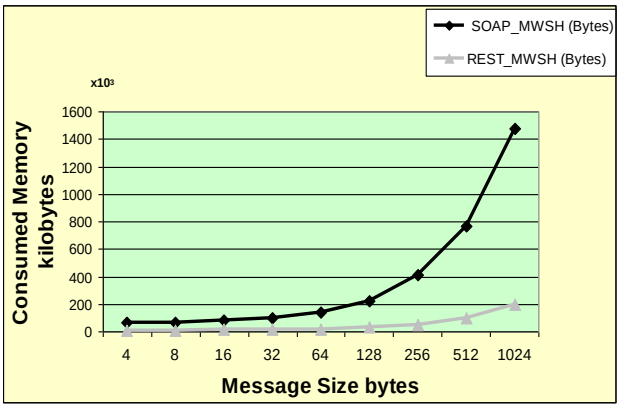

Figure 7: Comparison of consumed amount of memory between SOAP and RESTful-based MHWFs

After that, the two MHs are stressed by adding more concurrent requests to measure the threshold value. The threshold value is defined as the maximum number of concurrent requests that can be handled without failure. It is observed that in Table 3 SOAP-based MHWF starts to reject requests earlier when the threshold is beyond 60 but RESTful-based MHWF starts to reject requests when the threshold is beyond 80. This is expected because processing SOAP-based requests requires more time. Consequently, the consumed response time is larger and the server queue of the SOAP-based framework will be occupied and filled within a short period of time. As a result, there will be no more resources to accept new connections. Thus, RESTful-based MHWF is more scalable and reliable than SOAP-based MHWF.

The last scenario is for testing resource consumption and measuring memory footprints. Results in Fig. 7 illustrate that the amount of consumed memory during processing Web Service requests is increased as the message size increases. As shown in the graph the amount of consumed memory in SOAP-based framework is larger than the amount of consumed memory in RESTful-based framework for the same message size. The reason for this is that SOAP-based framework demands more memory footprint during processing. This consumed memory footprint is used to store general temporary parsed objects and to load the classes, kSOAP and kXML libraries.

TABLE3. Comparison of rejected requests between SOAP-based and RESTful-based MHWFs

No of Requests	Average rejected requests (SOAP)	Average rejected requests (REST)
60	10	0
80	59	4
100	64	9
120	86	14

V. MOBILE WEB SERVICE DISTRIBUTION

The purpose of this research as mentioned before is to investigate, define and provide mechanisms that will facilitate continuous provisioning of complex services in a light-weight processing power with efficient levels of performance. This is achieved through distribution of mobile Web Services. There are some factors that necessitate Web Service distribution in mobile environments. An important issue relates to the enormous spreading of distributed computing systems in a *Peer to Peer* (P2P) network. In P2P networks, nodes are both providers and consumers. P2P networks have some advantages that make it outperform its corresponding typical client/server networks. Avoiding single point of failure and increasing system capacity are some of these advantages. Since P2P is increasingly evolving, therefore, the application of distributed mobile Web Services executed and deployed in a distributed network environment is an important direction for future research.

Moreover, distributing Web Services is done to lighten the processing weight on limited resource mobile web servers. In spite of the fact that these constraints may be eliminated in the future and the resource capabilities might advance, the ideal performance and the minimum latency will always be the dominant requirements. In addition, resource limitations will still exist as user demands increase. For example, the memory capacity of mobile devices will continue to increase but memory limitation occurs when user wants to run multiple services or multiple instances of the same service on the MH. Furthermore, battery life will, for the foreseeable future, remain a bottleneck. Hence, the distribution of mobile Web Services results in preserving energy resources, scalability increase and an overall performance enhancement. It should also be noted that running complex large Web Services on an overloaded MH requires large processing power and might affect its core functionality.The first step for distributing Web Services is to define criteria for triggering distribution, in our case this has been done using Fuzzy Logic, however, this and the

resource monitoring system are beyond the scope of this paper. The next step is to partition the execution tasks of a Web Service and execute partitions on different remote machines. This mechanism is called *offloading*.

We have defined different schemes for applying the offloading mechanism in mobile network environment. The main difference between these schemes is the methodology used by the mobile host for handling requests and responses. The first scheme is called Forward-Offload and shown in Fig. 8. In Forward-Offload a client sends a request to the MH then it forwards the request to an AMH for processing. After that, the AMH sends its response to the MH, which forwards the response to the client. This type of communication relies on the MH to partially process the request, select the AMH and to maintain communication subsystem TCP. However, it supports ubiquitous computing through distributing the execution autonomously without the client being aware.

The second case is called Bounce-Offload. Fig. 9 illustrates Bounce-Offload where the client sends a request to the MH, which then bounces the request back to the client, redirecting the request to another host for processing. This type of communication lessens the load on MH, preserves its resources and reduces the signaling exchanges (compared to Forward-Offloading). Thus, it increases the capability for the mobile host to handle more requests concurrently and increases scalability. However, these benefits are gained at the expense of putting a greater burden on the client to tackle the task of contacting another host. The critical analysis between the two offloading strategies has been carried out by us and will be published in another paper.

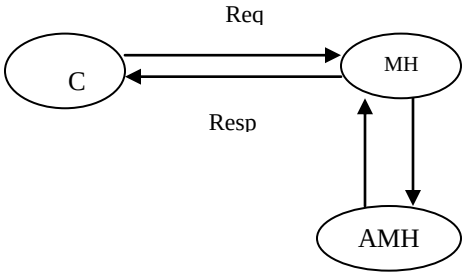

Figure 8: Forward-Offload

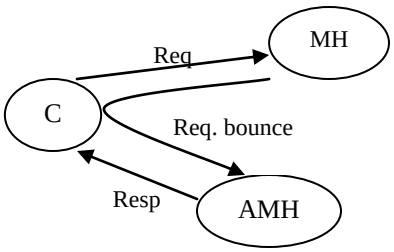

Figure 9: Bounce-Offload

In this publication, Forward-Offload is examined to support ubiquity and autonomy. However, this scheme consumes more resources than Bounce-Offload. Thus, our aim is to minimize resource consumption as much as possible. This goal can be achieved through a coherent study of the signaling and processing overheads for both extended SOAP- and REST- based MHWFs. The next section explains and illustrates the architecture of the aforementioned extended MHWFs.

VI. Mobile Web Service Distribution Architecture

The MHWFs architecture that has been implemented previously [1] for providing, deploying and executing SOAP and RESTful- based mobile Web Services is extended to allow distribution and offloading functionality. This is accomplished by using the previously implemented architecture for developing the AMH. The AMH will take the role of a mobile host temporary and performs its typical tasks such as handling the forwarded requests, invoking the required service, executing it and sending the result back to the MH. However, the architecture of the mobile host is an augmentation of the basic built MHWFs. The augmentation is taken place through adding an Offloading Module as shown in Fig. 10. The main task of the Offloading Module is to transform the role acted by the MH from server to client temporary. This is carried out to allow MH to forward incoming requests to AMH. MH partially processes incoming requests to extract the name of the requested Web Service and its associated parameters. Another important task for MH is to select the appropriate AMH that satisfies some predefined conditions. The following section introduces the prototype that is used for testing and examining the validity of distributing SOAP and RESTful-based Web Services in mobile environments.

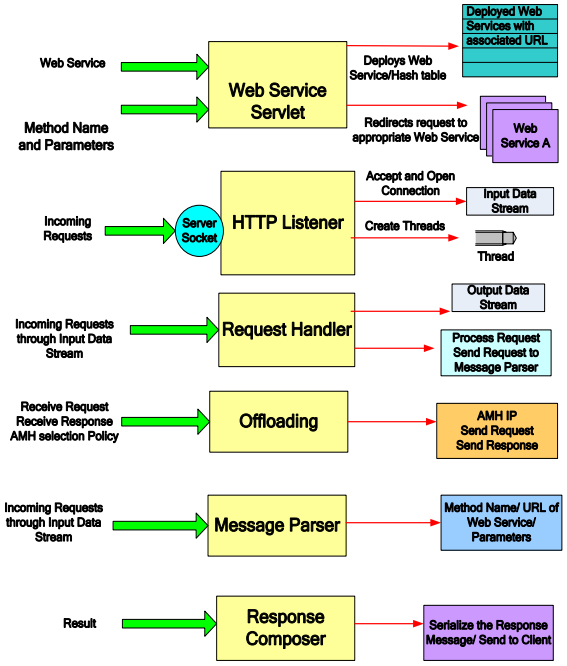

Figure 10: Architecture of MHWF with offloading functionality

VII. EXPERIMENTAL RESULTS AND EVALUATION

As aforementioned, the main objective is to investigate the offloading mechanisms and to examine the feasibility and validity of distributing SOAP and RESTful based-Web Services in mobile environments. Another objective is to test and compare two different architectures to assist in selecting an architecture that is most suitable for distributing mobile Web Services with fewer overheads and less resource consumption.

The experimental approach we followed evaluates functional and non functional properties in two different environments: offloading and non-offloading environments. It also applies two different resource intensive applications for each environment: processing and bandwidth intensive application types. Tests for non-offloading environment have been carried out in the previous section. Following is a description of the test taken for offloading environment.

A. *Offloading Experimental Environment*

A small prototype is proposed to carry out the experiments needed to address the validity of offloading mobile Web Services and distributing the execution tasks of a large complex Web Service between different mobile hosts. We have extended the two architectures for the main MH by adding an Offloading Module and using the same previous MHWF architectures for the AMH. The evaluation was conducted using a prototype compromising three mobile devices as shown in Fig. 11: The MH is executed on a mobile device (Nokia N97m) running MIDP 2.1 over Symbian OS. The other device, implementing the auxiliary AMH that acts as mobile host when the original MH is

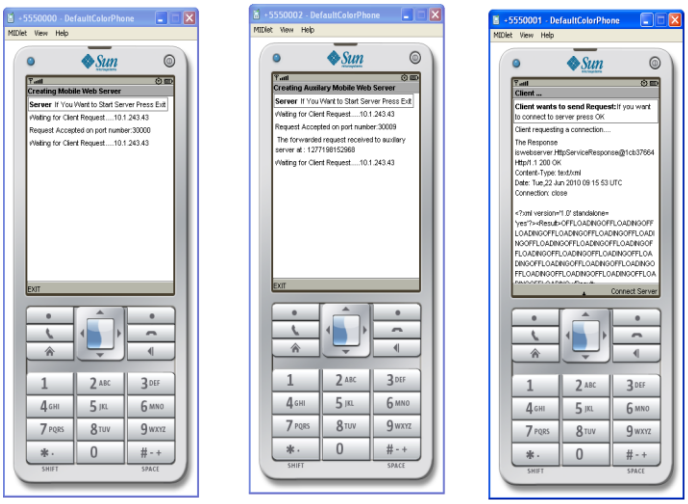

Figure 11: Prototype for offloading mobile Web Services

overloaded, was executed also on an N97m. The client was executed on a Laptop using the Sun Wireless Toolkit and emulator. The devices were connected via a wireless network. In this experiment Forward-Offload strategy has been applied. Since the MH is assumed to be overloaded it processes part of the incoming requests and forwards it to AMH. The MH elects an AMH.

The election is carried out using probe requests sent to all mobile devices that satisfy set of predefined criteria. However, this is beyond the scope of this paper. The evaluation has been accomplished for two different services. The first Web Service represents processing intensive application. The example used for this type of applications was a simple PI calculation service. In this service the accuracy for calculating PI depends on the number of terms that are added together. The number of terms is controlled by a client using an integer parameter. The other type of services represents bandwidth intensive application. The service used for bandwidth intensive applications was a simple String-Concatenation. In this service, the number of times constant is merged and concatenated depends on a parameter (i.e., an integer value) set by the client.

The evaluation for both services is carried out using three different scenarios. In the first set of experiments the level of internal resource consumption is examined including both memory and processor resources. In the second set of experiments the level of external resource consumption is estimated by calculating the total amount of interactions between the three connected mobile devices. In the third set of experiments the overall performance is evaluated by measuring total elapsed response time for execution of each request. After that, the offloading overhead is analyzed. Finally, the performance improvement is evaluated for both (SOAP and REST) architectures in the last set of experiments.

B. Results for Offloading Process Intensive Web Service

The first application scenario demands intensive processing power. The application represents a simple mathematical service called PI Web Service used to calculate the constant π whose value can be approximated using Gregory-Leibniz series [19]:

$$\pi = 4 * \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}$$

We used different values of *k* in our experiments to vary the computational intensity of the Web Service sample. PI is a suitable service for accomplishing the required tests. This is because it represents intensive power applications where the amount of consumed power can be controlled via *k* parameter, which determines the number of accumulated terms. First, the amount of internal resources consumption is examined for different values of *k* to investigate the effect of varying application process complexity level on the MH resources. These internal resources include both MH memory and MH processing power that are required during executing and offloading incoming Web Service requests. Tests run for both architectures RESTful and SOAP-based MHWF. The memory consumption is averaged for 50 requests for different values of *k*. Memory is estimated by calculating the difference between the total available amount of memory on MH before processing incoming requests and the available memory after processing requests before sending them to clients. However, since the heap memory size of mobile devices is variable, then a technique for controlling the variation of mobile host memory is applied. This is done by releasing the unused objects then freeing the memory heap by running garbage collection before measuring the total available memory amount. Results presented in Fig. 12 show that with offloading, changing the application processing complexity has no effect on the memory consumption amount for the main mobile host. This is because the real processing and memory allocation are delegated to another auxiliary mobile host. Moreover, RESTful-based architecture saves more memory resources than the conventional SOAP-based architecture. The average amount of CPU processing power is also tested for different values of *k*. In general the amount of CPU processing power can be estimated by measuring the processing time required to execute a predefined task by the CPU. In the offloading process the MH processing time includes two parameters they are: the time required to process incoming requests from clients and the time required to process incoming responses from the AMH. Thus, the average processing time is the summation of the average time spent for client requests in MH before it being forwarded to the AMH plus the average time spent for responses that are delivered from AMH to MH before it being forwarded to designated client. This average process

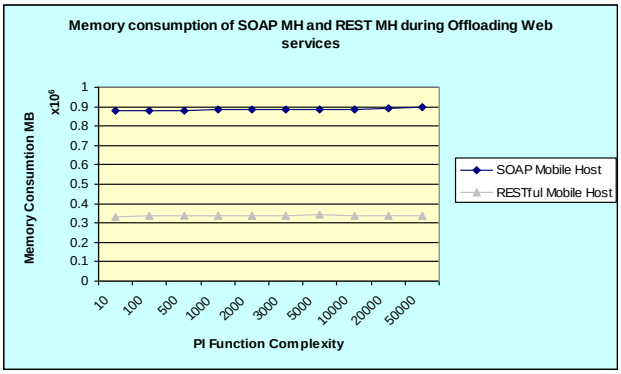

Figure 12: Memory Consumption of SOAP and REST mobile hosts

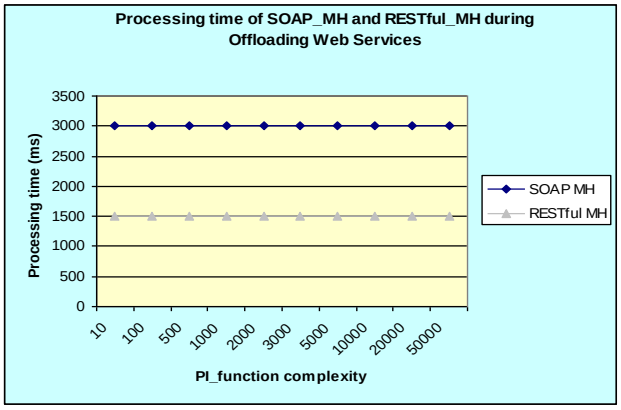

Figure 13: Processing time for SOAP and REST mobile hosts

time is measured for different *k* values. Fig. 13 illustrates that the average processing time required by MH is constant since it compromises the process of parsing requests and responses that have invariable payload length. On the other hand invoking and executing the required service that has variable complexity takes place remotely on AMH. Moreover, SOAP-based MHWF demands larger processing power than its corresponding RESTful-based MHWF. This is because processing SOAP requests requires heavy weight parsers to un-wrap the SOAP envelope from the incoming HTTP POST request. However, processing RESTful requests uses a light-weight String-based parser that is created by us to extract the information, which resides explicitly on the HTTP request. Thus, RESTful-based MHWF consumes fewer amounts of internal resources than SOAP-based framework. This preserves more resources for the MH to allow it to handle more requests and deploy more active Web Services. Consequently RESTful-based MHWF increases scalability and throughput in distributed mobile Web Service environment.

Second, the level of external resource consumption is tested for different values of *k*. Bandwidth consumption is one of the most critical external resources in mobile wireless environment. This resource is predicted through computing the total amount of data transferred in a predetermined amount of time, which mainly depends on the size of both request and its corresponding response.

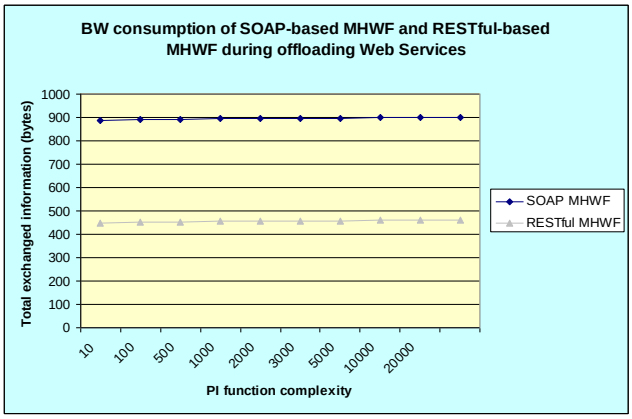

Figure 14: Bandwidth Consumption for SOAP and REST-based MHWF

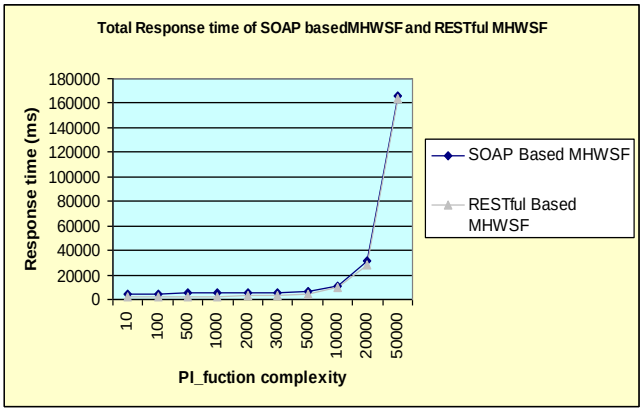

Figure 15: Total response time for SOAP and RESTful-based Web Services

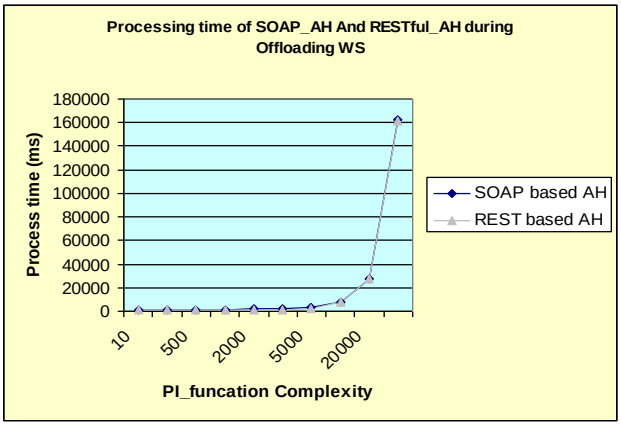

Figure 16: Processing time on Auxiliary Mobile Host for SOAP and RESTful-based MHWF

For simplicity we used the average total amount of interactions between the three mobile nodes (client, MH and AMH). With respect to Fig. 14, it is shown that RESTful-based MHWF outperforms the standard SOAP-based MHWF and contains approximately 50% less amount of data exchanged. This result is expected because SOAP messages are verbose XML and they require an envelope to hide the service name and parameters in the body of the HTTP request. However, RESTful-based messages are based on the standard HTTP and the service name with its associated parameters are explicitly reside in the HTTP URL. Hence, RESTful-based MHWF requires less bandwidth than SOAP-based MHWF.

Finally the average response time is measured for different *k* values and for both architectures. Response time is defined as the time that a client spends waiting to receive the result from the MH. This is measured by calculating the difference between the time when a response is received by the client from the MH and the time when a request is sent by the client to the MH. Results presented in Fig. 15 show that the average response time is directly proportional to the complexity degree of the application being processed. The proportional relation refers to the two parameters that dominate the response time value: communication delay and the processing time on both MH and AMH. Although the processing time on MH is constant and does not change with different *k* values, the processing time on AMH as shown in Fig. 16 is variable and it increases for larger values of *k*. Moreover, SOAP-based MHWF requires more response time than RESTful MHWF for the same *k* value. This is because SOAP-based MHWF requires more communication delay and processing time on MH than RESTful-based MHWF.

C. *Results for Offloading Bandwidth Intensive Web Service*

The second application scenario is aimed to carry out tests for applications requiring intensive bandwidth. The String-Concatenation service used to evaluate the architectures consumes network bandwidth and demands CPU processing power depending on the size of the concatenated string. The request contains an integer parameter value *l*. *l* determines the number of iterations for concatenating a specific string. The output of this service (a concatenated string) is then returned to the client. The size of the concatenated string is controlled by varying the value of *l*. Consequently the size of response message payload is increased by increasing the input value *l*.

The first set of experiments is conducted to examine the amount of internal resources consumption for different values of *l*. These resources include both MH memory and MH processing power that are required during executing and offloading incoming Web Service requests. Tests run for both architectures. The memory consumption is averaged over 50 requests. Memory is estimated by calculating the difference between the total available

amount of memory on MH before processing incoming requests and the available memory after processing requests before sending them to clients. However, since the heap memory size of mobile devices is variable, then a technique for controlling the variation of mobile host memory is applied. This is done by releasing the unused objects freeing the memory heap before measuring the total available memory. Results presented in Fig. 17 show that with offloading, the memory consumed on the MH increases as the response message size increases. MH allocates more memory for storing the increased response before it is forwarded to the corresponding client. Another observation is that the REST implementation uses less memory than the SOAP based architecture. This is due to the smaller overhead of REST messages compared to the corresponding SOAP messages. Then, the second examined resource is the CPU load consumed by the MH. This is determined by measuring the average process time on MH (averaged over 50 requests). Fig. 18 presents the effect of varying response message lengths on the average processing time for the SOAP- and REST implementations. The results show that the MH spends more time receiving and reading responses with larger payloads than those with smaller payloads. Moreover, the average processing time needed by the SOAP implementation to run a service is larger than the average processing time needed by the REST implementation. SOAP requests require comparatively heavy weight parsers to un-wrap the SOAP envelope from the incoming HTTP POST request while requests in REST use light weight string-based parsers. Thus, the REST implementation consumes overall fewer resources than the SOAP implementation.

The second set of experiments designed to evaluate the bandwidth required to offload and distribute the execution of mobile Web Services between several mobile nodes. This was accomplished through measuring the total amount of information that is transferred between client, MH and AMH. String-Concatenation service is used again, and as the input value *l* increases, the size of the concatenated string increases as well, which results in an increase of the response message size. This is clearly shown in Fig. 19.
In this case SOAP needs more information than REST by approximately 482 bytes to store the Web Service parameters and method names inside the body of the HTTP request. Therefore, SOAP messages require more wireless bandwidth than REST messages.

The third set of experiments measured the average response time for different input values of *l* for both architectures. Response time includes the processing time spent on both MH and AMH for handling client request, invoking the required Web Service, executing it, composing the result and sending it back to the client. In addition, it involves the transmission delay for messages to transfer between the designated mobile nodes through socket connections.

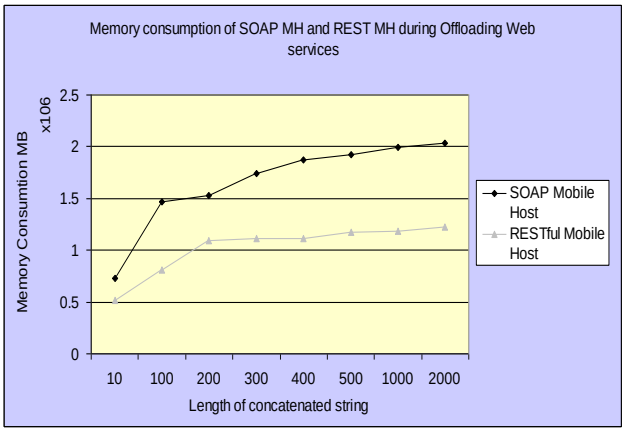

Figure 17: Memory Consumption of SOAP and REST mobile hosts

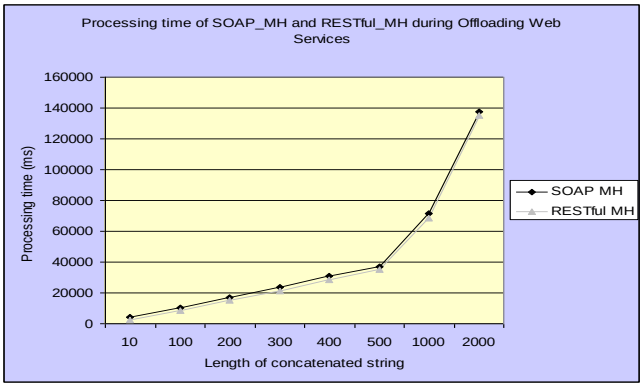

Figure 18: Processing time for SOAP and REST mobile hosts

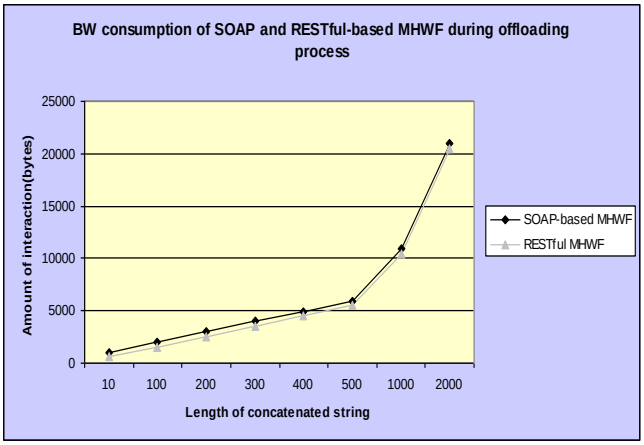

Figure 19: Bandwidth consumption of SOAP- and RESTful-based MHWF during offloading

The results of this experiment are presented in Fig. 20. As the size of the response message increases, the average response increases. This is expected because for this experiment, the response time is composed of the MH processing time, which increases with increasing message size as shown in Fig. 6. AMH processing time is another component for the response time that also increases with increasing message size as shown in Fig. 18.

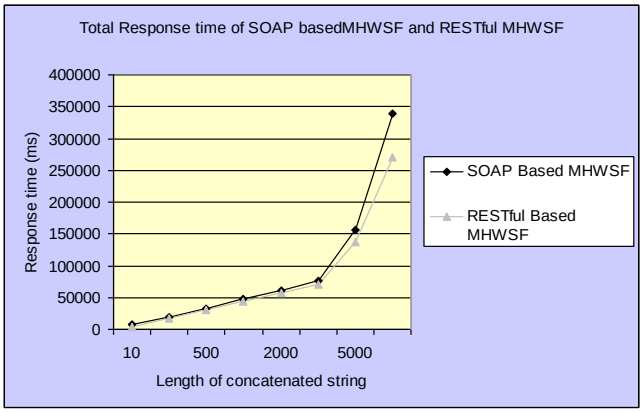

Figure 20: Total response time for SOAP- and RESTful-based Web Services

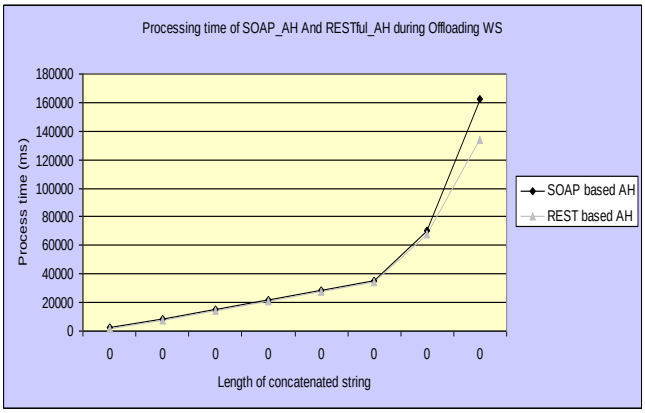

Figure 21: Response time for SOAP/RESTful-based MHWF during offloading

D. *Offloading overhead Experimental Results*

The overhead of distributing the execution of conventional SOAP-based MHWF and the new RESTful-based MHWF is examined in this section. The overhead is caused by the coordination and management of the task partitioning. The overheads include memory, processing, response time and signaling/messaging. Moreover, this is measured in for both implementations (as fore described). In this set of experiments we implemented prototypes for both architectures based on the typical original MHWF. Each of these prototypes consists of a client simulated using Sun Wireless Toolkits 3.0 emulator and n97 Nokia mobile host. The mobile host and client are connected in a wireless network. The test is carried out using the two aforementioned resource intensive applications. (i.e., PI and String-Concatenation)

In all experiments only one parameter is measured at a time. Each client operates cyclically and sends one request waits until it receives the response back then repeats the cycle and sends the same request again. This cycle is repeated 50 times for each experiment and the average of these 50 measurements is calculated. Then the measured parameters are compared with its corresponding parameters that are measured during applying offloading mechanism. As mentioned above these parameters include memory and processing consumption on the MH that indicate the amount of resource consumption overheads. Other parameters are the amount of interaction and response time that indicate the amount of communication/signalling overheads. RESTful-based MHWF framework shows an inferior performance in comparison to SOAP-based MHWF framework regarding distribution of mobile Web Services. Fig. 22 and Fig. 23 emphasize this fact and prove that RESTful MHWF shows smaller resource consumption and signalling overheads than SOAP- based MHWF. RESTful MHWF is also preserve approximately 42% more amount of memory than SOAP MHWF for k=10 in PI application. Moreover, the difference in overhead is more obvious for applications with more processing and bandwidth intensity. For example, in String-Concatenation test case REST-based implementation requires approximately 70% less processing cycles, 68% reduced delay and 59% fewer messages to provide the same service in SOAP-based implementation.

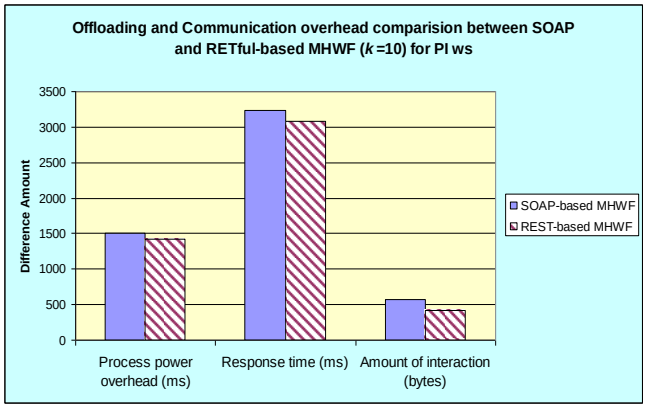

Figure 22: Offloading and Communication overhead for SOAP and RESTful-based MHWF (N=10) for PI Web Service

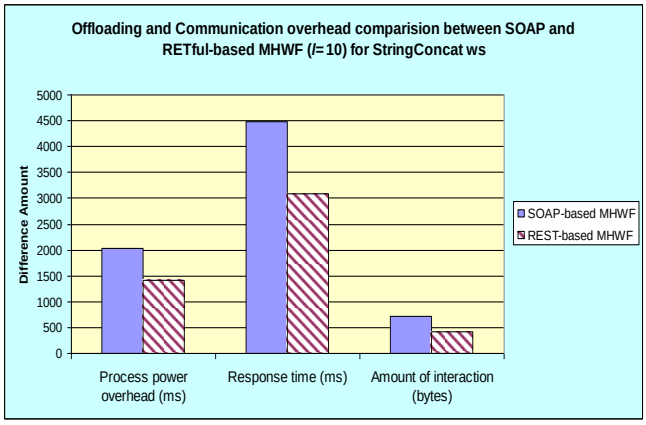

Figure 23: Offloading and Communication overhead for SOAP and RESTful-based MHWF (N=10) for String-Concatenation Web Service

E. Performance Improvement Experimental Results

The performance of distributing the execution of conventional SOAP-based MHWF and the new RESTful-based MHWF has been further analyzed and examined in this section. This analysis is carried out to critically measure the amount of REST over SOAP performance improvement gained from offloading. The parameters that are used for measuring performance improvement include amount of memory, response time and total message length enhancement. These parameters are evaluated for both Web Service samples (PI and String-Concatenation). Results in Fig. 24 and Fig. 25 show that offloading and distributing RESTful Web Services can achieve more performance improvement over its corresponding SOAP Web Services compared to the improvement that can be achieved in non distributed environments. In addition, the amount of processing power enhancement is slightly more for computational intensive. On the other hand, the amount of communication delay enhancement is more for bandwidth intensive applications.

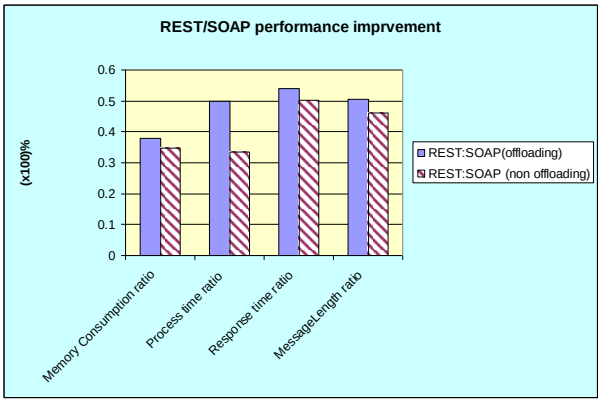

Figure 24: REST/SOAP performance improvement for offloading and non-offloading Web Services (PI Web Service)

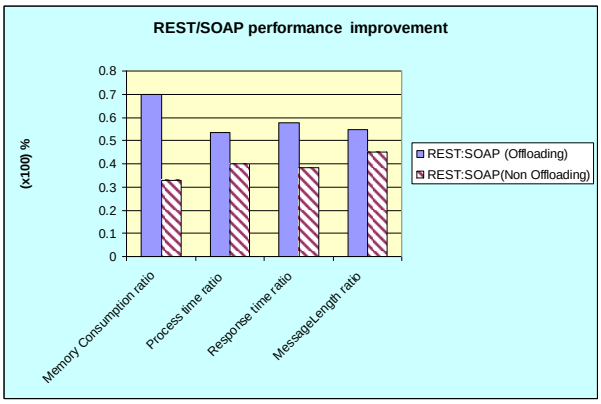

Figure 25: REST/SOAP performance improvement for offloading and non-offloading Web Services (String-Concatenation Web Service)

VIII. DISCUSSION

RESTful- versus SOAP-based mobile Web Service distribution are evaluated based on four main parameters. These parameters constitute an essential infrastructure used for selecting the most appropriate WS provisioning framework for providing distributed mobile Web Services. One of the most vital parameters is performance, which always forms the main goal for building efficient frameworks

Performance is measured by testing the average processing time on the main mobile host in addition to the average response time for Web Service requests. Results meet our expectations and show that RESTful-based MHWF provides improved processing time and response time over its corresponding SOAP-based Web Services. This is because SOAP-based Web Services require heavy weight parsers to un-wrap incoming request and extract the hidden SOAP envelope from the body of HTTP request. But RESTful Web Services require light-weight parsers based on string manipulator. This String-based parser is needed to extract the information required for invoking Web Services. This information resides explicitly on HTTP request.

Moreover, the improvement achieved in the average processing time is more for offloading case than non-offloading case. This is due to the distribution of Web Services and partial execution of Web Services on MH. Results have also shown that the processing time for Web Services with fixed length message payloads is almost steady state in the offloading environment and does not vary with increasing the processing power complexity. This is because processing time on MH consisted of reading the incoming request, identifying the parameters required for invoking a Web Service such as method name, service name and related parameters, forwarding these parameters to AMH, reading incoming responses from AMH, comparing the response and sending it to client. Thus, processing Web Service on MH depends mainly on the size of the incoming and outgoing respective requests and responses. Processing time on MH does not depend on the complexity of the Web Service logic that will be executed remotely on AMH. Similarly, RESTful-based MHWF provides better average response time than SOAP-based MHWF due to support for caching. In addition, response time involves processing time on MH, processing time on AMH and communication delay.

As illustrated earlier processing SOAP requires comparatively heavy-weight parsers and consumes more time. Furthermore, communication delay is directly proportional to the size of transferred message, which is larger for SOAP than REST. The second dominant parameter is scalability and reliability of the developed framework. Since RESTful Web Services are idempotent, therefore, sending repeated request to compensate for reliability is safe and simple. On the other hand, reliability of SOAP is achieved by using a WS- reliability standard that encounters

some implementation complexity and augments the size of the original SOAP message. Results present more scalability with RESTful-based MHWF and more requests can be executed concurrently than the conventional SOAP-based MHWF. This is because REST requests are stateful and reduce the need for the MH to maintain communication state. RESTful-based MHWF is also more scalable, because its corresponding requests are smaller in size and occupy less space waiting in the server queue than SOAP requests. Another parameter that is addressed by that evaluation is the amount of consumed resources: internal and external constrained mobile resources. Results have proved that RESTful-based MHWF preserves more processing power, memory storage space and network bandwidth than SOAP-based MHWF. This is because processing SOAP requests requires more extensive processing power for parsing and serializing SOAP object. More memory is also needed to load parser libraries and to store temporary parsed objects.

Furthermore, in comparing SOAP and REST requests we can easily notice a significant reduction in requests payload. Hence, RESTful-based MHWF consumes less network bandwidth during transmission of smaller REST message payloads. This result is more trivial with bandwidth intensive applications where the amount of interaction reduction increases approximately from 54%-97%.

The last parameter is the overhead caused by adding offloading module to the existing framework. Results have shown that RESTful-based MHWF intercepts less overhead than SOAP-based MHWF. This is due to less total amount of interactions, processing time, response time and memory requirement.

However, there are some limitations with RESTful Web Services. First they are only used for HTTP transport layer. In addition, transaction and federation are not supported by REST. SOAP is more suitable for complex Web Services that require a contract in advance between client and Web Service provider.

IX. Conclusion and Future Work

Mobile Web Services are provided from resource constrained mobile hosts in an intermittent wireless network. Thus, so far there were clear limitations in terms of complexity and size of the services that may be executed on mobile hosts. Providing adaptive mobile Web Services is vital to allow reliable provision of complex Web Services from resource limited mobile devices to overcome resource constraints.

This paper has explored one of the mechanisms used to facilitate the provisioning of adaptive mobile Web Services. The explored mechanism is known as offloading. This is accomplished by extending the two frameworks SOAP-based MHWF and RESTful-based MHWF developed in [1]. The novelty of this work to the best of our knowledge is that it is the first work that investigates provisioning of RESTful-based distributed Web Services from mobile devices.

The two frameworks are extensively tested and analyzed using two types of applications, process intensive application and bandwidth intensive application.This analysis is needed to select the most appropriate implementation technology that suits adaptive and distributive mobile Web Services.

Our preliminary work shows that extended RESTful-based MHWFs outperform SOAP-based MHWFs. Moreover, RESTful-based MHWF has less offloading and interaction overhead. In addition, it has more performance improvement over SOAP-based MHWF and less resource consumption in offloading environment than in non-offloading environment.

The level of resources consumption improvement depends on the type of application. Performance enhancement is obvious for resource intensive applications.

In addition, RESTful-based MHWF supports caching; this saves the limited network bandwidth and increases reliability and scalability. It also reduces consumption of mobile resources. Another feature of RESTful Web Services is the loosely coupled relation between the server and client because of the uniform interface that adds a balance towards using it for distributed mobile Web Services.

Regarding future work, the first area of interest is to investigate other schemes for offloading Web Services such as the Bounce-offload strategy. Another interesting issue is to define a general structure for implementing Web Service logic to facilitate partitioning it and build an interface for orchestrating the services [20]. Moreover, distributing and offloading Web Services in dynamic mobile environment must consider multiple, possibly contradictory, issues. For example, executing a code component on a remote AMH might reduce MH energy usage at the cost of increasing execution time. Moreover, due to the variable nature of the environment, it is not feasible to use static policies to determine when and where to remotely offload services as the current resource situation may make any statically chosen policy obsolete.

References

[1] Moessner, K. and AlShahwan, F., "Providing SOAP Web Services and RESTful Web Services from Mobile Hosts," in ICIW 2010 Fifth International Conference on Internet and Web Applications and Services, 2010, Barcelona, pp. 174-179.

[2] Srirama, N., Vainikko, E., Sor, V., and Jarke, M. "Scalable Mobile Web Services Mediation Framework," in 2010 Fifth International Conference on Internet and Web Applications and Services, Barcelona, Spain 2010.

[3] Ayyagari, D., Yongii, Fu., Jingping, Xu., and Colquit, N., "Smart Personal Health Manager: A Sensor BAN Application: A Demonstration," in Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE, 2009, pp. 1-2.

[4] S.-A. Ong and N. R. Center. 2006, A Mobile Webserver-Based Approach for Tele-Monitoring of Measurement Devices. Available:http://www.sigmobile.org/mobisys/2006/demos/Ong.pdf, Access:03.01.11

[5] SOAP Definition. Available: http://en.wikpedia.org/wiki/SOAP, Access:03.01.11

[6] Fielding, R., "Architectural styles and the design of network-based software architectures," PHD, University of California, Irvine, 2000.

[7] Berger, S., McFaddin, S., Chandra, N., and Mandayam "Web services on mobile devices-implementation and experience," in Mobile Computing Systems and Applications, 2003. Proceedings. Fifth IEEE Workshop on, 2003, pp. 100-109.

[8] Asif, M. and Majumdar, S.,"Performance analysis of Mobile Web Service Partitioning Frameworks," in ADCOM 2008, 16th IEEE International Conference on Advanced Computing and Communications, 2008., 2008, pp. 190-197.

[9] L. Luqun, "An Integrated Web Service Framework for Mobile Device Hosted Web Service and Its Performance Analysis," in HPCC '08, 10th IEEE International Conference on High Performance Computing and Communications, 2008, pp. 659-664.

[10] Srirama, N., Jarke, M., and Prinz, W., "A Mediation Framework for Mobile Web Service Provisioning," in Enterprise Distributed Object Computing Conference Workshops, 2006. EDOCW '06. 10th IEEE International, 2006, pp. 14-19.

[11] Aijaz, F., Adeli, S., and Walke, B., "Middleware for Communication and Deployment of Time Independent Mobile Web Services," in ICWS 2008, IEEE International Conference on Web Services,2008,pp. 797-800.

[12] Majumadar, S., Asif1, M., and Dragnea2, R., "Partitioning the WS Execution Environment for Hosting Mobile Web Services," in SCC 2008, IEEE International Conference on Services Computing, 2008, vol. 2, pp. 315-322.

[13] Asif, M., Majumadar, S., and Dragnea, R., "Hosting Web Services on Resource Constrained Devices," in ICWS 200, IEEE International Conference on Web Services, 2007, pp. 583-590.

[14] Kim, Y.-S. and Lee, K.-H.,"A light weight framework for mobile web services " Computer Science - Research and Development, pp. 199-209, May 2009.

[15] Saad, M., Hamad, H., and Abed, R., Computer Engineering Department, Palestine, "Performance Evaluation of RESTful Web Services for Mobile Devices" International Arab Journal of e-Technology vol. 1, p. 7, January 2010.

[16] Aijaz, F., Ali, S., Chaudhary, M., and Walke, B., "Enabling High Performance Mobile Web Services Provisioning," in Vehicular Technology Conference Fall (VTC 2009-Fall), 2009 IEEE 70th, 2009, pp. 1-6.

[17] Aijaz, F.,Ali, S., Chaudhary, M. A., and Walke, B., "Enabling resource-oriented Mobile Web Server for short-lived services," in Communications (MICC), 2009 IEEE 9th Malaysia International Conference on, 2009, pp. 392-396.

[18] Corroy, S., Beiten, J., Ansari, J.,Baldus, H., and Mahonen, P., "Selection of Computing Elements for Energy Efficiency in Wireless Sensor Networks using a Statictical Estimation Method," International Journal on Advances in Networks and Services vol. 2, p. 10, 2009.

[19] Available: http://en.wikipedia.org/wiki/Pi,Access:03.01.11

[20] Pietschmann, S., "A Model-Driven Development Process and Runtime Platform for Adaptive Composite Web Applications," International Journal on Advances in Internet Technology, 2009, vol. 2, pp. 277-290.