Fragmentation, Orchestration and Federation of Complex resource-based Mobile Web Services

Feda Alshahwan[1]*, Maha H. Faisal[2], Abdullah M. Almeshal[3], Tarek A. Selmi[4]
[1]College of Technological Studies, Public Authority for Applied Education and Training, Kuwait
f.alshahwan@paaet.edu.kw
[2]Kuwait University, Computer Engineering Department, Kuwait
maha.faisal@ku.edu.kw
[3]College of Technological Studies, Public Authority for Applied Education and Training, Kuwait
dr.almeshal.cts@gmail.com
[4]Electrical Engineering Department, Australian College of Kuwait, Kuwait
t.selmi@ack.edu.kw

Abstract

Complex mobile Web services can be provided from the mobile host through the development of Mobile Host Web service Framework (MHWF). However, the constraints of mobile resources may affect the core functionality of this device. Therefore, applying the distribution mechanisms to MHWF allows running more complex services in a light weight manner. Fragmentation, orchestration and federation are the main three used distribution mechanisms in this approach. These set of mechanisms, depends on the REST characteristics and HTTP methods to enhance service distribution. A general structure for the developed service logic has also been defined to facilitate tasks offloading. Verifying the correctness of the deployed Web service is carried out analytically based on formal methods. This is complemented with an empirical validation of the feasibility of the system through proof of concept prototype implementation.

Keywords: decomposing mobile Web services, distributed mobile Web service provision, orchestrating mobile Web services, RESTful-based mobile Web services

1 Introduction

The advances in the mobile device technology, progression of wireless networks and widespread use of Web services are increasingly making providing Mobile Web Services (MWS) popular. MWS are self-contained modular applications that are defined, published and accessed across the Internet, in a mobile communications environment using standard protocols[14]. MWSs can be classified into three main classes depending on the role taken by the mobile device in the MWS environment. These classes are: Mobile consumer Web services, Mobile hosting Web services and Mobile Peer to Peer (P2P) Web services. In mobile consumer Web services, mobile devices act as clients and request execution of Web services. For mobile producer Web services, mobile devices act as servers (hosts) and provide Web services to be invoked by clients. While in mobile P2P Web services mobile devices are connected in an ad hoc manner and each node plays dual functionality as a client and a server. Most of the research on MWSs has been focused on consuming standard Web services from mobile devices. However, solving the issues related to consuming Web services from mobile devices alone is not sufficient. This is because

IT CoNvergence PRActice (INPRA), volume: 3, number: 1 (March), pp. 51-78
*Corresponding author: College of Technological Studies Public Authority for Applied Education and Training, AlZahra, Block 4, Street 414, House 7, Kuwait, Tel: +965-22314735

the future of the network is directed towards P2P approach [24], where each node can be a client as well as a server with the capability to consume and provide Web services. Besides, hosting Web services from mobile devices has a range of useful applications in most aspects of real life. For example, embedding Mobile Hosts (MHs) with Global Positioning System (GPS) receivers allows the tracking of the current location of a fleet, or on high value goods and their delivery. Personalizing location-based services are feasible based on social networks [20]. Another application area for mobile Web services is in the health care domain [23]. Some applications require reliable and non-interrupted provision from mobile devices to allow providing latest instant information before it becomes obsolete. 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 place of the event. Furthermore, it requires mobile hosts that are aware of their location, publish the event as a live feed and take latest information gathered at the current location. Mobile hosts allow processing of the gathered information and photos, and then make them available, instantly, to clients. Moreover, some of these services are more complex and demand heavy-weight process. Thus, Adaptive mobile Web services are needed to compensate for the limited resource mobile hosts in order to free some resources for the mobile device to perform its core functionality and enhance its performance.

The main issue to be solved in this paper is the degradation in performance when providing complex Web services from mobile devices. This performance degradation leads to delay in response time for processing client's requests from the mobile Web service provider, scalability decrease in the number of handled requests, failure of the mobile host to perform its core functionality and difficulties in provisioning or consuming non-interrupted Web services from the MH, which is made more challenging by the range of constraints that MHs are exposed to [3]. The problem is magnified by the continuous variation to the current available amount of mobile resources [12]. In general, this limitation is due to the trade-off paid for the portability and the adverse effects of mobility on the connectivity of the devices [12]. Therefore, the aim of this approach is to allow non-interrupted provisioning of complex context-dependent mobile Web services in a light-weight processing demand. This is accomplished by defining and building a system that allows the collaboration of mobile devices to provide compound services as one homogenous service in a manner that does not deplete the devices limited resources too much and does not reveal the distribution process to the clients. The execution of these services is based on a distributed manner. The distribution scheme relies on the main characteristics of REST architectural style. The hierarchical structure of the Uniform Resource Identifier (URI) representation of the invoked services and the usage of HTTP uniform methods are the keystone used to facilitate service distribution. The paper is organized as follows. First, a short introduction to the current state of the art for hosting Web services from mobile devices is presented. Also, some of the proposed solutions to compensate the limitations of mobile resources are highlighted. Then, a detailed investigation of distributing mechanisms is included. This is followed by a description of the extended mobile Web services architecture. The proposed framework is evaluated experimentally using a small test-bed prototype. Furthermore, the correct behavior of the proposed framework is verified by analytical study. Finally, conclusions on the findings of this work are presented.

2 Background and Related Work

There has been extensive research into the development of frameworks for mobile-hosted Web services. Most of the frameworks implemented are based on the Simple Object Access Protocol (SOAP) using either a centralized [17],[10],[19] or a non-centralized approach [27],[1],[22]. Some research has attempted to reduce the load on mobile Web service hosts using different techniques such as partitioning the execution tasks of the main host and offloading some of the resource-independent tasks to run on

powerful stationary nodes. For example [9] and [8] propose a partitioning technique to execute complex mobile Web services. This partitioning mechanism allows the compensation for the lack of mobile processing resources. However, this approach lacks the adaptation to other resources such as battery power. Moreover, it fails to meet an essential mobility prerequisite of mobile Web service hosts, which is providing services directly from MH where the proposed frameworks in [19],[17],[9],[8]; clients send/receive requests/responses to/from mobile host indirectly through a stationary intermediate node. Furthermore, this approach does not support seamless provisioning of mobile Web services. Another adaptation technique that allows continuous provisioning of services is to transfer the Web service to another auxiliary host when the mobile host's battery power comes to its end. For example, the Modular Hosting Web Services architecture described in [21] contains built-in modules to support continuous provisioning of mobile Web services in P2P network environments. However, this approach does not address the issue of lightweight processing of complex MWS and lacks the adaptation to processing power of the auxiliary mobile host. Also, it does not consider the services that depend on the current context of the mobile host. In addition both approaches support only SOAP-based Web services that require heavy weight parsers and large message payloads, which in turn degrade the overall MH performance.

Recent research studies focus on building resource-aware Mobile Host Web Service Frameworks (MHWFs) for hosting RESTful-based services. The concept of providing RESTful-based Web services from mobile devices has been introduced in [2]. RESTful- and SOAP-based MHWFs have been implemented in [6] and a detailed comparison between the two developed MHWFs has been carried out. Results show that RESTful- is very promising and more suitable than SOAP-based MHWF for resource limited mobile network environments. However, while REST appears to be the more suitable technology to base a mobile Web services framework upon, there are still many issues to be overcome. One of the most important features a mobile host must achieve is the continuous, reliable and timely provision of the requested service. This needs to be independent if it is a large-scale/heavy-load, or a small/light weight service.

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 and distribution, thus reducing the individual MHs' energy usage and increasing the range and complexity of services that can be executed/provided on MHs. Although, distributing mobile applications is not a new concept and has been previously used for application distribution and load balancing [30],[18] and [4] but it has not been used for offloading the execution tasks of mobile Web services to run remotely on other mobile devices. The next section briefly classifies mobile Web services and describes complex services.

3 Complex Partial Distribution Scheme

Mobile Web service distribution is categorised into three main classes as shown in Figure 1 Contentment Distribution (CD), Simple Partial Distribution (SPD) and Complex Partial Distribution (CPD). These three classes differ by the following parameters: 1. Type of the acquired service. There are two types of services: context-dependent services and context-independent services. Processing context-dependent services requires invoking some parameters that are tightly coupled with the current situation or environment of the MH such as its location, time zone, user preferences, etc. The distribution strategy for these services involves SPD and CPD. On the other hand, the execution of context-independent services does not rely on any parameters that are extracted from the current surrounding environment. CD is an example of a strategy that distributes context-independent services. 2. Complexity level of the service. The complexity level of the service denotes the capability of producing the service by combining

more than one function. For example, some services consist of a single heavy-weight long process that cannot be fragmented; such as requesting a service to do rescaling of a photo and send it to the client. This includes the CD and SPD schemes. Contrary, some heavy-weight services includes multiple functions that can be split and executed on different hosts such as requesting a service to perform an image processing of an image and translation of the text written on it using an optical character recognition tool. This includes CPD strategy. 3. Mechanisms used for distributing Web services. It is common that different types of services and levels of service complexity demand different types of mechanisms for handling service distribution. For instance, CD adopts offloading mechanism while SPD involves offloading and migration mechanisms and CPD includes offloading, migration, decomposing, orchestration and federation techniques The purpose of this research, as mentioned before, is to define and set up the basic building blocks of a system that allows providing adaptive mobile Web services and investigate the system's distribution mechanisms. The main function of these mechanisms is to facilitate continuous provisioning of context-dependent complex services in a distributed execution aspect with efficient levels of performance. Complex MWS are compound services that deplete a relatively significant amount of mobile resources because they are composed of multi-function resource-intensive services. Moreover, the execution of these services is context dependent and tightly coupled with the current environment of the mobile host. Thus, complex services create a cumbersome burden to resource constrained mobile Web service providers that causes degradation of the overall performance. In addition, these services may become unavailable to clients due to the spatio changing that occurs almost consistently to mobile terminals. Besides, the resources of mobile hosts may be exhausted over time and become unable to provide and execute Web services.

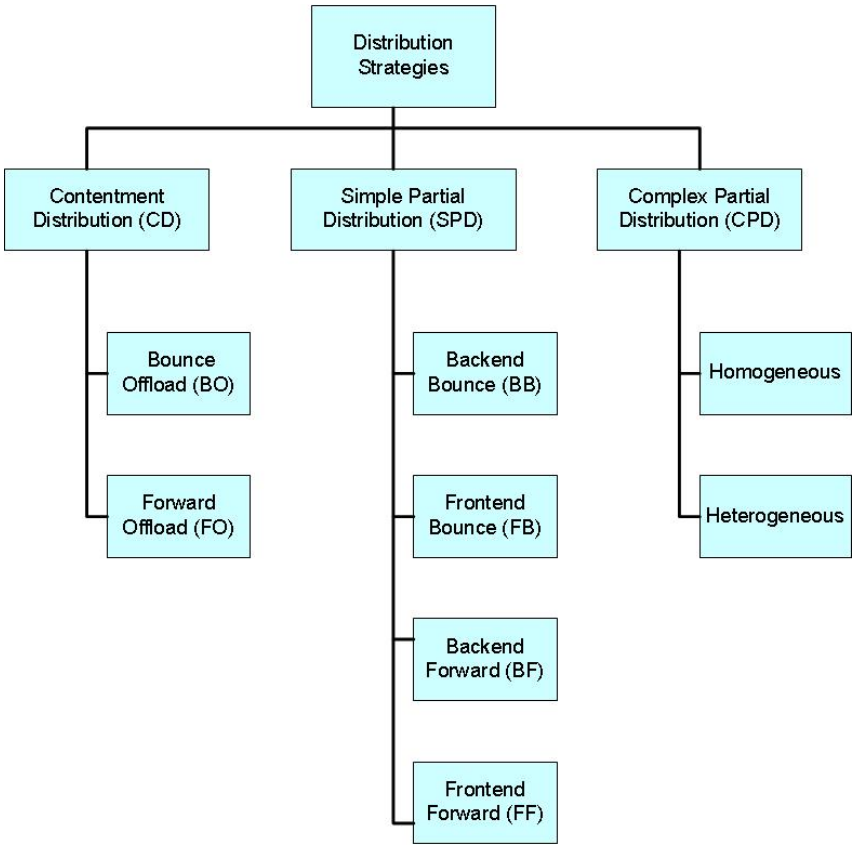

Figure 1: Classification of mobile Web service distribution schemes

Complex Partial Distribution (CPD) scheme is the strategy that is specially used for complex MWS. The identity of the composed functions is used to group a CPD scheme into heterogeneous (HT) and homogeneous (HM) –CPD. In HT-CPD, the involved processes are composed of both serial and parallel ones. However, HM-CPD can be classified based on the dependency factor of distributed services into sequential CPD (S) shown in Figure 1 and parallel (P) HM-CPD, which is illustrated in Figure 2. Dependency factor determines whether there is a correlation and information exchanged between the composite sub-services or there is no interaction. In PHM-CPD, the composite service consists of independent sub-services, which can be executed simultaneously in parallel and independently of each other. On the other hand, in SHM-CPD, the composite service consists of dependent sub-services, which cannot be executed simultaneously. This is because the execution of one service might require some information or resources that are provided by another service. Thus, services are executed sequentially (one after the other).

CPD contains five basic mechanisms: decomposing, offloading, migration, orchestration and data federation. Decomposing is a mechanism that allows fragmenting and partitioning the complex Web service into modules that can be wrapped and converted into simple Web services. Then the execution tasks of each of these partitioned services is distributed and offloaded to be executed remotely on other mobile hosts. Since these services are context-based services, therefore, modelling the transfer of the required context-based data is fundamental and can be achieved through migration. Moreover, the orchestration scheme provides a technique for coordinating the execution workflow among the fragments. Finally, federation is the mechanism that is guardian for data retrieval and error handling. The approach used to apply offloading and migration mechanisms has been described previously in [7]. However, the functionality of the other three mechanisms is described in the following section.

4 Distribution Mechanisms of Complex MWS

CPD is mainly associated with context-dependent complex services. The study of CPD integrates two diverse areas of research that complement each other. The first area is Web service decomposition and the second is Web service composition. In spite of the fact the decompositions and compositions of services have been studied by many researchers. This is accomplished by defining a system that allows the collaboration of a cluster of several mobile devices in a distributed scheme to provide complex services in a manner that does not deplete the devices limited resources too much. The functionality of this framework is based on distributed management and task distribution mechanisms. In addition, the framework technology relies on a RESTful-based approach. It is also established based on two basic fundamentals. The first relies on the fact that REST has a well-defined hierarchy URI scheme for naming the acquired resources. The second predicates on the usage of HTTP uniform methods for invoking services. These two fundamentals are used to allow for uncomplicated and autonomous distribution mechanisms. The outcome of these two fundamentals is a unified skeleton used for building the service pattern, which matches the naming structure used for invoking the service URI. Hence, the partitioning of MWSs is based on the assumption of a well-defined structure for the developed services. The main objective of the well-defined service pattern is to allow flexible and autonomous partitioning mechanism of the service logic. The interface of RESTful Web services consists of a set of resources that are defined by URIs. REST has a conventional style for representing resources that are acquired by the services. Since the services are hosted on a Web, then Web's RFC URI syntax [11] For example, the address of the hosted Web is used as the base address of the service such as: "http://uk.olympic.com/London". Furthermore, path variables are used to separate element of a hierarchy [18]. For instance, suppose a snapshot of all the games played in the Olympics is required to be retrieved, then they are represented as a collection URI such as; "http://uk.olympic.com/London/games". On the other hand, if a specific game is required, then it can be tagged as an element URI as follows: "http://uk.olympic.com/London/games/ gamename" and

so on. The main steps and the techniques for distributing the execution of complex services are described in the following subsections.

4.1 Partition Complex Service into Fragmented Services

In spite of the fact that service decomposition and orchestration has been studied by many researchers in the context of applications that is used to partition the execution tasks of these applications to achieve load balancing and lessen the load on processes [29],[13]. However, service decomposition is a new technique for Web services. Most of the algorithms that are used for segmenting the applications are based on graph theory [15]. These graph based algorithms are either very complex or they intend to partition the application at design time to only achieve minimum communication cost. In contrast, the decomposition of MWS has different constraints and objectives that make achieving it more challenging than its peer, which is application partitioning. Some of these challenges are:

- Web services are provided and executed on machines with heterogeneous platforms while applications are partitioned to run on machines with similar platforms.
- Web services are accessed and interacted by machines but applications are accessed by users, thus anonymous distribution is a must.
- The trade-off between the frequent changes in the amount of resources available in mobile network environment and the key objective of decomposing MWS, which is to preserve the scarce resources of the mobile host, requires adaptive decomposition algorithms.
- Handling Web service requests requires a series of uniform and well-defined events that can be predicted.

Business/service logic is defined as a set of processes. Each process consists of a sequence of related tasks that produce a particular sub-service to the client. The business logic for implementing our service has to be consistent with the naming structure that has been described above. These processes are structured in a specific pattern that conform modularity and consistency and the collection of these processes composes the main service. The business logic for implementing this service has to be consistent with the naming structure that has been described above. The modular structure facilitates fragmenting these tasks into finer-grained subtasks that are likely to appear as subservice modules. This takes place to allow execution tasks of these sub-services and expose them as Web services to be executed on different mobile devices called Secondary Auxiliary Mobile Hosts (SAMH). Therefore, the structure of the developed Web service S consists of a set of atomic modules (processes) P1, P2...Pi that can be decomposed by a decomposer. Then each Pi can be wrapped and translated into a single service Si through projecting a transformation function f (P1), f (P2)...f (Pi). Each of the created sub-services Si is dispatched and executed on different secondary auxiliary mobile hosts SAMH1, SAMH2...SAMHi through the distributor. Some of these dispatched services are dependent of each other and some are self-dependent thus an orchestrator is obligatory to coordinate the execution of these modules and integrate the results into a single output response. Proving the correctness of decomposing and composing methodology is an essential step towards applying our approach. Formal methods are used for verification purposes. Model verification is a technique used to determine that a model implementation accurately represents its developer's conceptual description using formal methods. It has been declared previously in the literature [16] that Web services are best described using process algebra formal languages such as Concurrency Sequential Processes (CSP). The valid behavior of our partitioning scheme is carried out through applying the following theorem [25], which states that:

A process Q2 is a trace refinement of a process Q1 if and only if every trace communication that P2 perform is also a trace of P1 And it is expressed as:

$$Q1 \subseteq_T Q2 \tag{1}$$

Assuming that our service is composed of four well known processes: P1, P2, P3, P4 that can be translated into four Web services: S1, S2, S3 and S4, where S2 and S3 can be executed in parallel but the execution of S4 depends on the output of S2 and S3. First, the un-partitioned system is built as a CSP specification (Spec) as shown in Figure 2. Then the system with fragmented sub-services is built as CSP implementation (Impl) as shown in Figure 3. The CSP-M code implementation is listed in Table 1. After that the implementation is verified, that it is a trace refinement of the specification using Failure Divergence Refinement (FDR2) tool as shown in Figure 4. It has been shown that some of the execution tasks of the complex service are partitioned and executed remotely on different hosts called Auxiliary Mobile Host (AMH) such as invoking the service and processing its logic.

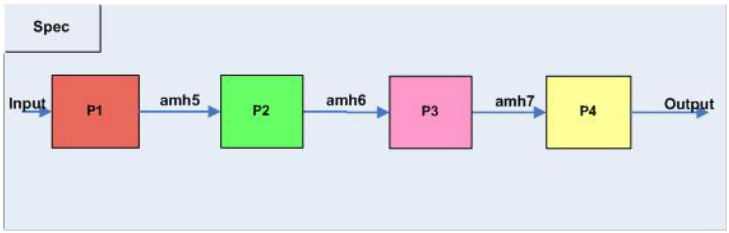

Figure 2: CSP specification for the decomposed services

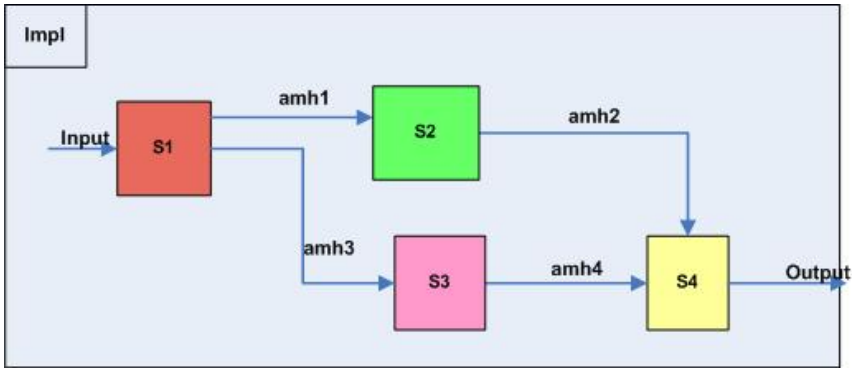

Figure 3: CSP implementation for the service with no partitioning

In addition, the execution of the complex service logic is fragmented and each fragment is wrapped as a simple service and transferred to another mobile host called Secondary Auxiliary Mobile Host (SAMH) for further processing. However, the d-centralized system is not able to provide an integrated service without organizing a conversation between the offloaded execution tasks. As a result, an orchestration mechanism is needed to define policies that link between these distributed services, managing the execution workflow among these services and coordinating the traverse of data between them.

4.2 Coordinate between Distributed Services

Orchestration in mobile environment is a challenging because of the intermittent network connections that causes an interaction failure between the interacted composite services. Therefore, a technique for

Table 1: CSP model of decomposed and composed complex service

```
aaA2={|aA2|}
aaA1={|aA1|}
aaA3={|aA3|}
aaA4={|aA4|}
--The Implementation of the composite services
Impl = let
START = SYSTEM1
SYSTEM1=EQUATION
EQUATION = P1;P2[Union({aaA1,aaA2})||Union({aaA3,aaA4})]P3;P4
P1=  ainput?input->aA1!a1->aA2!a2->SKIP
P2=  aA1?a1->aA2!a2->SKIP
P3=  aA3?a3->aA4!a4->SKIP
P4=  aA2?a2->aA4?a4->aoutput!output1->SKIP
within START
-- The Specification of the composite services
Spec = let
START = SYSTEM
SYSTEM= S1;S2;S3;S4
S1= ainput?input->aA5!a5->SKIP
S2=aA5?a5->aA6!a6->SKIP
S3=aA6?a6->aA7!a7->SKIP
S4=aA7?a7->aoutput!output2->SKIP
within START
-- The implementation with hiding events that do not exist
   in the specification code
Impl2  = Impl \ {aA1.a1,aA2.a2,aA3.a3,aA4.a4,aoutput.output1}$
```

handling such failures is needed to achieve reliability. Moreover, the constrained resources on mobile devices obstruct the processing of the conventional heavy-weight Business Process Execution Language (BPEL) workflow. Hence, light-weight orchestrator is required to preserve the restricted resources. Above all, the execution of some services requires input parameters or resources that are outputted from other services at run time. Accordingly, a mechanism for manipulating and assigning data and resources dynamically is needed. The decomposed execution tasks of the distributed services is orchestrated and coordinated by the AMH: It starts the orchestration by mapping the incoming URI request to a predefined URI that has a compatible sequence structure with the logic structure of the invoked complex service. AMH also, generates a workflow that coordinates the distribution and execution of the decomposed services based on the dependency and the homogeneity properties of these services. The orchestration workflow is generated using Reduced-Business Process Execution Language for RESTful services (R-BPEL4REST) and the properties are extracted from a small xml descriptive file, which is called "property.xml". The property file is provided at the instance of deploying the associated Web service. The presence of the property file liberates the workflow language from the restriction of the existence of a particular process-based description language. Each decomposed service is described by three main elements in the property file as follows:

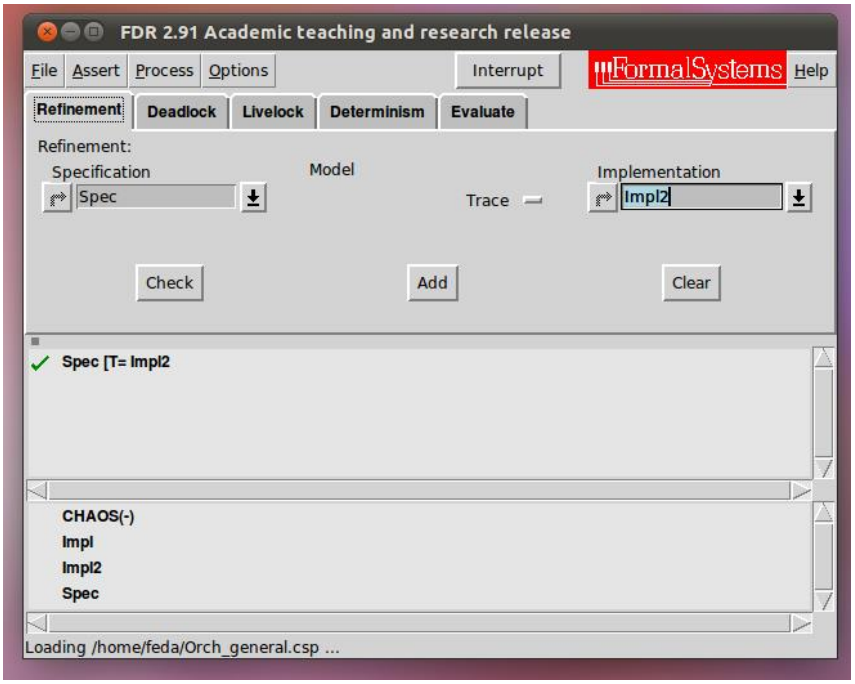

Figure 4: Refinement check of the decomposed and composed services

<name>:Determines the name of the service

<parallel>:Determines the dependency and homogeneity properties. The element's content can take either a true or a false value. A value corresponding to true means that the service can be executed independent of other decomposed services and in parallel. However, false, means that processing this service depends on some results of the execution of other services.

<params>:Determines the parameters or the URI of parameters' resources.

As an example, a sample of a property file is shown in Table 2, which describes the property of a complex location-based image processing service that involves three basic operations as presented in the <name>element: imageScaling to change the image size, imageRotation to tilt the image and imageGrayscale to convert a colored image into balck and white. The sequence order of operations in this scenario is to do imageScaling and imageRotation to a location-based photo. Next, perform imageGrayscale on the resulted image. The value of <parallel>element for imageScaling and imageRotation services is "true" to indicate that these are associative operations and have independent parameters (operands), therefore, imageScaling and imageRotation can be executed in parallel. However, imageGrayscale is performed after extracting the parameters (operands) from the two imageScaling and imageRotation functions. Thus it is a serial operation, which is denoted by setting the associated <parallel>element with "true" value. Furthermore, the locations of the operands are in paths "imageScaling/imageScaling_result" and "imageRotation/imageRotation_result" as presented in the respective <param>elements. The R-BPEL4REST generates atomic requests and sends them to the appropriate SAMHs.

Table 2: Sample of "property.xml" file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Services>
        <Service>
                <name> imageScaling </name>
                <parallel>true</parallel>
                <params>
                        <param>photo1</param>
                </params>
        </Service>
        <Service>
                <name> imageRotation </name>
                <parallel>true</parallel>
                <params>
                        <param>photo1</param>
                </params>
        </Service>
        <Service>
                <name> imageGrayscale </name>
                <parallel>false</parallel>
                <params>
        <param>imageScaling/ imageScaling_result </param>
        <param>imageRotation /imageRotation_result</param>
                </params>
        </Service>
</Services>
```

4.3 Generating R-BPEL4REST

There are several issues to be addressed regarding the significance of applying R-BPEL4REST language with mobile Web services. The suitability of RESTful Web services to mobile environments is one of these issues because it raises the demand for a general and formal description language to support the composition of RESTful services. There are several existing block-structured or graph-based languages for supporting Web service composition such as WS-BPEL, BPML, WSCI, BPSS and WSFL [28]. However, there are some restrictions that hinder the usage of these languages. One of these restrictions is the inflexible communication model that acts against deploying it on mobile environment. This is because the communication link is represented as partner links in BPEL. The partner link has one sender and at most one recipient, which is bounded to a static endpoint at design time. On the contrary, these endpoints in RESTful services designate the resources' URIs, which are generated at run time. Additionally, RESTful Web services allow the client to negotiate for the representation method of the response through the usage of "Accept" in the request's header, but current languages do not support access to request's header. Similarly, the dynamic set up of the representation of the resources is not supported. Moreover, they all rely on the existence of the Web Service Description Language (WSDL), which is not the case for RESTful-based services. Although the emerged WSDL 2.0 version includes other HTTP binding that can be used to describe the interfaces of RESTful Web services, it has not been practically deployed

yet and is given only a weak support by the existing software tools that are mainly arisen to support WSDL 1.1. Besides, the approach of building BPEL on top of WSDL 2.0 creates an unnecessary burden on the BPEL developer to generate interface description for each RESTful service involved and map BPEL through the heavy-weight created WSDL interface. Further, it becomes more complex and cumbersome to maintain WSDL when changes are made to the corresponding service. The design of the R-BPEL4REST implementation is based on the following REST principles:

- RESTful Web services can be invoked and interacted with each other through one of the four fixed HTTP methods (POST, PUT, GET and DELETE) using a synchronous request-response interaction. Therefore, the activities in R-BPEL4REST are monopolised to one of these HTTP Create, Read, Update and Delete (CRUD) methods.

- The interface of RESTful Web services is a set of resources that are accessed by the client. The URI of these resources and their parameters are dynamic and may not be known at design time. Thus, there is no apparent demand for an explicit description of them using WSDL.

- The HTTP header requests and responses provide some important information about the format of the resource representation, access control, caching, content length, etc. Thus, they should be accessed and managed within the R-BPEL4REST using header syntax.

- Statelessness is one of the features of RESTful services that require having connectedness links to allow the hypermedia to act as the engine of application state. Therefore, a mechanism is needed to extract the URI from the response message at run time.

- Services might be invoked from R-BPEL4REST process using one of the CRUD methods. Consequently, R-BPEL4REST provides some of event handlers for the incoming requests that allow the R-BPEL4REST process to perform a set of tasks and to send the proper response.

- POST requests are unsafe and repeated calls to the same resource for this type of requests have to be carried out carefully and isolated to avoid side effects. The structure of the R-BPEL4REST is an extension of the existing BPEL4REST and consists of a small set of constructs as shown briefly in Table 3 and described as follows:

- The <invoke>activity existing in the standard BPEL is replaced by the four activity elements that are used to invoke RESTful services <get><post><put><delete>.

- The four fixed activities are associated with four attributes:

- ”uri”: specify the address of the invoked resource. Since the host that manages and executes R-BPEL4REST is a constrained mobile device, the load on the resources of this device that are required to process R-BPEL4REST should be kept at minimum level. The ”uri” value is extracted from the path directory of resources that is defined in the property file for each associated service. Thus, the value of ”uri” is a static value.

- ”host”: specify the server IP address that is responsible for executing the designated service. The existence of this attribute emancipates the R-BPEL4REST from the inflexible communication model of the BPEL. Subsequently, the generated workflow is capable of modeling the interaction with endpoints at deployment time and not only at design time as is the case for BPEL. In fact, the split of the resource into host and uri results in isolating the logical (virtual) address from its physical location and supports mobility.

- "request": denote the data or parameters that are appended to the body of the request payload. It is required only by <post>and <put>activities;
- "response": denote the data or parameters that are attached to the body of the response message. It resides only on <post>and <get>activities.
- The <header>element is optional and is used to denote the values of the header requests and responses.
- The attribute assigned with the <header>child element is "name", which can take any of the values of the headers' names such as "cache-control", "content-type" etc.
- R-BPEL4REST is equipped to catch failures using a <catch>element and deal with the failure by resending the request or redirecting it to another pre-defined host. R-PEL4REST can also deal with invocation failures but it does not provide a systematic approach for handling the failure.
- The <resource>element identifies a published or existing resource, where it can be invoked, accessed and manipulated by requests within its scope. It has "uri" attribute that contains the address of the designated resource.
- The <onMessage>and <onAlarm>handlers that exist in the standard BPEL are replaced with four handlers <onPost>, <onPut>, <onGet>and <onDelete>to tune with the four HTTP methods used to invoke RESTful-based services.
- The <variable>element is used to allow dynamic binding of data associated with requests and responses. There are three predefined variables $ProcessID, $request and $response that denotes different data information. For instance, $ProcessID is used to store the ID of the running instance of the process to allow safe and isolated functioning of POST requests. In contrary $request and $response are associated with the input/output payload of the request/response, respectively. Also "uri", "response" and "request" attributes may have variable values.
- The <assign>element sets up the values of the variables.

Unlike BPEL4REST that includes publishing resources to the clients, in R-BPEL4REST processes are generated from partitioning the invoked service. Thus, they are known and there is no demand for publishing them as resources for clients. Furthermore R-BPEL4REST does not include data-dependent behavior, hence, the conditional elements such as <while>and <if>are omitted because the flow sequence of execution is predetermined from the service's properties that are deployed in advance. A major difference between BPEL4REST and R-BPEL4REST is the separation of the logical and physical addresses of resources to reflect the frequent changes of the service provider's location. Last but not least, the R-BPEL4REST is generated anonymously based on the existence of a simple property file, but the corresponding BPEL4REST is implemented by the BPEL developer and does not rely on any file description.

Table 4, presents some of the R-BPEL4REST workflow that is generated from the above described complex image processing service. In this example the workflow specifies two request handlers for the resource located in "amh" in a path named "image1": "onGet" and "onPost". Some variables in each service are declared for the incoming requests, outgoing responses and uri locations as "imageScaling_request", "imageScaling_response" and "uri_imageScaling" in the "imageScale" service respectively. These variables can either store the parameter values or the locations of these parameters, which are needed for processing. The "assign" constructor defines values for each of the declared variables. For instance, the request payload for "imageScale" service is initialized with "imageScaling_request" and

Table 3: Constructors used in R-BPEL4REST

Constructor Type	Constructor Name	Brief Description
Definition activities	variable	Define the set of variables whose values are decided upon processing
	process	Defines the top level process (name of complex service)
Invoke activities	Post	Invokes a service using POST method
	Get	Invokes a service using GET method
	Put	Invokes a service using PUT method
	Delete	Invokes a service using DELETE method
Control activities	Flow	Allows concurrent execution of activities
	Sequence	Allows sequential process of activities
Request Handler activities	onPost	Determines the actions occur on receiving POST request
	onGet	Determines the actions occur on receiving GET request
	onPut	Determines the actions occur on receiving PUT request
	onDelete	Determines the actions occur on receiving DELETE
Assignment activities	assign	Assigns data values to the variables

the response output "imageScaling_response" is stored in the "uri_imageScaling" variable. Then, this output is defined to be the value of the first parameter of the "imageGraydcale" service. Moreover, the "imageScaling" and "imageRotation" are executed in parallel through setting a "flow" constructor. After processing these services the last service is processed through setting the "sequence" constructor.

4.4 Federation and Data Retrieval

Federation is the technique concerned for aggregating responses, retrieving the required data parameters and handling execution faults. Creating resource federation and data retrieval relies on the described R-BPEL4REST. Some URI's of the invoked services or resources as we have illustrated in R-BPEL4REST are extracted from the response of the x-invoked service. For example, post request is used to process a resource and create a new resource that holds the output of the process. The response to that post invocation is the URI of the new created resource. R-BPEL4REST introduces a mechanism that allows the dynamic binding between the resources' URI of one service to the response's content of another service. Then, get request is sent to access the bind resource's URI. Moreover, the parameters required for processing a service are extracted from the request's payload of the invoked service or the response's payload of a previously invoked service. The mechanism used for assigning parameters' values depends on the type of the invoked service, whether it is depending or non-depending service. A depending (serial) service requires some parameters that are available after completing the execution of one or more services. On the other hand, a non-depending (parallel) service can be executed independently and it does not rely on some parameters to be generated dynamically after completing the execution of some processes. Resource consistency has to be maintained to assure that the created resource in response to the main service call has a URI path that does not contradict with the requested resource. This is

Table 4: Example of generated R-BPEL4REST for the Image process service

```
<?xml version="1.0" encoding="utf-8"?>
.
.
            <process name="image1" />
            <mrest:resource uri="http:// image/" host="amh" />
<!--Request handlers for the above resource  <onGet> and <onPost > -->
            <mrest:onGet>
                        <mrest:response code = '200'>
                    <mrest:header name  = 'Content-Type'> $request.header.Accept
              </header>
                        <p:Data xmlns:p="http:// image/data " xmlns:xlink="http://www.w3.org/1999/xlink">
                            <a>photo1.png </a>
                        </p:Data>
                  </mrest:response>
            </mrest:onGet>
            <mrest:onPost>
<!--Variable declarations -->
            <bpel:variable name=" imageScaling _request" /> < bpel:variable name=" imageScaling _response" />
            < bpel:variable name="uri_ imageScaling " /> < bpel:variable name="imageRotation_request" />
                  .
          .
          .
<!-- Assign values to the parameters defined above from the incoming requests -->
            <bpel:sequence>
                        < bpel:flow>
                  < bpel:sequence>
                 < mrest: assign>
<! --Initialize the request payload by assigning values to its parameters-->
            $ imageScaling _request_para1 = photo1.png
          < mrest: /assign>
          <mrest: post uri="http:// imageScaling " host="amh1"
            request=" imageScaling _request " response="imageScaling_response" />
            < mrest: assign>
                  $uri_ imageScaling =$ imageScaling _response
            < mrest: /assign>
              < mrest:get uri="uri_ imageScaling " host="amh1"
                 response=" imageScaling _result_response" />
                    < bpel:/sequence>
                    < bpel:/sequence>
              < mrest: assign>      $imageRotation_request_para1 = photo1.png
              < mrest: /assign>
              < mrest:post uri="http://imageRotation" host="amh2"
          .
          .
          < bpel:/sequence>
< bpel:/flow>
          < mrest:assign>
           $ imageGrayscale _request_para1 = $imageScaling_result_response
                  .
          .
         < mrest:/assign>
< mrest:post uri="http:// imageGrayscale " host="amh3"
          .
          .
< mrest:get uri=" uri_imageGrayscale " host="amh3"
            response=" imageGrayscale _result_response" />
                  .
          .
            < mrest:put uri="final_uri" host="amh"     request=" final_request " />
      < bpel:/sequence>
            < mrest:response code="201">
            <header name="Location">
                        http://image/$ProcessID
            </header>
            < mrest:/response>
             < mrest:/onPost>
<bpel:/process>
```

hard to achieve because the decomposed tasks are offloaded on different hosts with different URIs. Two mechanisms are applied to facilitate persisting data and resource consistency. The first one relies on the dynamic binding feature, which allows extracting the required data from the incoming response and

dispatching it to a predefined resource location through the use of the (get + assign + put) combination of activities. The second one is based on having shared database storage structure that is located in the AMH or in the cloud that can be accessed by the orchestrator module. The shared database is static and contains a collection of (URI, resource) tuples, where URI represents the physical location of the resource and resource denotes the abstract data object of the resource itself. A framework is built up to implement the application of the mechanisms, which are needed for enabling the distribution of complex mobile Web services. The main structure of the developed Extended Mobile Host Complex Web service Framework (EMHCWF) is depicted briefly in the following section.

5 Implementation

The main structure of the developed Extended Mobile Host Complex Web service Framework (EMHCWF) is similar to the conventional Mobile Host Web service Framework (MHWF) [6]. However, there are some additional modules, which are appended to extend the functionality of the original MHWF and allow distribution and adaptation of complex mobile services. The EMHCWF consists of nine basic units as shown in Figure 6. EMHCWS consists of the same basic blocks of the MHWF: Web serviceServlet, HTTP Listener, Request Handler, Parser Module, DecisionMaking Module and Response Composer. In addition, it includes the following extended modules: Augmented Offloading Module, Orchestrator Module and SharedData Module.

- 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 customise the particular handling of requests and responses. In addition, it allows the server to check the availability of a requested service before calling it through isWebServiceRegistered() to preserve the system's reliability. Its structure is identical to the previous conventional MHWF but its Hashtable includes an additional field for defining the URI address of the AMH that will execute the acquired service;

- HTTP Listener: The main functions of it include listening to incoming requests, accepting incoming client's requests, initiating a new thread for each request to support concurrency and creating input and output streams for communication. It starts by opening a TCP/IP socket connection through ServerSocket class and waits for any incoming requests. Then, acceptAndOpen() method is called when the HTTP Listener receives a call. After that, a thread is created and data stream connections are opened for exchanging messages using InputStream and OutputStream classes;

- Request Handler: The main tasks of the Request Handler comprise reading/writing data streams using CommonHttpProcessor class, controlling the processing path of the request and setting up the execution environment such as loading service properties. Its functionality depends on the service name and request method. Therefore, it dispatches the request to the message parser;

- Parser module: The main function for it is to get the needed information for invoking a Web service such as the HTTP method name, service URI and some parameters. This information resides explicitly in the request. As a consequence a simple String-based parser is used to distinguish the variant message components and getParameter() method is adopted to return any required parsed components. The extracted service URI information is sent to the ServiceServlet for further validation and then to the Request Handler to generate the orchestrator. The Parser module also includes kXML parser for parsing the body request and any other xml files;

- DecisionMaking module: It is the module that monitors internal and external resources and activates the Augmented Offloading module whenever the MH is overloaded. The functionality of the

decision making module is based on Fuzzy logic. The service is considered as a resource heavy one and needs to be offloaded when the available amounts of resources are not sufficient to execute it. It also decides upon the distribution scheme to be applied based on the available and required resources;

- Augmented Offloading Module: The main function of the Offloading module is similar to its correspondence in the previously described Enhanced Mobile Host Web service Framework (EMHWF) [5], which is to distribute the execution tasks of a mobile Web service and to model the transfer of the required data resources. However, the two Offloading modules of the EMHWF and EMHCWF differ in the amount of the distributed tasks, the distributed depth and data transfer model. For instance, in EMHWF, the execution tasks are comparatively few. Therefore, the distribution depth level is surficial and takes place among the MH and the AMH. On the other hand, in EMHCWS, the number of execution tasks is almost large and a collection of these tasks are organized in units that are decomposable among more than one AMHs. Augmented Offloading module also, extends the basic functionality of the EMHWF through generating a workflow to organize and coordinate the execution sequence of the fragmented tasks and model the data transfer among these fragments;

- Orchestrator Module: This is the vital module in the complex framework, it manages the overall service execution by applying an orchestration mechanism.

- SharedData Module: This is the module that is used for storing and retrieving shared data;

- Response Composer: It is responsible for interpreting the result before sending it back to the client. It can interpret the result using different representations such as XML or JavaScript Object Notation (JSON). The CPD system is verified and validated using two methods: formal methods and proof-of-concept demonstration. The next section presents the evaluation and validation process of this framework.

6 System Evaluation using PROOF-OF-CONCEPT Demonstration

The evaluation of the complex partial distribution mechanisms takes place through proof-of-concept demonstration, which can be performed through testing the EMHCWF's modules that provide these mechanisms (decomposing, orchestration and federation) and examining their functionality.

A test prototype has been implemented for different CPD strategies to give a proof-of-concept demonstration and assist in addressing the issues related to the mechanisms of complex partial distribution.

6.1 Setup Environment

The tested prototype consists of six nodes that are simulated using Wireless Simulation Toolkit: The first entity represents the client, the second one is the MH, the third node interprets the main AMH and the last three represent three SAMHs. The architecture of the MH is implemented with the basic EMHWF and the three SAMHs are implemented with the conventional MHWF. However, the architecture of the main AMH is distinguished from others through using EMHCWF. The interactions between the six nodes are shown in brief in Figure 7. First the client submits the request to the MH after setting the network connections. The MH checks the availability of the service and sends an error message to the client when the service not registered. Otherwise, the MH redirects the request back to the client and the URI of the new selected AMH. The location-based information is sent from the MH to the client upon receiving an acknowledgment signal from the client, which indicates the recipient of the message.

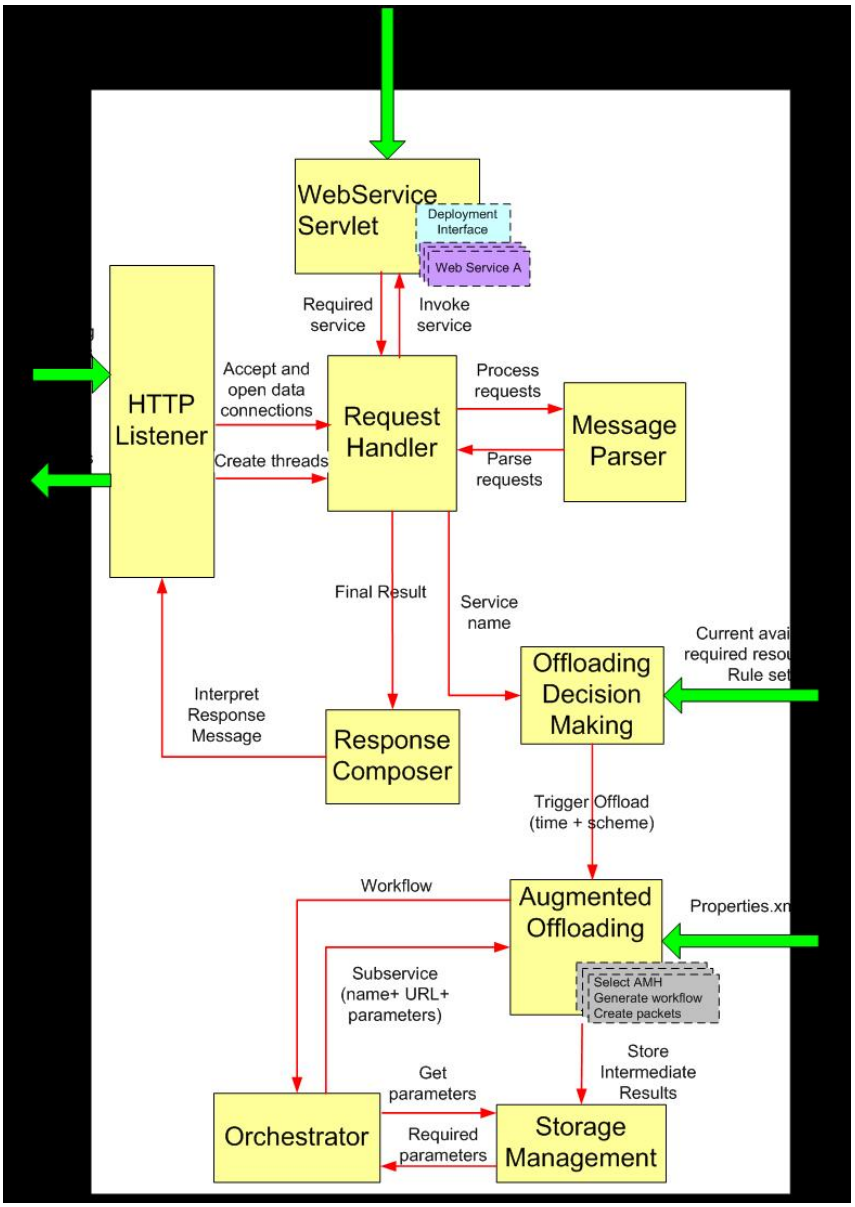

Figure 5: Basic Building Blocks of the EMHCWF

At this stage, the interaction between the client and the MH is terminated and a new network connection is set up between the client and the AMH. After that the AMH checks the availability of the invoked service and sends an error message when the service is unavailable but when the service is available, the AMH generates the coordination workflow, fragments the request and creates packets for each fragment. The AMH sends the first and second independent requests to their designated SAMH simultaneously while the execution of the third request is suspended until the two previous requests complete their execution and store the results in the Shared Data. Then, the AMH aggregates, represents the final response based on the clients' preferences and sends it to the client.

In general, the three frameworks perform a common function that is to allow deploying, providing and executing Web services from mobile devices but the EMHWF allows for applying migration of data and EMHCWF extends the functionality further. The extension includes the distribution of complex

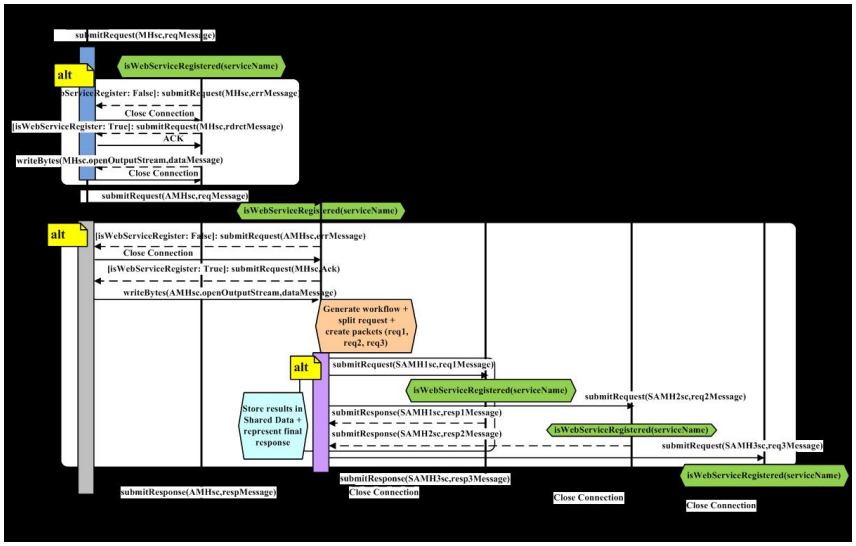

Figure 6: Message sequence diagram for CPD system

Web services and providing the essential mechanisms for this distribution to take place in an anonymous, transparent and independent way. So that human interference is eliminated and client's awareness is avoided. Extending the functionality leads to augmenting the architectural design. The augmentation includes the Orchestration, Shared Data and the Augmented Offloading modules as described earlier.

In implementing this 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). CLDC has been selected because it is a low level specification, suitable for wide range of mobile devices with limited memory capacity. APIs and libraries are added to support more features through Mobile Information Device Profile (MIDP). MIDP 2.0 was chosen because it supports devices with limited network communication and device internal resources. It also provides rich networking functionality and supports the HTTP protocol. In addition it supports the Server Socket connection that is required for implementing mobile server. The cluster of all mobile hosts is assumed to be connected in a high speed and reliable enterprise intranet. The services are deployed in all mobile servers.

A complex service model has been implemented to provide an illustrative case study. The complexity of the service resides in two fundamental issues. The first issue results from having a service that can perform a multifunction. The second one causes from having a resource intensive service that consumes a significant amount of mobile resources. The output of the prototype is compared with the output of a similar prototype that does not involve distribution of the execution tasks. This output is assumed to be identical. The service model is a stringOp service that extracts a string stored in the MH and performs a set of parallel and serial string manipulation functions such as stringReverse, stringUpper and stringConcatenate which are used to present the string in reverse order, change the alphabet case of the string to capital or small letters and expand the string through appending it to itself several times, respectively. The amount of consumed resources can be varied and controlled by the client through determining the number of times that a string is concatenated. Then the response time is measured and compared between serial and parallel CPD schemes. This is to differentiate between PHM-CPD and SHM-CPD.

6.2 Results and Discussion

As mentioned above, the goal of these experiments is to validate the correct behaviour of the defined CPD system with its associated mechanisms and verify its functional requirements through proof-of-concept demonstration. This can be accomplished by developing different types of service models and running each of them on two different architectures: complex distribution framework that incorporates service decomposition and the previously implemented framework that does not involve partitioning the service. Then, the final result is compared between the two, which is expected to be identical. In addition, the amount of distribution overhead is estimated by calculating the difference in response time.

This service is a homogeneous one and includes a sequence of mono task operations, which are examined for both schemes parallel and serial to test the capability of fragmenting a service and executing the fragments concurrently, when possible. The client sends a request to the MH associated with two parameters: “number” and “count”. The “number” denotes the location of the string to be processed and “count” designates the number of iterations that is used in the concatenation subservice.

The correct behavior of CPD is confirmed in the output similarity found for PHM-CPD and non-CPD in Figure 7 and Figure 8. Results elaborated in Figure 9 show that the response time for SPD is the minimum because of the absence of interactions with SAMHs. However, PHM-CPD has more response time with small message size than its corresponding SHM-CPD because of the overhead required for setting up and initiating threads initially. However, in general, parallel execution reduces the total response time and enhances the overall performance with larger message payloads. The second part of our evaluation is based on formal methods verification as described in the next section.

Figure 7: StringOp service prototype for CPD framework

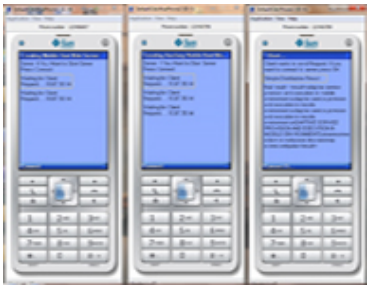

Figure 8: StringOp service prototype for non-CPD framework

7 CSP MODEL OF THE SYSTEM

After that, we create a Communicating Sequential Processes (CSP) model for our system and analyze it to verify the correct behavior of the implemented framework. CSP is the language that is used to describe

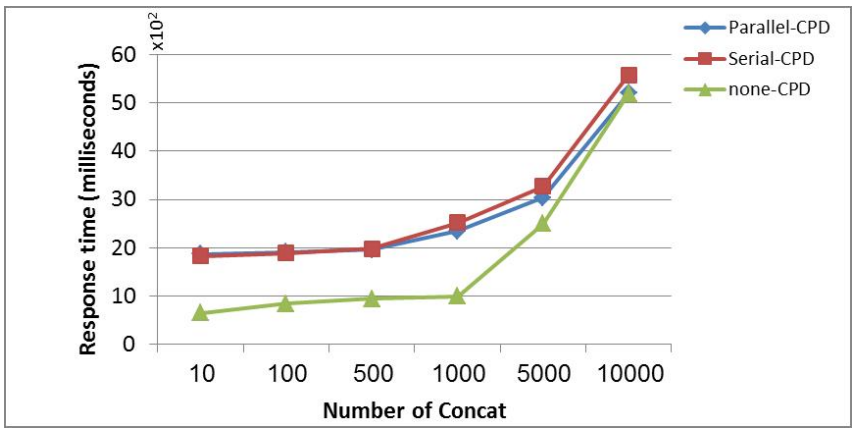

Figure 9: Response time for stringOp service using different strategies

concurrent systems that consist of interacting processes through communication channels. The created model consists of nine basic processes that are transmigrated from the basic building modules. The model focuses on the interaction and communication between the modeled entities. The existence of three similar SAMHs to carry out the execution of the offloaded tasks is assumed. Furthermore, a CSP model of a workflow process thread is added to allow testing the system on different scenarios. The first scenario assumes that the three offloaded services are all parallel and represents PHM-CPD strategy, the second scenario assumes that the three offloaded services are all sequential and represents SHM-CPD, and the last scenario interprets the heterogeneous scheme HT-CPD and assumes that two services are parallel and one is sequential. A parallel combination of the generated processes is created and the model is loaded into FDR2 tool. After that we select a proper semantic model such as failure, failure/divergence or traces. This is because CSP theory is based on a mathematical model that defines a set of observable behavior of the processes and each of these models is suitable for a particular verification method. For instance, the traces model allows checking safety properties. Deadlock analysis is best carried out through the stable-failures model and the failures divergences model is particular for livelock analysis. Finally, the livelock and deadlock properties are examined. The FDR2 converts each CSP model into Labeled Transition System (LTS), which consists of finite-state processes and contains all the semantic representation of the transformed model. As a consequence, any safety property is verified autonomously through an extensive analysis of the LTS by FDR2 tool. [25] has provided some mathematical techniques for reasoning about deadlock of CSP model.

A system is deadlock-free as defined in [26] if and only if it does not reach a state of STOP.
And the system is livelock free if and only if it does not reach a state of unbounded sequence of events.

The model as shown in Figure 10 consists of 10 main processes representing the communicating entities that are interacting with each other through 25 channels. The following assumptions are considered as a basis for this research approach to facilitate the verification procedure and limit its scope:

- First assumption is concerned with setting up the initial state of the system. In general, the system starts the MH and boots it first. Then, the AMH is connected and followed by running the SAMH. After that, the client starts processing. This order of run represents the logical sequence flow of requests. The MH is the main host that first receives client's requests and has to be prepared to listen to any incoming requests.

After that, requests are bounced to AMH that is also booted and listens to incoming requests. Finally, requests are bounced to SAMH.

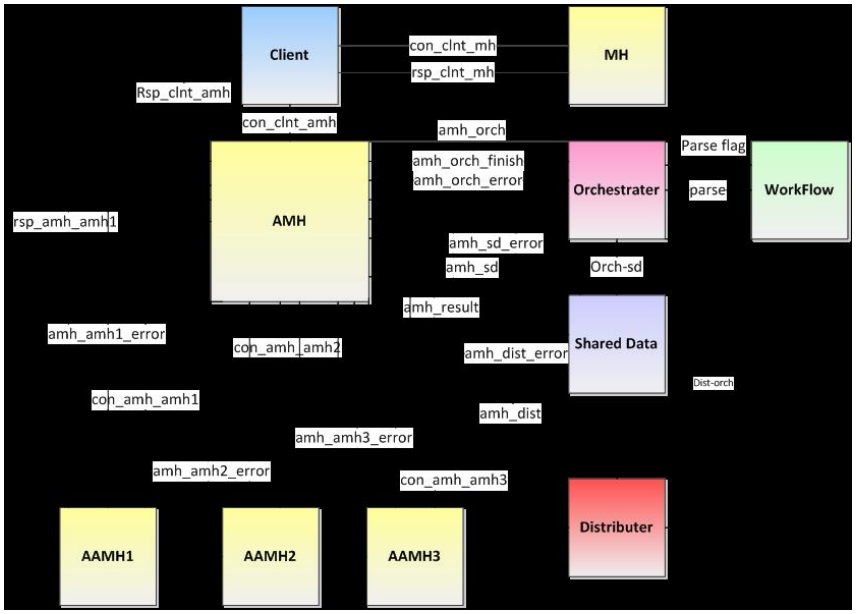

Figure 10: CSP Model of the Complex Mobile Web Service Framework

- Second assumption is related to the maximum number of decomposed tasks that has impact on the number of available SAMHs in the system. This number is restricted to three to avoid state explosion problem of CSP/FDR framework.
- Third assumption is that the MH is initially overloaded and is unable to process incoming requests. Requests are bounced back to the client and the data extraction model is of type frontend.
- Fourth assumption is relevant to the failures' sources and error handlers. There are two types of failures that might occur: connection refusal or service inexistent. The method for handling a failure depends on the error type and the source of error creation.
- Finally, the orchestration workflow is assumed to be generated in advance. It is activated and loaded by the AMH.

Now we present a brief description of each process and its events. CLIENT: The system is initiated by the CLIENT. This is done through sending a connection request to the MH via con_clnt_mh channel then it waits for reply from the MH via the same channel. When the CLIENT receives the reply it acts upon the reply type. If reqConAccept_mh is received then the CLIENT sends the acknowledgment (ack_mh) indicating that the connection is opened and established. The CLIENT also sends a request message (reqMsg) to the MH via the bidirectional con_clnt_mh channel and waits for a response. However, if reqConReject_mh is received then the CLIENT repeats sending requests to setup a connection with the MH. While the CLIENT waiting for a response to be received through the bidirectional rsp_clnt_mh channel it might receive three types of responses:

- rdrctMsg indicating that the request should be bounced to the AMH. The recipient of rdrctMsg is followed by sending a complete signal to the MH indicating that the CLIENT is ready to receive the context-dependent data information and go back to the waiting state through using a recursion technique and wait for the data.
- error indicating that an error has occurred due to unavailable requested service

- data indicating that the context-based parameters for the requested service are received and the CLIENT is ready to request a connection with the AMH

WAITINGRESPONSEMH = rsp_clnt_mh?x →
((x== rdrctMsg)&rsp_clnt_mh!complete -¿ WAITINGRESPONSEMH
[](x==error)& DISPLAYERRORRESPONSE
[](x==dataMsg)& REQUESTAMHCONNECTION)

When the CLIENT starts communicating with the AMH it goes into a series of events that are similar to those ones described for the communication with the MH except that the events occurs on different channels con_clnt_amh and rsp_clnt_amh and there are different kinds of responses from the AMH, as shown below, where data1, data2 and data3 represent responses from AMH1, AMH2 and AMH3, respectively and respMsg represents the final aggregated response.

WAITINGRESPONSEAMH = rsp_clnt_amh?x →
((x == error)& DISPLAYERRORRESPONSE
[](x==respMsg)& DISPLAYRESPONSE
[](x==complete_amh)& rsp_clnt_amh!dataMsg → WAITINGRESPONSEAMH
[](x==data1)& DISPLAYDATA1RESPONSE
[](x==data2)& DISPLAYDATA2RESPONSE[](x==data3)& DISPLAYRESPONSE)

MH: This is the process that models the mobile host, which starts by opening a socket connection and waiting to any incoming requests through con_clnt_mh channel. On receiving openCon_mh event, it might either allow the communication to take place and waits for the ack_mh or it might reject it and go back to the initial waiting state. If the ack_mh is received, then it waits for the request message and starts processing the request. The invoked service might be registered which means deployed and exists or it might not exist. Service executing process depends on the availability of the service. If the service exists, then rdrctMsg is send to the client followed with the corresponding location-based data. However, if the service does not exist, then an error is send back to the CLIENT.

WAITINGREQUEST = con_clnt_mh?reqMsg → PROCESSREQUEST
PROCESSREQUEST = webServiceRegistered_MH_True → WEBSERVICEXECUTION
[]webServiceRegistered_MH_False → WEBSERVICEXECUTIONERROR
WEBSERVICEXECUTION = rsp_clnt_mh!rdrctMsg → WAITINGCOMPLETE
WAITINGCOMPLETE = rsp_clnt_mh?complete → rsp_clnt_mh!dataMsg → CLOSESTREAMCONNECTIONS
WEBSERVICEXECUTIONERROR = rsp_clnt_mh!error → CLOSESTREAMCONNECTIONS
CLOSESTREAMCONNECTIONS = closeStreamConnections_mh → WAITINGANDLISTENING

AMH: This is the process that models the auxiliary mobile host and its events. It starts by opening a socket connection and waiting for incoming requests via con_clnt_amh channel. On receiving openCon_amh event, it might either allow the communication to take place and waits for the ack_amh or it might reject it and go back to the initial waiting state. If the ack_amh is received, then it waits for the request message and starts processing the request. The invoked service might be registered or it might not exist. Service executing process depends on the availability of the service. If the service exists, then AMH sends a complete_amh signal through rsp_clnt_amh channel to the CLIENT indicating that it is ready for receiving the data. When the data is received, it starts processing the request and decomposing as follows:

WAITINGORCHESTRATION = (amh_dist?x → ((x==packet1)&PROCESSINGFIRSTTHREAD
[](x==packet2)&PROCESSINGSECONDTHREAD
[](x==packet3)&PROCESSINGTHIRDTHREAD))
[]amh_orch?done → PROCESSREQUEST

Then the fragmented packets are dispatched into its corresponding SAMH (AMH1, AMH2 or AMH3)

for processing it as shown below. The AMH sends a request for initiating a connection with the designated SAMH as illustrated earlier, which can either be accepted or rejected. Then it waits for a flag to be received from the orchestrator through amh_orch channel that indicates whether the execution of the packet depends on the execution of others (sequence) or it can be executed independently of other packets (flow). If it is flow then AMH sends the required information dataMsg to the SAMH, otherwise it checks for all other dependent packets and waits for their process completion through pardone flag. Then it can send the dataMsg to the SAMH, which processes the packet and sends back the corresponding result. This result is forwarded to the shared data for storing it to maintain consistency.

–Dispatching First request
PROCESSINGFIRSTTHREAD = con_amh_amh1!openCon_amh1 → con_amh_amh1?x → ((x==reqConAccept_amh1) & SENDINGFIRSTREQUEST [](x==reqConReject_amh1) &PROCESSINGFIRSTTHREAD)
SENDINGFIRSTREQUEST = con_amh_amh1!ack_amh1 → con_amh_amh1!packet1 → rsp_amh_amh1?x → if(x==complete_amh1)then WAITINGPARAM1
else WEBSERVICEEXECUTIONERROR1
WAITINGPARAM1 = amh_orch?x → ((x==flow)&rsp_amh_amh1!dataMsg
→ WAITINGFIRSTRESPONSE [](x==sequence)& CHECKDEPENDENCY)
CHECKDEPENDENCY = amh_orch?x → ((x==par2)&CHECKDEPENDENCY
[](x==par3)&CHECKDEPENDENCY[](x==pardone)&rsp_amh_amh1!dataMsg
→ WAITINGFIRSTRESPONSE)
WAITINGFIRSTRESPONSE= rsp_amh_amh1?data1 → amh_sd!data1 -¿amh_orch_finish!finish → WAITINGORCHESTRATION

AMH1: It represents the SAMH1. Since it represents a mobile host, thus, the first steps are similar to any mobile host for preparing its socket connection. When the ack_amh1 is received from the AMH through the con_amh_amh1 channel, the AMH1 waits for the request. Then the AMH1 checks availability of the corresponding service. If it is available and registered then it sends a complete signal indicating that it is ready for receiving the context-based parameters, after that, it starts processing the designated service. Then, it sends the result back to the AMH through rsp_amh_amh1. On the other hand, it might happen that an error has occurred during any stage of processing the main service. Therefore a technique for handling this fault is required. This is accomplished through sending an error message from the main AMH to all the relevant SAMH to allow them go to the initial waiting state. ORCHESTRATOR: This is the process that models the coordinator and its corresponding events. Activating its functionality starts when it is loaded by the AMH and this is expressed through receiving a loadOrch message from the AMH. After that, it starts the orchestration process. This is accomplished through sending a startParsing flag to the workflow to initiate the parse process.

STARTINGAPPLICATION = amh_orch?loadOrch → parseflag!startParsing → PARSINGWORKFLOW1

The first step acted upon initiated the parse process is to extract the type of the fragmented service. Therefore, the parsed data are received through the parse channel, which can be tag_FLOW to indicate parallel service, tag_SEQUENCE for sequential service or tag_END to represent the end of the workflow. Then, the orchestrator sends the name of the service to the distributor and the distributor on the other side replies with the corresponding keyPrefix of the service, which is its URI. The interaction with the distributor process takes place through the dist_orch channel. When the orchestrator receives the keyPrefix of the service, it dispatches the packet and its type to the distributor for preparing it to be sent to the corresponding SAMH for further process. The next step depends on the service type. For instance, for serial services, the orchestrator keeps on parsing the required parameters from the workflow until there are no more parameters. Each parsed parameter represents the fragmented service that it depends on. Hence, it is sent to the AMH via amh_orch. Then it waits for the AMH to send its corresponding

termination index (finish) via amh_orch_finish. The orchestrator sends back an acknowledgment to the AMH that it has received the completion index and parses the next parameter. After parsing all the parameters, it continues to parse the other service unless it is reaching the last service, after which it goes back to its initial state. An error might occur any time during the orchestration process. Therefore, a technique for handling the error exists at each process stage that returns the system back to its initial state. This technique depends on the error type and its source. Errors can be either due to non-existence of a service or an execution failure. Sources of the error vary among the different service providers (MH, AMH and SAMH). If it is caused by execution failure then it informs the client to resend the request. However, if the cause of the error is due to the inexistence of the service then it informs the client to allow it to redirect the request to another host. Regarding the source of the error the place where the error occurred is initialized and all other upper levels are also initialized as described later.

GETPARAMREQUEST1 = dist_orch!serviceName → dist_orch?keyPrefix -¿SENDINGPACKETPARALLEL1
GETPARAMSHAREDDATA1 = dist_orch!serviceName → dist_orch?keyPrefix -¿SENDINGPACKETSERIAL1
SENDINGPACKETPARALLEL1 = dist_orch!packet1 → (amh_orch_error?error → STARTINGAPPLICATION
[]amh_orch!flow → amh_orch_finish?finish1 -¿amh_orch_finish1!finish1 → PARSINGWORKFLOW2)
SENDINGPACKETSERIAL1 = dist_orch!packet1 → amh_orch!sequence → PARSEPARA1
PARSEPARA1 = parse?x → ((x==para2)& amh_orch!par2 → amh_orch_finish?finish2
→ amh_orch_finish2!finish2 → PARSEPARA1
[]amh_orch_error?error → STARTINGAPPLICATION
[](x==para3)& amh_orch!par3
[](x==tag_END)& dist_orch!done → STARTINGAPPLICATION)

DISTRIBUTOR: It is the process that interprets the dispatcher; it is activated by receiving the reqMsg and the serviceProperties from the AMH. ServiceProperties is an event to denote the successful loading of the property file. The distributor starts decomposing process through splitURL event then activating the orchestration model by sending setOrchValues via the amh_dist channel and waiting for services. The distributor replies with a keyPrefix when it receives the name of the service through the dist_orch channel, as mentioned previously. After that the distributor waits for packet arrivals to forward them to the AMH through the amh_dist channel. The AMH sends an error message to the distributor via amh_dist_error channel when an error occurs at any level of processing the request. Consequently, the distributor goes back to its initial state and waits for receiving a reqMsg. SHAREDDATA: It is the process that models the storage database of the parameters and responses. . It can be activated through using one of the two methods: the first happens when the orchestrator sends getParam event via orch_sd channel. The second activation method occurs when the AMH sends the responses of the fragmented services through amh_sd channel and the SHAREDDATA continues to store the results until it receives getFinal event from the AMH through the amh_result. If an error is received from the AMH through amh_sd_error channel it goes back to the initial state waiting for activation.

STARTINGAPPLICATION = amh_sd_error?error → STARTINGAPPLICATION[]orch_sd?getParam
→ orch_sd!param → STARTINGAPPLICATION []STORERESULT
STORERESULT = amh_sd?x → ((x==data1)& SETRESP1 [](x==data2)& SETRESP2
[](x==data3)& SETRESP3)
SETRESP1 = (amh_sd_error?error → STARTINGAPPLICATION)
[](amh_result?getFinal → amh_sd!data1 → STARTINGAPPLICATION)[](STORERESULT)

WORKFLOW: It is the input to the system that includes the composed services, their type and their corresponding parameters with a demonstration of the dependency issue of each service. It is activated by receiving startParsing flag from the orchestrator. Then it parses the serviceProperties file and sends the parsed values to the orchestrator for further processing. As with any other module when an error occurs at any step, it handles the error and goes back to its initial state. Three different scenarios of the WORKFLOW model are tested and validated in the system. This is to achieve coherently and justify the correctness of the system in three different situations. The first runs for all parallel services (PHM-CPD), the second runs for all serial services (PHM-CPD) and the last runs for heterogeneous services (HT-CPD). After building the system model, it is loaded into the FDR2 tool and tested. Results show that our system is deadlock and livelock free.

8 Summary and Future Work

This paper proposed a framework for context-based complex mobile Web services. Complex services are heavy-weight context-based composite services that consist of multiple and decomposable functions, which are too complex to be processed on a single mobile host without affecting its core functionality and degrading the overall performance. A general service logic structure has been also developed and defined to facilitate task distribution.

This framework is based on RESTful architecture. The goal of this framework is to mitigate the issue of resource limitation on mobile devices to allow continuous provisioning of services and increasing the overall performance. This is accomplished through distribution mechanisms. The main criteria for the distribution are autonomy, lightness and platform independence. Defining a framework and workflow that address these mechanisms and allow distribute execution in a homogeneous manner is the main goal of this study. The distribution strategy is based on CPD that involves partitioning and dispatching the execution tasks of the main host and the invoked service logic among a set of collaborative mobile hosts. The workflow generated for orchestration relies on using R-BPEL4REST language. This assumption is claimed to allow the coordination of partitions in mobile environment and to take into account the constraints of mobile resources. Since complex services are constituted of multi mono task processes, the distributing scheme of these compound services has been classified into three different strategies: PHM-CPD, SHM-CPD and HT-CPD. This categorization is based on the reliance issue of the assembled tasks for the distributed service. Evaluation of the framework has shown that distributing the execution tasks of mobile Web services through offloading, migrating, fragmenting, orchestrating and federation mechanisms is a must for complex and context-based mobile services to overcome mobile resource constrains. Experimental results have shown that PHM-CPD outperforms other strategies. The safety properties of the EMHCWF have been verified with the assistance of CSP.

There are some issues that have not been explored in this paper such as the authentication and security of the distributed services and collaborative hosts. In addition, centralization is the main aspect of the interface for the orchestration workflow. This is because the coordinator needs to interact frequently with the MH to obtain the required context-dependent information. Since communication is critical in the wireless environment and needs to be at low levels, the workflow is stored and executed by one node. However, distributing the orchestration workflow demands a change to the structure of the proposed system. Further study for the modifications required by distribution and the gain and/or loss attained from distribution may be addressed in the future. Thus, adapting and distributing mobile Web services is a promising research area that needs further investigation.

Acknowledgments

This work was supported by Kuwait Foundation Advancements of Sciences (KFAS) grant P11418EO02.

References

[1] F. Aijaz, S. M. Adeli, and B. Walke. Middleware for communication and deployment of time independent mobile web services. In *Proc. of the 2008 IEEE International conference on Web Services (ICWS'08), Beijing, China*, pages 797–800. IEEE, September 2008.

[2] F. Aijaz, S. Z. Ali, M. A. Chaudhary, and B. Walke. Enabling high performance mobile web services provisioning. In *Proc. of the 70th 2009 IEEE Vehicular Technology Conference Fall (VTC'09), Anchorage, Alaska, USA*, pages 1–6. IEEE, September 2009.

[3] E. Al-Masri and Q. H. Mahmoud. Mobieureka: an approach for enhancing the discovery of mobile web services. *Personal and Ubiquitous Computing*, 14(7):609–620, October 2010.

[4] M. Alrifai, T. Risse, and W. Nejdl. A hybrid approach for efficient web service composition with end-to-end qos constraints. *ACM Transactions on the Web*, 6(2):1–31, May 2012.

[5] F. AlShahwan, F. Carrez, and K. Moessner. Providing and evaluating the mobile web service distribution mechanisms using fuzzy logic. *Journal of Software*, 7(7):1473–1487, July 2012.

[6] F. AlShahwan and K. Moessner. Providing soap web services and restful web services from mobile hosts. In *Proc. of the International Conference on Internet and Web Applications and Services (ICIW'10), Barcelona, Spain*, pages 174–179. IEEE, May 2010.

[7] F. AlShahwan, K. Moessner, and F. Carrez. Providing light weight distributed web services from mobile hosts. In *Proc. of the 9th IEEE International Conference on Web Services (ICWS'11), Washington, DC, USA*, pages 652–659. IEEE, July 2011.

[8] M. Asif, S. Majumdar, and R. Dragnea. Hosting web services on resource constrained devices. In *Proc. of the 2007 IEEE International Conference on Web Services (ICWS'07), Salt Lake City, Utah*, pages 583–590. IEEE, July 2007.

[9] M. Asif, S. Majumdar, and R. Dragnea. Partitioning the ws execution environment for hosting mobile web services. In *Proc. of the 2008 IEEE International Conference on Services Computing (SCC'08), Honolulu, Hawaii, USA*, pages 315–322. IEEE, July 2008.

[10] S. Berger, S. McFaddin, C. Narayanaswami, and M. Raghunath. Web services on mobile devices-implementation and experience. In *Proc. of the 5th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA'03), Monterey, California*, pages 100–109. IEEE, October 2003.

[11] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform resource locators (url). Technical Report 1738, Network Working Group, 1994.

[12] C. Biancalana, F. Gasparetti, A. Micarelli, and G. Sansonetti. An approach to social recommendation for context-aware mobile services. *ACM Transactions on Intelligent Systems and Technology*, 4(1):1–32, January 2013.

[13] E. d. V. Noguera and M. R. Pedruelo. A survey on web service discovering and composition. In *Proc. of the 4th International Conference on Web Information Systems and Technologies (WEBIST'08), Funchal, Madeira-Portugal, LNCS*, pages 135–142. Springer-Verlag, May 2008.

[14] O. DOSPINESCU and M. PERCA. Web services in mobile applications. *Informatica Economică*, 17(2):17–26, February 2013.

[15] J. L. Gross and J. Yellen. *Graph Theory and Its Applications*. Chapman and Hall/CRC, 2005.

[16] M. S. Gwen Salaun, Lucas Bordeaux. Describing and reasoning on web services using process algebra. In *Proc. of the 2004 IEEE International Conference on Web Services (ICWS'04), San Diego, California, USA*, pages 43–50. IEEE, July 2004.

[17] J.-s. H. Jae-won Lee. Server and method for providing mobile web service. Technical Report US 8352580 B2, Samsung Electronics Co., Ltd., 2013.

[18] Y. Jin, J. Jin, A. Gluhak, K. Moessner, and M. Palaniswami. An intelligent task allocation scheme for multihop wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 23(3):444–451, June 2011.

[19] L. Li. An integrated web service framework for mobile device hosted web service and its performance analysis. In *Proc. of the 10th IEEE International Conference on High Performance Computing and Communications (HPCC'08), Dalian, China*, pages 659–664. IEEE, September 2008.

[20] S.-P. Ma, W.-T. Lee, and C.-H. Kuo. Location explorer with information services: A mobile application to deliver locationbased web services. In *Proc. of the 2nd IEEE International Symposium on Next-generation Electronics (ISNE'13), Kaohsiung, Taiwan*, pages 283–286. IEEE, February 2013.

[21] K. Mohamed and D. Wijesekera. A lightweight framework for web services implementations on mobile devices. In *Proc. of the 2012 IEEE First International Conference on Mobile Services (MS'12), Honolulu, Hawaii*, pages 64–71. IEEE, June 2012.

[22] S. Murtaza. Implementation and evaluation of a json binding for mobile web services with ims integration support, December 2011.

[23] S.-A. Ong. A mobile webserver-based approach for telemonitoring of measurement devices. In *Proc. of the 4th International Conference on Mobile System, Applications, and Services (MOBISYS'06), Uppsala, sweden*, pages 1–2. ACM, June 2006.

[24] J. Risson and T. Moors. Survey of research towards robust peer-to-peer networks: search methods. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 50(17):3485–3521, December 2006.

[25] A. Roscoe. *Theory and Practice of Concurrency*. Prentice Hall; 1 edition, 1997.

[26] S. Schneider, J. Davies, D. M. Jackson, G. M. Reed, J. N. Reed, and A. W. Roscoe. Timed csp: Theory and practice. In *Proc. of the REX Workshop (REX'91), Mook, Netherlands, LNCS*, volume 600, pages 640–675. Springer-Verlag, June 1991.

[27] S. N. Srirama, M. Jarke, and W. Prinz. A mediation framework for mobile web service provisioning. In *Proc. of the 10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW'06), HongKong, China*, pages 1–14. IEEE, October 2006.

[28] W. M. van der Aalst, M. Dumas, and A. H. ter Hofstede. Web service composition languages: Old wine in new bottles? In *Proc. of the 29th EUROMICRO Conference (EUROMICRO'03), Belek-Antalya, Turkey*, pages 289–305. IEEE, September 2003.

[29] J. Xu, K. Chen, and S. Reiff-Marganiec. Using markov decision process model with logic scoring of preference model to optimize htn web services composition. *International Journal of Web Services Research*, 8(2):53–73, April-June 2011.

[30] K. Yang, S. Ou, and H.-H. Chen. On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. *IEEE Communications Magazine*, 46(1):56–63, January 2008.

Author Biography

Feda AlShahwan is currently an Assistant Professor at the Electronic Engineering Department/Computer Section of the College of Technological Studies in the Public Authority for Applied Education & Training. She has a diverse research interest in Mobile Web Services and their applications. Dr. AlShahwan was born in Kuwait, obtained her B.Sc., M.Sc. in Computer Engineer from Kuwait University 1992, 2004 respectively. Her Ph.D. degree was in "Adaptive Service Provision and Execution in Mobile Environments" from Centre for Communications Systems Research in University of Surrey. The current research interests included studies of Adaptive Mobile Web Services, Social networks and Mobile Cloud Computing.

Maha Faisal was born in Kuwait. She earned her B.Sc. and M. Sc. in computer engineering from Kuwait University at 1997 and 2000 respectively and her Ph.D. from the University of Colorado at Boulder, 2005. Dr. Faisal is currently an Assistant Professor at the Computer Engineering Department, Kuwait University. Her research interests include: human computer interaction, social networks, mobile computing and software system modeling.

Abdullah M. Almeshal was born in Kuwait. He received his PhD in 2013 in Automatic Control and Systems Engineering, University of Sheffield, UK. The M.Sc was in Automatic Control and systems engineering, University of Sheffield and B.Sc in Electrical Engineering from Kuwait University. His contributions include a number of published books and academic journals in various disciplines of his research interests as well as has been an invited speaker in International conferences. His current research interests are in nanotechnology, neural networks, robotic vehicles modelling, nonlinear control, fuzzy logic control, optimisation and operations research, mobile communications and data networks. Currently he is an assistant professor at Electronics engineering technology department, College of technological studies, Public authority for applied education and training (PAAET), Kuwait.

Tarek Selmi was born in Kairouan (Tunisia). He received the B.Sc. degree from Tunis University of Sciences in 2002, the M.Sc. degree from Monastir University of Sciences in 2007 and the Ph.D degree from the National Engineering School of Sfax, Tunisia in 2013. He worked as quality control Engineer at the Tunisian Telecom Electronics company (TTE) for several years before joining the Tunisian Ministry of High Education in 2002 and the Oman Ministry of Manpower in 2008 where he worked as a power electronics instructor for five years and served as coordinator of the Electronics department for three years. Currently, Tarek works at the Australian College of Kuwait as Lecturer. His special fields of interest include VHDL-AMS modeling of power semiconductor devices based on silicon carbide (SiC), power electronic systems, microprocessors/microcontrollers and renewable energy systems especially solar photovoltaics.