Providing and Evaluating the Mobile Web Service Distribution Mechanisms Using Fuzzy Logic

Feda AlShahwan
Centre for Communications Systems Research, University of Surrey, Surrey, UK
Email: F. AlShahwan@surrey.ac.uk
Francois Carrez and Klaus Moessner
Centre for Communications Systems Research, University of Surrey, Surrey, UK
Email: {F. Carrez, K. Moessner}@surrey.ac.uk

Abstract—providing adaptive resource intensive Web services from mobile hosts needs to be done in a rather light-weight manner to allow continuous service provisioning. Processing and communication will drain the battery rapidly; hence both should be kept at a minimum. This paper describes the outcomes of an investigation into offloading and migration mechanisms that facilitate provision of adaptive and distributed mobile Web services. The investigation goes through three phases. The first phase integrates these mechanisms with the Simple Object Access Protocol (SOAP) and Representational State Transfer (REST) architectures producing extended mobile Web service frameworks. This phase is achieved by the implementation of a prototype that allows performance evaluation of both extended frameworks. The evaluation of the load and performance of the distributed services is taking place using resource intensive applications. The results presented show that basing distributed mobile-hosted services on REST is more suitable than using SOAP as underlying Web service infrastructure. The second phase relies on the outperforming REST-based framework to examine four distinct strategies for mobile Web service distribution mechanisms. In the last phase, evaluation results of the second phase are interpreted as Fuzzy Logic rules. These rule sets are used to trigger and control offloading schemes.

Index Terms—Adaptive mobile Web services, Fuzzy Logic, migration, offloading, REST, SOAP

Mobile Web services are self-contained modular applications that are defined, published and accessed across the Internet, in a mobile communications environment using standard protocols. 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. Mobile Web services can be classified as shown in Figure 1, into three main categories depending on the role taken by the mobile device in the mobile Web service environment. These categories are: Mobile Provider Web Services MPWS, Mobile Consumer Web Services MCWS and Mobile P2P Web Services MP2PWS.

Figure 1: Classification of mobile Web services

Most of the research on mobile Web services has focused on consuming standard Web services from mobile devices MCWS [1-4]. However, solving the issues related to consuming Web services from mobile devices alone is not sufficient. This is because the future of the network is directed towards P2P networks, where each node plays dual functionalities such as a client and server, for consuming and providing Web services respectively. So there is a need for studies on provisioning Web services from mobile hosts. These studies become useful since hosting Web services on mobile devices has an enormous number of useful real life applications such as location-based applications, healthcare applications, pervasive gaming, data collection, mobile learning services, etc...

There is a range of benefits to be gained from providing location-based Web services from mobile hosts. For example, embedding mobile hosts with Global Positioning System (GPS) receivers allows the tracking of the current location of a fleet, or on high value goods and their delivery[5]. Some of these location-based services require non-interrupted reliable provision from mobile devices to allow providing latest instant information before it becomes obsolete. Moreover, light-weight processing of Web services is needed to compensate for limited resource mobile hosts. The main objective of resource compensation is to free some resources of the mobile device and allow it performs its core functionality and enhances its performance.
The technologies used for developing Web services can be classified into two groups: RESTful- and SOAP-based Web services. This classification is based on the architectural style used in the implementation technology. SOAP stands for Simple Object Access Protocol. It is defined as protocol specification for exchanging structured information in the implementation of Web services in computer networks [6]. REST stands for Representational State Transfer; it is a resource oriented technology and it is defined by Fielding in [7] as an architectural style which consists of a set of design criteria that define the proper way for using web standards such as HTTP and URI’s. Both SOAP and RESTful-based Web services 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.

This paper is an extension of our previous work[8]. Both studies aim to provide non-interrupted reliable mobile Web services in a light-weight processing manner. This can be accomplished by investigating and developing mechanisms to allow distribution and adaptation of complex mobile Web services. However, this paper goes further and provides an extensive investigation of two different frameworks for implementing these mechanisms. In addition, it explores more mechanisms and presents an evaluation of different schemes for applying these mechanisms. This assessment is acquired to facilitate selecting the most appropriate scheme and framework for the most suitable corresponding application scenario. The main characteristics of this selection are: first, it overcomes the constrained mobile environment. Second, it is carried out with assistance of a Control system based on Fuzzy logic.

Offloading and migration of Web services are the mechanisms to be explored towards achieving our goal. This exploration as mentioned above includes the evaluation of these mechanisms for different frameworks through extending our previous implementation of both SOAP- and REST-based mobile hosted services. Our original implementation[9] has been extended to facilitate offloading and migration of services and service fragments. Moreover, it also includes testing our implementation for different offloading and migration schemes. These schemes differ in the fragments execution process and data model transfer.

The paper is organized as follows. First, a brief overview of the current state of the art for hosting Web services from mobile devices is introduced in Section II. Also, some of the proposed solutions to compensate the limitations of mobile resources are highlighted in Section III. Then, the mechanisms used for mobile Web service distribution are outlined in Section IV. This is followed by a description of the extended mobile Web services architecture in Section V. Section VI provides experimental evaluation of the distribution mechanisms. This evaluation is accomplished using two different approaches: framework- and strategy- based evaluation. Furthermore, a detailed demonstration of the Fuzzy Logic Module for controlling the selection of the suitable offloading strategy is presented in Section VII. Finally, Section VIII contains the conclusions drawn from this work and provides recommendations for future work.

I. STATE OF THE ART

There has been extensive research into the development of Mobile Hosts Web service Frameworks (MHWF). Most of the implemented frameworks allow deploying and providing SOAP-based mobile Web services either in a client/ server environment [10-12] or in a P2P network [4, 13, 14] [15]. Few researchers have explored applying mechanisms that allow mobile Web services adaptation and compensation for the lack of resources. For example, [11] proposed a partitioning technique to the layered MHWF approach [16] that allows the execution of complex large Web services on Mobile Hosts (MHs). However, in this approach clients send requests first to a stationary intermediate node, which fails to meet an essential mobility prerequisite of mobile Web service hosts. Moreover, this approach supports only 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 [17] contains built-in modules to support continuous provisioning of mobile Web services in P2P network environment. This is accomplished through service migrating to another surrogate mobile node when the mobile host’s battery power comes to its end. However, this framework provides SOAP-based simple Web services and does not allow a lightweight process of complex mobile Web 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 [18] but this evaluation is limited to mobile Web service consumers and does not include mobile Web service providers. The concept of REST-based Mobile Web services is introduced in [22] and a comparison with SOAP is carried out in [19], [20] and [4]. On the other hand, implementation of mobile hosts based on REST is not addressed. A REST-based Mobile Web service framework has been developed and initially evaluated in [9], a comparison is carried out between SOAP- and REST-based mobile service hosts. The evaluation involves performance, resource consumption and scalability. The results indicate that REST-based mobile hosted services are more suitable than those based on SOAP for resource limited mobile network environments. However, the approach in [9] does not address the non-interruptible provisioning of complex mobile Web services in a light-weight processing manner.

The resource limited mobile devices, the intermittent network connections and the frequent context and location change of a mobile host act as a barrier against the smooth development of this area. To alleviate this limitation, we aim at facilitating light-weight provisioning of complex mobile Web services through service fragment offloading and data migration, thus
reducing the individual mobile hosts’ energy usage and increasing the range and complexity of services that can be executed/provided on MHs. Although, offloading mobile applications is not a new concept and has been previously used for application distribution and load balancing [23][29], but it has not been used for offloading the execution tasks of mobile Web services to run remotely on other mobile devices. In addition, it has not been used as a comparison parameter to distinguish between different mobile Web service frameworks’ performance. The next section highlights the motivation towards mobile Web service distribution, it also, presents offloading and migration mechanisms that are used for service distribution. Furthermore, it elaborates in brief some of the researchers’ attempts for resources compensation and services adaptation.

II. MOBILE WEB SERVICE DISTRIBUTION

Providing adaptive Web services from MHs is a new approach in mobile Web services to cope resource scarcity of mobile network environment. This approach is explored through investigating mechanisms that are used to achieve mobile Web service adaptation. The main area of these mechanisms are offloading and migration, which allow continuous and reliable provision of mobile services through fragmenting and distributing the execution tasks of mobile Web services and modeling the transfer of required location-based information. In spite of the fact that mobile agents and mobile Web service distribution are common in their main functionality, since both allow the migration of computer software from one computer to another autonomously and continue its execution on the destination computer. However, mobile agents do not include partitioning the execution tasks of the software and the decision on when and where to move relies on the migrated software itself.

There are some factors that necessitate Web service distribution. One of these issues relates to the enormous spreading of distributed computing systems in a P2P network. Therefore, the application of distributed mobile Web services executed and deployed in a distributed network environment is an important direction for future research. Moreover, most of mobile devices are equipped with built-in cameras that are used by MHs to take snapshots and save pictures. This leads to the emergence of complex services provided by MH such as Image processing- and video stream-based services, which consume a huge amount of the constrained mobile resources. Therefore, distributing Web services takes place to lessen the processing load 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 limitation will still exist when the user demands increase. For example, the memory capacity of mobile devices will continue to increase but memory limitation occurs when a mobile host runs multiple services or multiple instances of the same service. Furthermore, battery life will always be a bottleneck for any portable device. Hence, the distribution of 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 functionalities.

The purpose of this research as mentioned above is to investigate mechanisms and technologies that facilitate mobile service distribution. The scope of this research focuses on location-based Web service adaptation. This is because location-based services are one of the most useful applications that can be provided from hosting services on mobile devices. Moreover, embedding MHs with built-in GPS and cameras and the large processing amount required by image processing applications make provisioning of location-based mobile services ideal for interpreting heavy-weight image processing applications. Offloading and migration mechanisms are used to achieve dynamic distribution of mobile Web services. The execution logic of a Web service is fragmented into partitions that are executed remotely on different Auxiliary Mobile Hosts (AMH). This mechanism is called offloading. Simultaneously, the method for transferring the required location-based data is modelled. This mechanism is called migration. There are some tasks that can only be executed locally and cannot be offloaded such as resource- or location- dependant tasks. This is because some location-based services require depend on the current location of the MH and require invoking the location-based information.

In this paper, offloading is classified based on the fragmenting methodology used for handling requests and responses into two main types: Bounce- and Forward-Offloading. In Bounce-Offloading, the MH dispatches the request back to the client, which sends the forwarded request to the AMH to process it and sends the results directly to the client. However, in Forward-Offloading, the MH transmits the request to the AMH that processes the request and sends the result to the client via the MH. Moreover, there exist two classes of migration: Backend and Frontend. These two classes differ in the data model transfer used for data extraction. For example, the AMH invokes the required data from the MH at execution time in case of Backend strategy but the data is transferred in advance to the AMH in case of Frontend strategy.

Consequently, these classifications results in four different distribution strategies: Backend-Bounce Offload (BBO), Frontend-Bounce Offload (FBO), Backend-Forward Offload (BFO) and Frontend-Forward Offload (FFO). The main difference between these strategies is in the method used by the mobile host and the auxiliary mobile host for processing requests and responses. They are also distinguished by (transfer mode) the invocation scheme used for extracting the parameters needed to process the requested service. For example, in BBO shown in Figure 2, the MH sends the request back to the client associated with location address of the selected AMH; the required parameters are invoked by the AMH from the original MH. FBO illustrated in Figure 3 allows...
the MH to send the request back to the client associated with the AMH's URI and the required parameters. In BFO scheme which is shown in Figure 4, the MH delegates the request to the selected AMH, which enquiries the required parameters from the MH. Finally, FF0 elaborated in Figure 5 forwards the request from MH to AMH in conjunction with the required parameters.

The message sequence diagram for each of the four schemes has the same initial exchanged messages used for initiating a three handshake TCP socket connection with the MH. After establishing a connection, the client sends the request message to the MH, which detects the existence of the acquired service. MH sends an error message to the client if the service is not available and closes the connection. However, if the invoked service is available then the MH processes the request. Processing the request by MH depends on the type of the distribution scheme. For example, in BBO (Figure 2) the MH redirects the request together with the location address of AMH to the client and sends messages to close its connection with the client. Then, the client initiates a socket connection with the designated AMH by sending "Connector.open(socket://AMHsc:port,Connector.READ_WRITE)". After setting up the connection, the client sends the request message for executing the service to the AMH. Then the AMH verifies the availability of the service and extracts the location dependent data from the MH. Thus, the AMH initiates a socket connection with the MH by sending "Connector.open(socket://MHsc:port,Connector.READ_WRITE)". Moreover, the AMH requests the data from MH after establishing the connection by sending submitRequest (MH, getDataMessage). The MH sends the requested information and closes the socket connection. Then, the AMH processes the request and sends the results directly to the client. The message sequence of FBO that is illustrated in Figure 3 is generally similar to the BBO except that the location-based needed data are directly forwarded to the AMH through the client and does not require creating a connection between the AMH and MH. On the other hand, the message sequence of BFO (Figure 4) has the same common initial messages for setting up connection between the client and the MH, sending the request by the client, checking the validity of the acquired service and sending an error message to the client when service is not available. However, it differs in processing the request when the service exists. In BFO, the MH establishes a three handshake TCP/IP socket connection with the AMH and forwards the client's request to it. The AMH sends submitRequest (MH, getDataMessage), which is a request to the MH acquiring location-based data that is needed to process the client's request. The AMH processes the request and sends the final result to the client through the MH and closes the (MH-AMH) connection. Finally the MH forwards the response to the client and closes the (MH-client) connection.

The message sequence diagram shown in Figure 5 for FF0 is similar to BFO but the location-based acquired information is directly forwarded from the MH to the AMH.
Distributing Web services can be accomplished through defining criteria for triggering distribution and controlling the distribution strategy using a Fuzzy Logic-based control system and a resource monitoring system. The approach to achieve our objective consists of two phases: The first phase is to examine the offloading and migration mechanisms for different architectural styles and offloading strategies. The second phase is to build a Fuzzy Logic control system in order to set up rules for triggering offloading time and deciding on type of offloading strategy based on some input and output parameters. This exploration takes place in order to minimize resource consumption as much as possible with preservation of the ubiquitous computing. In the next section, we will describe in brief the architecture of the extended RESTful- and SOAP-based MHWF.

III. MOBILE WEB SERVICE DISTRIBUTION ARCHITECTURE

The first step towards achieving our goal is to build, provide and execute distributed mobile Web services using Extended Mobile Host Web service Framework (EMHWF) that consumes fewer amounts of resources, allows high performance and causes less distribution and offloading overheads. SOAP- and RESTful-based MHWF architectures described in [9] have been extended to allow distribution and offloading functionality. This is accomplished by using the same architecture for Auxiliary Mobile Host (AMH) that will take the role of a mobile host and handle the offloaded requests. However, the architecture of the MH is extended by adding FuzzyLogic and Offloading modules to the basic architecture (see Figure 6).

In general the architecture of Extended Mobile Host Web service Framework (EMHWF) for SOAP- and RESTful-based services is identical. However they differ in the details for handling and parsing the request. Each framework consists of seven basic building blocks: WebServiceServlet, HttpListener, RequestHandler, FuzzyLogic, Offloading, MessageParser and ResponseComposer

1. Web serviceServlet: It deploys new services into the mobile device and invokes the requested service. It also supports the flexibility of allowing Web service developers to customize the particular handling of requests and responses. Its structure is identical to the previous conventional MHWF but its hash table includes an additional field for defining the URI address for the AMH that will execute the acquired service;

2. HTTP Listener: The main functions of it include listening to incoming requests through ServerSocket class, accepting incoming client’s requests, initiating a new thread for each request to support concurrency and creating input and output stream for communication;

3. Request Handler: The main task for it is to process the request. However the way this task is carried out is different between the two frameworks.
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. However the request handler for RESTful-based MWSF will extract the HTTP request directly and send it to the message parser.

4. FuzzyLogic Module: It is the module that monitors internal and external resources and activates the Offloading Module whenever the MH is overloaded. It also decides upon the offloading strategy to be applied based on defuzzification of the Fuzzy set rules as illustrated later in this paper;

5. Offloading Module: The main task of the offloading module is to distribute the execution tasks of a mobile Web service among AMH and to model the transfer of the required data resources. The methodology used for these mechanisms depends on the applied offloading strategy as described later in this paper;

6. Parser Module: The main function for it 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 ServiceServlet. However, the way this is performed is different between the two frameworks. In SOAP-based MHWF, the SOAP parser deserializes the SOAP object and maps the data types into Java objects using kSOAP2 and kXML2 that are open source APIs for SOAP parsing. On the other hand, we have created our own string manipulator based parser to be used in RESTful-based MHWF. This parser will extract the needed information that resides explicitly in the client’s request;

7. Response Composer: It is responsible for interpreting the result through calling the getResponseXML() from HttpServiceResponse class in order to compose XML representation of the result then sending it back to the client. It can also interpret the result using different representations such as JavaScript Object Notation (JSON) but we have used XML as an example.

In the next section, we explore the distribution mechanisms for different Web service architectures and strategies.

IV. EVALUATION OF DISTRIBUTION MECHANISMS

Evaluation of mobile Web service distribution mechanisms compromises three distinct approaches: The first approach explores the performance gained from distributing mobile web service tasks, which has been analyzed previously using queuing theory[25]. The second one relies on examining service distribution with two different framework environments (SOAP and REST), which has been studied previously [8]. The third one depends on investigating service distribution with four offloading and migration strategies (BBO, FBO, BFO and FFO). The next section starts with evaluation of the first discipline.

A. SOAP- and RESTful- based frameworks

SOAP and REST are two different technologies used for developing Web services. This classification is based on the architectural style used in the implementation technology. We have developed two implementations for providing, executing and deploying stateless Web services from mobile devices: The first implementation is based on the conventional SOAP; the other one is based on REST. The architectures are fairly similar and there is no apparent difference in complexity. However, the main differences are in the details of handling and parsing the request. Consequently, both architectures differ in their performance. Moreover, the amount of internal and external resource consumption is quite different for both architectures.

1) Experiment Environment

The main objective of our experiment is to investigate offloading mechanisms and to examine the feasibility and validity of distributing SOAP- and RESTful-based Web services in mobile environments. Another goal is to test and compare between the two different architectures to assist in designing an architecture that is most suitable for distributing mobile Web services with fewer overheads and less resource consumptions. Offloading Web services has shown an increase in the MH performance in our previous work [25] over non-offloading.

In our experiments [8], the distribution of resource intensive applications in both architectures is evaluated using simple String-Concatenation service. The service is developed using identical algorithm for both architectures and on same JME platform to achieve similarity. In the String-Concatenation service, a constant word is merged and concatenated with itself several times. The number of
times constant is concatenated depends on a parameter (i.e., an integer value) set by the client. The bandwidth and processing demands can be easily controlled through this parameter, thus the application is suitable for testing the performance of both frameworks. Although, location-based services is the main scope of our study, however, the concatenated word in String-Concatenation service can be a representation of any location-based data in order to achieve simplicity. The prototype consists of three mobile devices: MH, which is executed on a mobile device (Nokia N97m) running MIDP 2.1 over Symbian OS. The other device, implementing the AMH that acts as mobile host when the original MH is overloaded, was executed also on an N97m. The client was executed on Nokia n80. However, the evaluation was conducted using Sun Wireless Simulation Toolkit as shown in Figure 7.

In this experiment, the Forward-Offload strategy has been applied to support autonomic and ubiquitous. Autonomy is accomplished because in Forward-Offload scheme, the distribution takes place transparently from the client. Since the MH is assumed to be overloaded, it processes only part of the incoming requests and forwards it to AMH. The MH elects an AMH. The election is carried out using probe request that is broadcasted to all mobile devices, which satisfy a set of predefined criteria. The probe request contains information about the minimum amount of required resources. Then, the first responded device is selected as AMH. However, selecting AMH is beyond the scope of this study. The evaluation is carried out using four different scenarios. In the first set of experiments, the level of internal resource consumption is examined. In the second set of experiments, the level of external resource consumption is calculated. After that, overall performance is evaluated in a third set of experiments by measuring total elapsed time for execution of each request. Finally, the offloading overhead is analysed and the performance improvement is evaluated for both (SOAP and REST) architectures.

2) Results and Discussion
The String-Concatenation service is used to evaluate the architectures. It 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 \( l \). Similarly, the size of response message payload is increased by increasing the input value \( l \).

The first set of experiments is conducted in order to examine the amount of MH memory capacity that is required during executing and offloading incoming Web service requests. Tests run for both architectures. The memory consumption is averaged over 50 requests. The amount of consumed memory is estimated by calculating the difference between the total available memory capacity on MH before processing incoming requests and the available free memory capacity after processing requests. However, since the heap memory size of mobile devices is variable, a technique for controlling the variation of mobile host memory has to be applied. This is done by releasing the unused objects and freeing the memory heap before measuring the total available memory using garbage collection. Since releasing and freeing heap memory and garbage collection consume a lot of CPU cycles and might affect the accuracy of some parameters such as processing power, they are executed only when measuring the memory consumption and therefore avoided when examining the processing and response time. Results presented in Figure 8 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.

The second set of experiments aims at evaluating the bandwidth required in order to offload and distribute the execution of mobile Web services between several mobile nodes. This is accomplished through computing the total amount of interactions between the three connected mobile devices (client, MH and AMH) during service execution. Figure 9 shows that as the input value \( l \) increase the size of the concatenated string increases as well, which results in an increase of the response message size. In this case, SOAP needs more information than REST 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 measures the average response time for different input values of \( l \) for both architectures. Response time is defined as the time that a client spends waiting to receive the result from the MH. It 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. The results of this experiment are presented in Figure 10. As the size of the response message increases, the average response increases. This is expected because for this experiment, the response time includes the processing time on MH, which increases with increasing message size. It also includes the processing time on AMH, which also increases with increasing message size. Moreover, from Figure 10, it can be concluded that the SOAP implementation requires more time than REST. This is due to the average processing time and communication delay needed by SOAP implementation in order to execute a service run is larger than its corresponding REST implementation. SOAP requests require comparatively heavy-weight parsers such as kSOAP and kXML libraries to un-wrap the SOAP envelope from the incoming HTTP POST request. But requests in REST use light-weight string-based parsers. Thus, REST implementation consumes overall fewer resources than SOAP implementation. It is observed that the increase in both Figure 9 and Figure 10 is linear then it becomes exponential. This is because the increase of the response message size is logarithmic.

The offloading overhead for distributing the execution of SOAP- and REST-based mobile Web services has been further analyzed. The overhead is caused by the coordination and management of the task partitioning. The overhead includes processing, response time and signaling/messaging overheads. This is measured for both implementations. In the experiment, the client sends a request and waits until it receives the response. The response time is averaged over 50 repetitions for both offloading and non-offloading scenarios and for both (SOAP and REST based) implementations. The REST-based framework shows smaller offloading overhead in comparison to the SOAP implementation. Figure 11 illustrates this and shows that using REST consumes less processing power and causes less overhead for offloading. In addition, REST shows also a smaller response time and less messaging overhead. The experiments have shown that the REST based implementation requires approximately 70% less processing cycles, has reduced delay (68%) and needs 59% fewer messages to provide the same service in the given test scenario.

B. Offloading and Migration Schemes

As aforementioned there are four distinct schemes for defining the service fragmenting process and data transfer model. These strategies result from classification of service offloading and data migration into BBO, FBO, FFO and BFO.

1) Experiment Environment

The main objective is to investigate offloading mechanisms, examine and compare between the four different offloading schemes. In our experiments, the provided services are based on RESTful Web services. This is because it has been previously shown in our preliminary work that RESTful- outperforms SOAP-based Web services in mobile network environment Web services. The architectures are fairly similar. However, the main difference is in the offloading module, which determines the strategy used for handling requests and extracting location-based parameters. This difference causes distinctions in their response time, performance and the amount of internal and external resource consumption between these four architectures. The service used is a simple Image-Processing service. This location-based service is one of the most common resources intensive applications that can be provided from mobile hosts, where clients send requests to the MH to process an image taken by the mobile camera from a specific location and adjust its dimensions to be compatible with the client’s screen size. These dimensions could be either all or a combination of: the height, width or bit depth of the picture. In this experiment, we modify only the image’s height. This is done to allow changing the size of the transferred message payloads using a parameter that is set by the client. The bandwidth and processing demands can be easily controlled through this parameter, thus the application is suitable for testing the performance of all the implementations. In addition, it is appropriate for representing data transfer models that are based on current location of the mobile host. Each prototype consists of three mobile devices: MH, AMH and client. The prototype is simulated using Sun Wireless Toolkit in Figure 12. The test is carried out for each of the four offloading strategies using two different scenarios.
In the first set of experiments, the level of internal resource consumption is examined including both memory and processing resources. In the second set of experiments, the response time is analysed.

2) Results and Discussion

The Image-Processing service used for testing, consumes a large amount of resources depending on the acquired image height. The request contains an integer parameter value $h$, which determines image height. The output of this service is then returned to the client. The size of the returned photo is controlled by varying the value of $h$.

The first set of experiments examines the amount of internal resources consumption for different values of $h$. These resources include both MH memory and MH processing power that are required during executing and offloading incoming Web service requests. Tests run for all four architectures. Amount of memory consumption is computed from calculating the difference in available memory capacity on MH before processing the incoming request and after processing it. This is averaged over 50 requests. Furthermore, the frequent size change of the MH's heap memory necessitates the application of a technique to control the variation of MH's memory. This has been described previously. Results presented in Figure 13 show that: in BFO and FFO schemes memory increases with increasing the payload, but in BBO and FBO it remains almost steady state. This is expected, since Forward Offload allocates more memory for storing the increased response message size before it is forwarded to the corresponding client. In addition, Backend scheme consumes more memory than its corresponding Frontend scheme. The larger memory is needed to open new socket connection with AMH and maintain input/output data streams. The second examined internal resource is the CPU processing load on MH. This is calculated by measuring the average process time on MH (averaged over 50 requests). Figure 14 elaborates the effect of varying response message lengths on the average processing time for the four different implementations. Results show that the MH in Forward-based framework consumes more processing time with services having large response payloads than those with smaller payloads. However, the average processing needed by the MH for Bounce Offload-based frameworks, such as FBO and BBO to execute a service run is almost stable and not affected with different response message size. Another observation is that the Backend implementation consumes more processing time than its corresponding Frontend architecture. This is due to the larger processing overhead needed by MH to process AMH requests that enquiry for location-based data.

The second set of experiments is conducted to measure the average response time. Response time is estimated by subtracting the time when a client sends a request from the time when it receives the response back. It includes processing time spent by MH and AMH for handling client request, invoking the required service, distributing the execution of it, composing the result and sending it back to the client. In addition, it involves the transmission corresponding Frontend scheme. The larger memory is needed to open new socket connection with AMH and maintain input/output data streams. The second examined internal resource is the CPU processing load on MH. This is calculated by measuring the average process time on MH (averaged over 50 requests). Figure 14 elaborates the effect of varying response message lengths on the average processing time for the four different implementations. Results show that the MH in Forward-based framework consumes more processing time with services having large response payloads than those with smaller payloads. However, the average processing needed by the MH for Bounce Offload-based frameworks, such as FBO and BBO to execute a service run is almost stable and not affected with different response message size. Another observation is that the Backend implementation consumes more processing time than its corresponding Frontend architecture. This is due to the larger processing overhead needed by MH to process AMH requests that enquiry for location-based data.

The second set of experiments is conducted to measure the average response time. Response time is estimated by subtracting the time when a client sends a request from the time when it receives the response back. It includes processing time spent by MH and AMH for handling client request, invoking the required service, distributing the execution of it, composing the result and sending it back to the client. In addition, it involves the transmission corresponding Frontend scheme. The larger memory is needed to open new socket connection with AMH and maintain input/output data streams. The second examined internal resource is the CPU processing load on MH. This is calculated by measuring the average process time on MH (averaged over 50 requests). Figure 14 elaborates the effect of varying response message lengths on the average processing time for the four different implementations. Results show that the MH in Forward-based framework consumes more processing time with services having large response payloads than those with smaller payloads. However, the average processing needed by the MH for Bounce Offload-based frameworks, such as FBO and BBO to execute a service run is almost stable and not affected with different response message size. Another observation is that the Backend implementation consumes more processing time than its corresponding Frontend architecture. This is due to the larger processing overhead needed by MH to process AMH requests that enquiry for location-based data.
delay for messages to transfer between the designated mobile nodes through socket connections. The results of this experiment are presented in Figure 15 and Figure 16. As the response message size increases, the average response time increases as well. Notice that for small payloads, FFO outperforms its corresponding offloading strategies as illustrated in Figure 15. We also observe that the response time for FBO becomes the smallest, when l almost exceeds 500. This is due to the larger communication delay needed for transferring the increased response message size from the AMH to the MH, then from MH to the client. However, for small response message, response time is small and its effect on the communication latency is almost neglected compared to the ACK signal needed by the client for confirming the recipient of location-based information. Results also show that BBO has the largest response time over the other three schemes because BBO mechanism requires creating and maintaining the largest number of socket connections to provide distributed mobile Web services. In all tests carried out the FBO implementation did outperform Forward-based and Backend-based implementations. This is more observable for resource intensive services. Thus, in terms of performance FBO is more suited for distribute resource intensive mobile hosted Web services. Consequently, Bounce-based strategies enhance performance by lessening the load on MH, preserving MH’s resources and connections. Hence, the scalability increases by preserving resources for handling more requests. However, these benefits are gained at the expense of putting more burdens on the client for redirecting the request to another AMH and requesting additional network and data stream connections with AMH. Forward-based implementations such as FFO and BFO spends more time receiving and reading responses with larger resources and connections. However, Forward-based implementations support ubiquitous computing through autonomous distribution of the execution tasks without the client being aware and are useful when MH has the required amount of resources to parse and forward requests.

The experimental analysis of the four distribution strategies has shown that each of these strategies has its own features that make it suitable for specific circumstances. Therefore, a Fuzzy Logic Module is needed to trigger offloading action. The functionality and methodology used in the augmented Fuzzy Logic Module are explained in the following section. Furthermore, Fuzzy Logic rule sets that are used to determine the suitable offloading strategy based on the current available resources and the payloads of the transferred messages are defined in the next section.

V. FUZZY LOGIC MODULE

FuzzyLogic Module is the decision making unit that triggers offloading time and action. The offloading action could be any one of the four aforementioned strategies: BBO, BFO, FFO or FBO. This is based on applying a predefined set of Fuzzy logic rules and examining amount of available mobile host resources. A decision is taken on the reaction and amount of resources to be freed up through the selection of the appropriate mechanism. Our model is an extension of [8] based on the control logic and control theory that have been proposed in [26]. Fuzzy logic has several features that make it suitable for mobile environment. One of the most significant features is the flexible constraint of the data parameters interpretation, which supports adaptation and variation of inputs without affecting the result decision. This characteristic is useful for the variable mobile environment and does not consume large amount of resources (CPU and Memory). Implementation of Fuzzy logic is not limited to the computation of precise input values. Any input data that provides some indication of a system's actions and reactions is sufficient. The broad acceptable data range allows inexpensive and imprecise measurement tools which do not require accurate and dedicated sensors, thus keeping the overall system cost
and complexity low. In addition, it is extensible since the user-defined rules governing the target control system can be extended by adding more inputs and generating appropriate rule sets. Furthermore, the FuzzyLogic Module is inherently robust. This is because it does not require precise, noise-free inputs and can be programmed to catch errors and fail safely. The output control is a smooth control function despite a wide range of input variations where this is the case in mobile environment. Application of Fuzzy logic concept has been extensively explored in various fields by researchers. However, to the best of our knowledge, it has not been applied it in the area of adaptive mobile offloading applications except in[24]. Moreover, utilizing Fuzzy logic concept for triggering the time and the mechanisms’ schemes for mobile Web service distribution is explored for the first time in this paper. The FuzzyLogic Module includes the following components:

- Linguistic decision making rules
- Linguistic variables and membership functions
- Generic Fuzzy inference control engine based on Fuzzy logic theory

JFuzzyLogic tool package[27] is written in java under an open source licence for researchers. It is used in our paper to implement Fuzzy Control Language (FCL). Fuzzy Inference System (FIS) is created using the FCL, which is defined by IEC 1331 part 7 specifications[28]. This FIS file defines the following input and output parameters of our FuzzyLogic Module:

- Available memory capacity (AvailMem)
- Available Processing power (AvailCPU)
- Available battery power (AvailBat)
- Available network bandwidth (AvailBW)
- Request message payload (ReqPayld)
- Response message payload (RspPayld)

The output parameter (OffloadAction) represents the action to be taken by the system for distributing Web service execution. The triggered action relies on both: offloading strategy and data modelling transfer.

FIS contains three parts: Inputs fuzzification, the rule-base application to obtain the output sets and the defuzzification of the outputs.

In our experiment, input parameters can be measured using any of the available third party application tools such as Nokia Energy Profiler that consumes a few KB. The triangular shape is used in this study as a base application to obtain the output sets and the defuzzification of the outputs. The OffloadAction output parameter is defuzzified in the FIS as follows:

<table>
<thead>
<tr>
<th>Table 1: Membership functions for input parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>// Fuzzify input variable 'AvailMem'</td>
</tr>
<tr>
<td>FUZZIFY AvailMem</td>
</tr>
<tr>
<td>TERM small := (0.0) (6400.0) (12800.0);</td>
</tr>
<tr>
<td>TERM large := (6400.0) (65536.1) (124672.0);</td>
</tr>
<tr>
<td>END_FUZZIFY</td>
</tr>
<tr>
<td>// Fuzzify input variable 'AvailCPU'</td>
</tr>
<tr>
<td>FUZZIFY AvailCPU</td>
</tr>
<tr>
<td>TERM low := (0.0) (33.1) (66.0);</td>
</tr>
<tr>
<td>TERM high := (12.0) (116.1) (220.0);</td>
</tr>
<tr>
<td>END_FUZZIFY</td>
</tr>
<tr>
<td>// Fuzzify input variable 'AvailBat'</td>
</tr>
<tr>
<td>FUZZIFY AvailBat</td>
</tr>
<tr>
<td>TERM poor := (0.0) (2.1) (4.0);</td>
</tr>
<tr>
<td>TERM moderate := (3.0) (5.5) (8.0);</td>
</tr>
<tr>
<td>TERM excellent := (6.0) (9.1) (12.0);</td>
</tr>
<tr>
<td>END_FUZZIFY</td>
</tr>
<tr>
<td>// Fuzzify input variable 'AvailBW'</td>
</tr>
<tr>
<td>FUZZIFY AvailBW</td>
</tr>
<tr>
<td>TERM weak := (0.0) (26.1) (52.0);</td>
</tr>
<tr>
<td>TERM strong := (24.0) (62.1) (100.0);</td>
</tr>
<tr>
<td>END_FUZZIFY</td>
</tr>
<tr>
<td>// Fuzzify input variable 'ReqPayld'</td>
</tr>
<tr>
<td>FUZZIFY ReqPayld</td>
</tr>
<tr>
<td>TERM short := (16.0) (64.1) (256.0);</td>
</tr>
<tr>
<td>TERM long := (128.0) (1024.1) (2048.0);</td>
</tr>
<tr>
<td>END_FUZZIFY</td>
</tr>
<tr>
<td>// Fuzzify input variable 'RspPayld'</td>
</tr>
<tr>
<td>FUZZIFY RspPayld</td>
</tr>
<tr>
<td>TERM short := (16.0) (64.1) (256.0);</td>
</tr>
<tr>
<td>TERM long := (128.0) (1024.1) (2048.0);</td>
</tr>
<tr>
<td>END_FUZZIFY</td>
</tr>
</tbody>
</table>

The rule-base application is implemented with a six input and one output system. Four rules were established and outputs were chosen based on the desired linguistic decision making rules as demonstrated in Table 4. The amounts of available resources, transferred payload, offloading strategy together with the data transfer model. Outputs of these logical rules determine one of the previous aforementioned four offloading actions. Setting
Table 4: Definition of the linguistic rules

RULE 1: IF AvailMem IS large AND AvailCPU IS high AND AvailBat IS excellent AND AvailBW IS strong

THEN OffloadAction IS NoF;

RULE 2: IF AvailMem IS large AND AvailCPU IS high AND AvailBat IS poor AND AvailBW IS weak AND ReqPayld IS long AND RspPayld IS short

THEN OffloadAction IS FFO;

RULE 3: IF AvailMem IS large AND AvailCPU IS high AND AvailBat IS moderate AND AvailBW IS weak AND ReqPayld IS long AND RspPayld IS short

THEN OffloadAction IS BFO;

RULE 4: IF (AvailMem IS small OR AvailCPU IS low) AND AvailBat IS poor AND AvailBW IS weak AND ReqPayld IS short AND RspPayld IS long

THEN OffloadAction IS FBO;

RULE 5: IF (AvailMem IS small OR AvailCPU IS low) AND AvailBat IS moderate AND AvailBW IS weak AND ReqPayld IS short AND RspPayld IS long

THEN OffloadAction IS BBO;

up the decision making rules is very critical stage and should be chosen carefully to obtain accurate decisions. For example, if the MH has the required amount of resources (AvailMem and AvailCPU) then forward-based offloading is the most suitable decision to achieve autonomy but if the available amount of resources (AvailMem or AvailCPU) is few, then bounce-based offloading is more suitable. After developing the FIS file, it is loaded and parsed using jFuzzy API. Then the crisp value of each input parameter is set up using setVariable(input, value) method. After that, the system is tested by calling evaluate() function. Figure 17 through Figure 23 demonstrate the membership function plots and linguistic term values for each input and output parameters. Due to space limitation, we show the output run for one test only. Figure 24 shows the output when the input values for AvailMem, AvailCPU, AvailBat, AvailBW, ReqPayld and RspPayload are 6400 KB, 100 MHz, 3.5hrs, 10 Mbps, 64KB and 1024KB, respectively. This is expected because of the limited resources on MH: membership rate for AvailMem to be small is 1.0 and membership rate for it to be large is 0, AvailCPU has 0.96 large membership rate value, AvailBat has 0.24 moderate membership, AvailBW is weak with 0.46 rate value and the ReqPayld has 0.24 short membership degree while the RspPayld has 0.98 long membership degree. Therefore, Rule4 and Rule5 are fired at 0.25 and 0.2 and the crisp output value is 39.51 to execute Bounce Offloading action.

We observe Bounce Offloading is suitable when the available internal resources of the MH are limited and the available external resources are moderate. Finally, the next section concludes our work and pinpoint on some of the open future issues.
VI. CONCLUSION AND FUTURE WORK

Adaptive mobile Web services can help to provide complex Web services hosted on resource limited mobile devices, despite their intrinsic resource constraints. This goal can be achieved through extending the architecture of our previous MHWF with the offloading module. Thus, we implemented two versions of architecture for distributed service execution. One based on SOAP, the other based on REST. Both implementations were tested and analyzed using a resource intensive application. Distributing the execution tasks of mobile Web services is feasible and can enhance service performance and reliability in mobile network environment. In all tests carried out in this paper (signalling overhead, message processing and delay) the REST implementation did outperform SOAP. Thus, in terms of performance REST is more suited for the implementation of mobile hosted Web services. In this paper the investigation of service distribution mechanisms has been extended to include offloading and migration. These two mechanisms are used to facilitate the provisioning of adaptive mobile Web services. Offloading is carried out through service distributing. On the other hand, migration is accomplished through modeling data transfer. This classification leads to four different offloading strategies: FFO, BBO, FBO and BFO. The extensive performance study of these schemes recommends applying Bounce-based strategies when resources are scarce. However, Forward-based strategies support anonymous service distribution, hence; it is preferred when moderate amount of resources is available. This behavior is translated and expressed using Fuzzy logic sets.

As a future work, we intend to explore more features of our Fuzzy logic module. This exploration extends to the examining of different membership functions, defuzzification methods, implementing more rule sets and experimenting different resources.

Offloading mobile Web services may lead to infinite cycles of forwarding or bouncing. This may occur when the delegated requests are always diverted to an overloaded AMH. Thus, as a future work, some strategies for selecting AMH and controlling the offloading process are required. Moreover, investigating service fragmentation, in cases where services are executed on more than one AMH is an interesting area of research. This is compromised of the methodology followed for fragmenting, orchestrating between the service fragments for exchanging data and coordination to accumulate response back to client.

REFERENCES


Feda AlShahwan born in Kuwait, She obtained her B.Sc, M.Sc in Computer Engineer from Kuwait University 1992, 2004 respectively. She is currently a Ph.D. student in University of Surrey in Mobile group of Centre for Communications Systems Research. Her current research interests include studies of Adaptive Mobile Web Services.

Dr Francois Carrez has received a PhD in Theoretical Computer Science from the University of Nancy – France in 1991. He has been then working 18 years for the Alcatel Research centre in Paris in areas such as Security, Distributed Artificial Intelligence, AdHoc Networking and Semantics. He recently joined the CCSR forces (2006) and is currently PM of the FP7 Coordination Action “Internet of Things Initiative”.

Prof Klaus Moessner is a Professor for Cognitive Networks and Services, in the Centre for Communication Systems Research at the University of Surrey, UK. Klaus earned his Dipl-Ing (FH) at the University of Applied Sciences in Offenburg, Germany, an MSc from Brunel University and PhD from the University of Surrey (UK). His research interests include dynamic spectrum allocation, cognitive networks reconfiguration management, service platforms and adaptability of multimodal user interfaces.